DEA Dynamique des systèmes gravitationnels

DEA Astrophysique et méthodes associées

PROJET INFORMATIQUE F95 2000/2001

D. Pelat

5 juin 2002

On se propose de résoudre numériquement un problème d'algèbre linéaire avec contraintes de positivité. De telles contraintes surgissent naturellement dans de nombreux domaines de la physique où interviennent des quantités positives comme l'énergie, la température ou comme ici une intensité lumineuse. On trouve des problèmes similaires par exemple en économie où les productions de certains biens de consommation sont des quantités positives.

1 Le problème astrophysique

Le problème que nous vous proposons de résoudre, concerne la synthèse de populations stellaires à l'aide d'une bibliothèque d'étoiles.

Il s'agit de reproduire le spectre d'une galaxie lointaine en ajoutant les uns aux autres, et dans une certaine proportion, des spectres d'étoiles de types donnés. On arrête la somme lorsque le spectre synthétique est identique ou ressemble le plus possible au spectre observé.

En réalité, on ne cherche pas à reproduire intégralement le spectre de la galaxie, mais seulement certaines observables qui révèlent la présence d'étoiles au sein de la galaxie. Ces observables sont les *largeurs équivalentes* de certaines raies d'absorption. Il s'agit donc pour nous d'ajouter des largeurs équivalentes d'étoiles connues jusqu'à représenter au mieux les largeurs équivalentes de la galaxie. Cette opération étant effectuée dans l'espoir que la population stellaire synthétique ainsi construite reflète la population stellaire réelle de la galaxie.

2 Formulation mathématique

L'addition de largeurs équivalentes que nous venons d'évoquer n'est pas une combinaison linéaire des inconnues k_i. Une analyse relativement simple du problème conduit à l'expression des largeurs équivalentes synthétiques $W_{\mathrm{syn}j}$ suivant la formule :

$$W_{\mathrm{syn}j} = \frac{\sum_{i=1}^{n_\star} W_{ji} I_{ji} k_i}{\sum_{i=1}^{n_\star} I_{ji} k_i}\,, \quad j = 1, \ldots, n_\lambda\,, \tag{1}$$

où W_{ji}, I_{ji} sont respectivement les largeurs équivalentes et les hauteurs du continu de l'étoile de type i pour la longueur d'onde λ_j ; k_i est, pour une longueur d'onde donnée,

la proportion de la luminosité de la galaxie due aux étoiles de type i. Finalement $n_\star$ est le nombre d'étoiles composant la bibliothèque stellaire.

De plus, le système (1) est soumis aux contraintes :

$$\sum_{i=1}^{n_\star} k_i = 1\,, \quad k_i \geq 0\,.$$

Ces contraintes expriment le fait que les k_i sont des proportions et qu'il n'existe pas d'étoiles possédant de flux négatifs. Il s'ensuit que le dénominateur de (1) n'est jamais nul et que l'on peut alors écrire le système sous la forme matricielle :

$$\begin{aligned} \mathbf{A}\boldsymbol{k} &= \mathbf{0}\,, \\ \boldsymbol{u}^T\boldsymbol{k} &= 1\,, \\ \boldsymbol{k} &\geq \mathbf{0}\,. \end{aligned} \tag{2}$$

La matrice $\mathbf{A}$ est rectangulaire et a pour éléments :

$$A_{ji} = (W_{\text{syn}j}u_i - W_{ji})I_{ji}\,,$$

où $j = 1, \ldots, n_\lambda$ est l'indice des lignes et $i = 1, \ldots, n_\star$ celui des colonnes. Le vecteur $\boldsymbol{u}$ a toutes ses composantes u_i égales à un, le vecteur $\boldsymbol{k}$ est le vecteur solution, c'est lui qui représente la population stellaire. Les vecteurs $\boldsymbol{u}$ et $\boldsymbol{k}$ sont des vecteurs colonnes, par conséquent le vecteur transposé $\boldsymbol{u}^T$ est un vecteur ligne. Nous vous proposons d'étudier le cas *dégénéré*, c'est-à-dire lorsque la dimension du noyau de la matrice $\mathbf{A}$ n'est pas nulle. Avant de commencer, il faut introduire quelques définitions relatives au traitement des contraintes $\boldsymbol{k} \geq \mathbf{0}$, dites de *positivité*.

2.1 Interprétation géométrique

Les contraintes de positivité s'étudient dans le cadre de la théorie des ensembles convexes. Pour bien suivre notre interprétation géométrique, il est nécessaire d'acquérir quelques notions de cette théorie. En première lecture, il ne faut cependant pas passer trop de temps sur ces concepts.

2.1.1 Enveloppe convexe de vecteurs

On connaît l'enveloppe linéaire d'un ensemble de vecteurs : $\boldsymbol{k}_1, \ldots, \boldsymbol{k}_n$, c'est l'ensemble de toutes les combinaisons linéaires de la forme $\sum_{i=1}^{n} \alpha_i \boldsymbol{k}_i$, où les α_i sont des réels quelconques. L'enveloppe convexe ("convex hull" en anglais) de ces mêmes vecteurs est restreinte aux seules combinaisons linéaires $\sum_{i=1}^{n} \lambda_i \boldsymbol{k}_i$ *positives* : $\lambda_i \geq 0$ et *barycentriques* : $\sum_{i=1}^{n} \lambda_i = 1$.

2.1.2 Convexité dans un espace vectoriel

Un sous-ensemble C d'un espace vectoriel est dit *convexe* si l'enveloppe convexe de tout couple de vecteurs $(\boldsymbol{k}_1, \boldsymbol{k}_2)$ de C appartient aussi à C. Soit :

$$\forall \lambda \in [0,1]\,, \quad \forall \boldsymbol{k}_1, \boldsymbol{k}_2 \in C \;\Rightarrow\; \lambda\boldsymbol{k}_1 + (1-\lambda)\boldsymbol{k_2} \in C\,.$$

Ceci veut dire que le segment de droite sous-tendu par deux points quelconques du convexe appartient aussi au convexe. Nous admettrons la notion intuitive de points

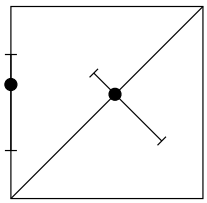

FIG. 1 – La diagonale d'un carré n'est pas une face car il existe un segment du carré dont un point intérieur appartient à la diagonale sans que les points extrêmes appartiennent à la diagonale ; en revanche, les côtés du carré sont des faces.

situés à l'intérieur du convexe et de points situé à la frontière du convexe. Par exemple : le segment de droite situé entre $\boldsymbol{k}_1$ et $\boldsymbol{k}_2$ a pour intérieur tous les points du segment sauf $\boldsymbol{k}_1$ et $\boldsymbol{k}_2$ et pour frontière $\boldsymbol{k}_1$ et $\boldsymbol{k}_2$ (on dit aussi que $\boldsymbol{k}_1$ et $\boldsymbol{k}_2$ sont les points terminaux du segment).

2.1.3 Faces, points extrêmes et facettes

Une *face* d'un convexe C est un sous-ensemble convexe C' de C tel que tout segment de droite de C dont au moins un point de son intérieur appartient à C' implique que ses points terminaux appartiennent aussi à C' (et donc tout le segment puisque C' est convexe). Ceci veut dire que les points d'une face ne peuvent pas être dans l'enveloppe convexe de deux points qui n'appartiennent pas à la face (voir figure 1).

L'ensemble vide et C lui-même sont des faces de C, une face de dimension nulle est un *point extrême* de C. Par exemple, les points terminaux d'un segment de droite sont ses points extrêmes, pour un cube ce sont ses sommets. Un point $\boldsymbol{k}$ de C est donc extrême si, et seulement si, il n'existe pas de combinaison convexe $\boldsymbol{k} = \lambda\boldsymbol{k}_1+(1-\lambda)\boldsymbol{k}_2$ où $\boldsymbol{k}_1 \in C$, $\boldsymbol{k}_2 \in C$ et $0 \leq \lambda \leq 1$, sauf en posant $\boldsymbol{k}_1 = \boldsymbol{k}_2 = \boldsymbol{k}$. Une arête est une face de dimension 1 et une *facette* une face de dimension 2.

2.1.4 Cônes et coins

Un cône est un ensemble K qui est tel que : si le vecteur $\boldsymbol{k}$ appartient à K et si $\lambda \geq 0$, alors le vecteur $\lambda\boldsymbol{k}$ appartient aussi à K. Un cône est *pointé* s'il ne contient pas de sous-espace autre que l'origine (sous-espace de dimension nulle). Un cône qui n'est pas un cône pointé est un *coin*. Dans un cône pointé convexe il existe des vecteurs qui ne peuvent pas s'exprimer en tant qu'enveloppe convexe de vecteurs du cône autres qu'eux-même, ce sont les *vecteurs extrêmes* du cône. La demi-droite engendrée par les multiplications positives d'un vecteur extrême est un *rayon extrême* du cône. L'ensemble des rayons extrêmes d'un cône pointé convexe forme la *carcasse* du cône. Si les rayons extrêmes sont en nombre fini, le cône est un *cône polyèdrique convexe*. Un cône polyèdrique convexe est engendré par les combinaisons linéraires positives de ses vecteurs extrêmes (voir figure 2 page suivante).

2.1.5 Demi-espaces, polyèdres et polytopes

Intuitivement un demi-espace est un ensemble de points situés du même « côté » d'un plan arbitraire. Plus rigoureusement, étant donnés $\boldsymbol{x} \neq \boldsymbol{0}$ un vecteur quelconque de l'espace en question et β un scalaire quelconque, un demi espace est un ensemble

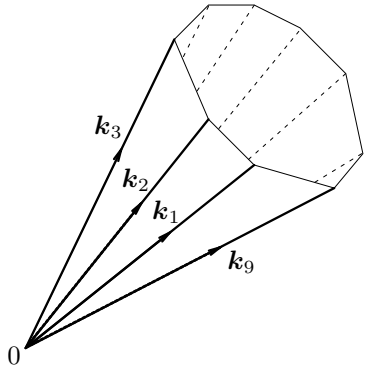

FIG. 2 – Cône polyèdrique convexe pointé engendré par les combinaisons linéaires positives des vecteurs : $\boldsymbol{k}_1, \boldsymbol{k}_2, \ldots, \boldsymbol{k}_9$. Les demi-droites supportées par ces vecteurs constituent les rayons extrêmes de la carcasse du cône.

de points $\boldsymbol{k}$ répondant à l'une des quatre définitions suivantes :

$$\{\boldsymbol{k}|\langle \boldsymbol{k}, \boldsymbol{x}\rangle \leq \beta\}\,, \quad \{\boldsymbol{k}|\langle \boldsymbol{k}, \boldsymbol{x}\rangle \geq \beta\}\,,$$
$$\{\boldsymbol{k}|\langle \boldsymbol{k}, \boldsymbol{x}\rangle < \beta\}\,, \quad \{\boldsymbol{k}|\langle \boldsymbol{k}, \boldsymbol{x}\rangle > \beta\}\,.$$

La notation $\langle \boldsymbol{k}, \boldsymbol{x}\rangle$ désigne, comme d'habitude, une forme linéaire. La première ligne correspond à des demi-espaces fermés et la seconde à des demi-espaces ouverts. Nous ne considérerons que des demi-espaces fermés.

L'intersection d'un nombre fini de demi-espaces est un *polyèdre convexe*, un polyèdre convexe dont aucun point n'est à l'infini est un *polytope*. Par exemple : un cône polyèdrique convexe est un polyèdre, un cube est un polytope. Un résultat important est qu'un polytope est l'enveloppe convexe de ses points extrêmes.

Comme par la suite nous n'aurons affaire qu'à des ensembles convexes, nous omettrons souvent le qualificatif « convexe » et quelquefois aussi ceux de « polyèdrique » et de « pointé ». Par exemple, nous dirons « cône » là où il faudra entendre « cône polyèdrique convexe pointé » ; le contexte ne laisse, en général, pas de place à l'ambiguïté.

3 Solution du problème

3.1 Ensemble des solutions de la synthèse

Plaçons nous dans l'espace E_k des populations stellaires, une solution $\boldsymbol{k}$ est un vecteur de cet espace. L'équation $\boldsymbol{k} \geq \mathbf{0}$ est celle d'un cône pointé, il s'agit, comme on dit, de l'*orthant* positif. L'équation $\boldsymbol{u}^T\boldsymbol{k} = 1$ est celle d'un hyperplan et le système : $\boldsymbol{u}^T\boldsymbol{k} = 1, \boldsymbol{k} \geq \mathbf{0}$ définit un hypertétraèdre régulier que nous notons $S(n_\star)$. Par exemple, si l'espace E_k est de dimension trois (seulement trois étoiles dans la base stellaire) alors l'hypertétraèdre $S(3)$ est un triangle équilatéral (voir figure 3, page suivante), c'est l'intersection de l'octant positif avec le plan s'appuyant sur les vecteurs de la base canonique : $(1,0,0), (0,1,0)$ et $(0,0,1)$. L'appellation « hypertétraèdre régulier » est justifié d'après ses propriétés de symétrie. Cet hypertétraèdre est un autre exemple de polytope.

Les solutions de l'équation $\mathbf{A}\boldsymbol{k} = \mathbf{0}$ forment un sous-espace vectoriel Σ_0 de dimension n_0. Cette dimension n_0 est inférieure ou égale à la plus petite taille de la matrice

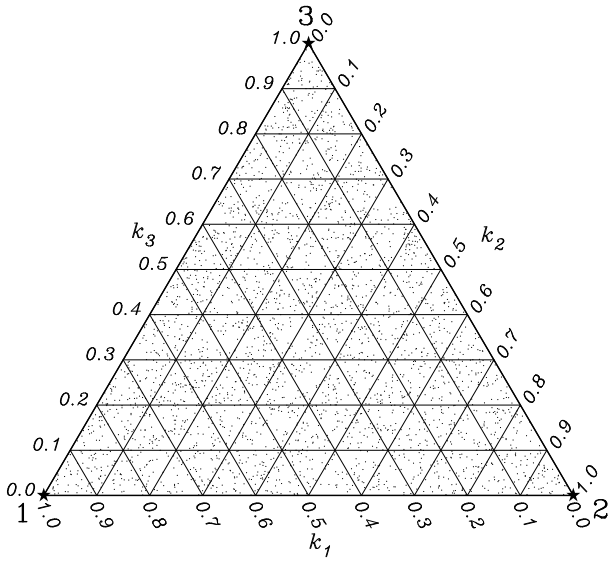

FIG. 3 – Hypertétraèdre $S(3)$ de l'espace $\mathbb{R}^3$. Il s'agit d'un triangle équilatéral sur lequel on a porté le réseau des coordonnées barycentriques : $k_1 + k_2 + k_3 = 1$ avec $k_1 \geq 0, k_2 \geq 0, k_3 \geq 0$.

A. Les solutions du problème sont donc tous les points situés à l'intersection de l'hypertétraèdre $S(n_\star)$ avec l'hyperplan Σ_0. Il s'agit aussi d'un polytope convexe, il peut être réduit à l'ensemble vide si le problème n'admet pas de solution.

3.2 Condition nécessaire et suffisante de « synthétisabilité »

Dire que la galaxie est synthétisable par la base stellaire revient à trouver au moins une solution $\boldsymbol{k}$ du système (2) où, pour calculer **A**, on a remplacé les $W_{\mathrm{syn}j}$ par les largeurs équivalentes observées $W_{\mathrm{obs}j}$. Il existe un moyen simple de vérifier, avant tout calcul, si les largeurs équivalentes de la galaxie peuvent effectivement être synthétisées par la base stellaire. Il suffit pour cela de remarquer que le système (2) est à peu de chose près un programme linéaire, il ne lui manque que la fonction linéaire (d'où vient le nom de programme linéaire) à minimiser. Cette fonction s'appelle la *fonction objectif*. En fait, dans la terminologie de la programmation linéaire, résoudre le système (2) pour au moins une solution revient à trouver une solution dite *réalisable*. Il existe des algorithmes extrêmement rapides basés, par exemple, sur la méthode appelée méthode du simplex qui permettent d'obtenir la solution optimale d'un programme linéaire. En modifiant un peu ces algorithmes il est possible d'obtenir seulement une solution réalisable.

En résumé, pour savoir si la galaxie est synthétisable, il suffit de se donner une fonction objectif quelconque et d'utiliser un algorithme standard ou, de modifier un algorithme déjà disponible de façon à l'arrêter dès qu'il a trouvé une solution réalisable.

D'un point de vue astrophysique, si une galaxie n'est pas synthétisable, cela peut vouloir dire que la base stellaire est incomplète ou que le bruit est si élevé qu'il a complètement dénaturé le spectre de la galaxie au point de le rendre méconnaissable.

3.3 Caractérisation du polytope

Le polytope solution (comme tout polytope) peut être caractérisé : soit par l'enveloppe convexe de ses points extrêmes (sommets) ; soit, de façon duale, par l'intersection des demi-espaces passant par ses facettes. Pour notre problème, il est assez facile de caractériser ce polytope de la seconde façon mais c'est la première qui est physiquement intéressante car les sommets du polytope sont des solutions extrêmes de notre problème, c'est-à-dire des populations stellaires caractéristiques.

Le passage d'une caractérisation à une autre est un problème qui, numériquement, n'est pas résolu en un temps polynômial, il faudra donc utiliser les méthodes les plus rapides appliquées à des problèmes possédant un nombre raisonnable de solutions extrêmes.

Les solutions de l'équation $\mathbf{A}\boldsymbol{k} = \mathbf{0}$ forment ce que l'on appelle le *noyau* de l'opérateur linéaire $\mathbf{A}$, ce noyau est un sous-espace vectoriel noté $\ker \mathbf{A}$. Soit n_0 sa dimension. Si l'opérateur $\mathbf{A}$ était une matrice carrée, une base de $\ker \mathbf{A}$ serait donnée par les vecteurs propres de $\mathbf{A}$ associés à la valeur propre nulle. Mais $\mathbf{A}$ est une matrice rectangulaire, par conséquent il faut procéder autrement. Il existe plusieurs façons d'obtenir une base de $\ker \mathbf{A}$, en voici deux que nous avons expérimentées.

3.3.1 Méthode par diagonalisation

Il est facile de démontrer que $\ker \mathbf{A}^T\mathbf{A} = \ker \mathbf{A}$, c'est-à-dire que l'on trouve une base de $\ker \mathbf{A}$ comme base du sous-espace propre associé à la valeur propre nulle de la matrice carrée $\mathbf{A}^T\mathbf{A}$. Soient $\boldsymbol{k}_1, \ldots, \boldsymbol{k}_{n_0}$ une base de $\ker \mathbf{A}^T\mathbf{A}$ et donc de $\ker \mathbf{A}$, notons $\mathbf{K}$ la matrice de ses vecteurs propres arrangés en colonnes. Un vecteur solution $\boldsymbol{k}$ est une combinaison linéaire positive des vecteurs propres. Soit $\boldsymbol{x}$ le vecteur des coefficients de cette combinaison linéaire, les équations à résoudre sont donc maintenant :

$$\mathbf{K}\boldsymbol{x} \geq \mathbf{0}\,,$$
$$\boldsymbol{u}^T\mathbf{K}\boldsymbol{x} = 1\,.$$

Dans ces expressions, $\boldsymbol{x}$ est un vecteur colonne à n_0 éléments et l'expression $\mathbf{K}\boldsymbol{x} \geq \mathbf{0}$ veut dire que toutes les composantes du vecteur $\mathbf{K}\boldsymbol{x}$ sont positives ou nulles.

L'équation $\mathbf{K}\boldsymbol{x} \geq \mathbf{0}$ est celle d'un cône décrit sous forme duale. En effet, une ligne de $\mathbf{K}$ multipliée scalairement par $\boldsymbol{x}$, soit $\langle \mathbf{K}^i, \boldsymbol{x} \rangle \geq 0$, est l'équation d'un demi-espace dont $\langle \mathbf{K}^i, \boldsymbol{x} \rangle = 0$ est le plan frontière (c'est d'ailleurs un sous-espace). Les vecteurs $\boldsymbol{x}$ définissant les solutions $\boldsymbol{k} = \mathbf{K}\boldsymbol{x} \geq \mathbf{0}$ appartiennent à l'intersection de n_0 demi-espaces, autrement dit à un cône (polyèdrique convexe).

3.3.2 Méthode par décomposition QR

Il existe une façon plus directe d'obtenir une base de $\ker \mathbf{A}$, mais elle fait appel à des notions moins répandues (ce qui est une bonne raison pour en prendre connaissance).

Considérons le sous-espace engendré par les lignes de $\mathbf{A}$, soit $\mathbf{p}$ une base de ce sous-espace et $\mathbf{q}$ une base de l'espace qui lui est *supplémentaire* (c'est-à-dire les vecteurs qu'il faut « ajouter » à $\mathbf{p}$ pour qu'ils forment avec eux une base de l'espace des $\boldsymbol{k}$). Notons E_p l'espace engendré par les lignes de $\mathbf{A}$ et E_q l'espace qui lui est supplémentaire. Un vecteur quelconque de E_q possède, par définition, une composante nulle dans le sous-espace E_p. De plus, si les bases $\mathbf{p}$ et $\mathbf{q}$ sont orthonormées alors ceci veut dire que le produit scalaire d'un vecteur quelconque de E_q par les lignes de $\mathbf{A}$ est nul, autrement dit : $\ker \mathbf{A} = E_q$.

Le processus bien connu d'orthogonalisation de Gram-Schmidt permet, en théorie, d'obtenir une base orthonormée du sous-espace engendré par des vecteurs. Cependant, son emploi n'est pas recommandé car, d'un point de vue numérique, il accumule les erreurs d'arrondis au fur et à mesure de son déroulement. La décomposition QR d'une matrice, disons $\mathbf{M}$, est un procédé bien plus fiable, elle permet, entre autre, d'obtenir une matrice $\mathbf{Q}$ dont les premières colonnes forment une base du sous-espace engendré par les colonnes de $\mathbf{M}$ et les suivantes forment une base de l'espace qui lui est supplémentaire. Toutes ces bases sont orthonormées. Il existe des méthodes QR adaptées aux matrices, dites déficientes, dont le rang est plus petit que leur nombre de colonnes (ce sont celles dont nous aurons besoin).

3.4 Solution par la méthode naïve

Le système $\mathbf{K}\boldsymbol{x} \geq \mathbf{0}$ est un système linéaire homogène de $n_\star$ inégalités à n_0 inconnues . Pour trouver une solution extrême, il suffit de transformer $n_0 - 1$ inégalités en égalités à zéro[1] et d'y adjoindre l'équation linéaire $\boldsymbol{u}^T\mathbf{K}\boldsymbol{x} = 1$ de façon à obtenir un système linéaire de n_0 équations à n_0 inconnues. Si, par chance, la solution de ce système satisfait la condition de positivité alors nous aurons obtenu une solution extrême. Pour obtenir toutes les solutions extrêmes, il suffit de considérer les $C_{n_\star}^{n_0-1}$ façons d'extraire $n_0 - 1$ lignes (linéairement indépendantes) parmi les $n_\star$ lignes composant la matrice $\mathbf{K}$. Cette procédure conduit inévitablement à la solution, mais le problème est que $C_{n_\star}^{n_0-1}$ peut être un nombre fantastiquement élevé. Dans la pratique, il n'est pas rare d'avoir à considérer des centaines de millions de cas.

La méthode naïve est impraticable pour un problème réel dès que $C_{n_\star}^{n_0-1}$ dépasse environ 10^4. En revanche, la méthode naïve, qui est très simple à programmer, permet de valider d'autres méthodes plus performantes comme celle que nous envisageons ci-après.

3.5 Solution par la méthode progressive

Oublions momentanément la contrainte de normalisation : $\boldsymbol{u}^T\mathbf{K}\boldsymbol{x} = 1$. En effet, si l'on trouve un vecteur solution : $\boldsymbol{k} = \mathbf{K}\boldsymbol{x} \geq \mathbf{0}, \boldsymbol{k} \neq \mathbf{0}$ mais pour lequel $\boldsymbol{u}^T\boldsymbol{k} \neq 1$, il sera toujours possible de le normaliser par : $\boldsymbol{k}_{\text{norm}} = \boldsymbol{k}/\sum_i k_i$, car $\sum_i k_i \neq 0$.

On cherche donc les vecteurs extrêmes du cône $\Sigma_0 \cap K_+$, où K_+ désigne l'orthant positif $\boldsymbol{k} \geq 0$. Les vecteurs extrêmes normalisés de la façon que nous venons de voir, seront les solutions extrêmes du problème de la synthèse de populations stellaires.

Désignons par K_n un cône inclus dans Σ_0 (de façon à ce que les vecteurs $\boldsymbol{x}$ correspondant existent) et tel que les n premières inégalités de $\mathbf{K}\boldsymbol{x} \geq \mathbf{0}$ soient satisfaites pour les vecteurs extrêmes de ce cône. Supposons que le cône K_n existe et procédons ensuite par récurrence. Considérons l'inégalité suivante, et examinons la $n+1$-ème composante correspondante des vecteurs extrêmes de K_n. Cette nouvelle inégalité sépare les vecteurs extrêmes en deux classes : (a) la classe k^+ pour laquelle cette inégalité est satisfaite (la composante $n+1$ des $\boldsymbol{k}$ est positive ou nulle) ; (b) la classe k^- pour laquelle elle n'est pas satisfaite (voir figure 4 page suivante). Trois cas peuvent alors se présenter :

1. La classe k^+ est vide, le problème n'a pas de solution et l'algorithme s'arrête là : la galaxie ne peut pas être synthétisée par la base stellaire. Si on a utilisé au

[1] Le fait qu'une solution est extrême si, et seulement si, elle possède au moins $n_0 - 1$ composantes nulles n'est pas simple à démontrer bien que l'intuition géométrique (exercée) l'accepte facilement.

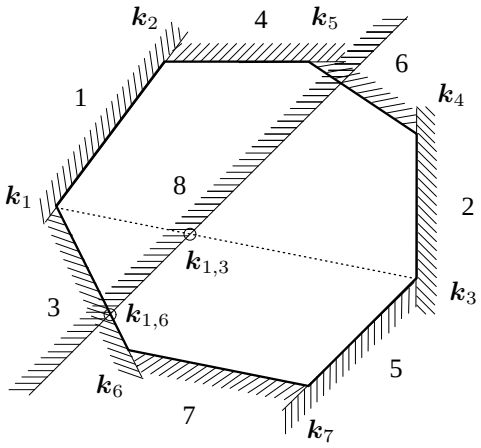

FIG. 4 – Le cône pointé K_7 est défini par les inéquations : 1,2,3,4,5,6 et 7. La nouvelle inéquation numéro 8 sépare les vecteurs extrêmes de K_7 en la classe k^+ qui comprend les vecteurs : $\boldsymbol{k}_3, \boldsymbol{k}_4, \boldsymbol{k}_6, \boldsymbol{k}_7$ et la classe k^- qui comprend les vecteurs : $\boldsymbol{k}_1, \boldsymbol{k}_2, \boldsymbol{k}_5$. Le vecteur $\boldsymbol{k}_{1,3}$ situé sur le segment de droite qui joint $\boldsymbol{k}_1$ à $\boldsymbol{k}_3$ n'est pas sur une facette de K_7 alors que $\boldsymbol{k}_{1,6}$ s'y trouve, c'est un vecteur extrême de K_8.

préalable le test de « synthétisabilité », ce cas ne devrait jamais se présenter.

2. La classe k^- est vide, les $n+1$ premières inégalités sont satisfaites par tous les vecteurs extrêmes de K_n, on pose alors $K_{n+1} = K_n$.
3. Les classes k^+ et k^- ne sont pas vides (ce qui est le cas le plus fréquent), on considère alors tous les couples formés d'un vecteur $\boldsymbol{k}_i$ de k^+ et d'un vecteur $\boldsymbol{k}_j$ de k^-. On cherche ensuite le vecteur $\boldsymbol{k}_{ij}$ de l'enveloppe convexe de ce couple qui soit sur une facette de K_n et tel que sa $n+1$-ème composante soit nulle. La procédure permettant d'obtenir ce vecteur est décrite dans le paragraphe suivant.

3.5.1 Nouveaux vecteurs extrêmes

Dans le cas 3, on vérifie d'abord si $\boldsymbol{k}_{ij}$ est susceptible d'appartenir à une facette de K_n. Un rayon extrême est défini par $n_0 - 1$ inégalités extraites du système $\mathbf{K}\boldsymbol{x} \geq \mathbf{0}$ et transformées en égalités à zéro. Pour simplifier nous dirons simplement qu'un rayon extrême est « défini par $n_0 - 1$ égalités » sans préciser que les égalités sont mises « à zéro ». Dans le même ordre d'idées, une facette de K_n est définie par $n_0 - 2$ égalités. La facette de K_n qui contient $\boldsymbol{k}_i$ et $\boldsymbol{k}_j$ possède $n_0 - 2$ égalités en commun avec celles qui définissent $\boldsymbol{k}_i$ et $\boldsymbol{k}_j$. Par conséquent, le vecteur $\boldsymbol{k}_{ij}$ sera sur une facette de K_n si les égalités qui définissent $\boldsymbol{k}_i$ et $\boldsymbol{k}_j$ sont les mêmes à une unité près, autrement dit si $\boldsymbol{k}_i$ et $\boldsymbol{k}_j$ ont $n_0 - 2$ égalités en commun.

Si tel est bien le cas, il faut ensuite construire le vecteur $\boldsymbol{k}_{ij}$ qui soit dans l'enveloppe convexe du couple $(\boldsymbol{k}_i, \boldsymbol{k}_j)$ (c'est-à-dire sur la facette de K_n qui les contient) et pour lequel sa $n+1$-ème composante soit nulle. De cette façon, ce vecteur sera défini par $n_0 - 1$ égalités, il satisfera $n+1$ inégalités du système $\mathbf{K}\boldsymbol{x} \geq \mathbf{0}$, ce qui signifie que c'est un rayon extrême de K_{n+1}. Il est facile de voir que la définition suivante du

vecteur $\boldsymbol{k}_{ij}$ remplit les conditions demandées :

$$\boldsymbol{k}_{ij} = (\beta_i \boldsymbol{k}_j - \beta_j \boldsymbol{k}_i)/(\beta_i - \beta_j)\,, \tag{3}$$

$$\beta_i = k_{n+1,i} \geq 0\,, \quad \beta_j = k_{n+1,j} < 0\,, \tag{4}$$

où $k_{n+1,i}$ et $k_{n+1,j}$ sont les $n+1$-ème composantes des vecteurs $\boldsymbol{k}_i$ et $\boldsymbol{k}_j$. L'équation (3) exprime bien que $\boldsymbol{k}_{ij}$ est dans l'enveloppe convexe en question et donc dans K_n et l'équation (4) permet d'affirmer que la $n+1$-ème composante de $\boldsymbol{k}_{ij}$ est nulle (en effet, elle vaut : $(\beta_i\beta_j - \beta_j\beta_i)/(\beta_i - \beta_j) = 0$).

Les vecteurs extrêmes du nouveau cône K_{n+1} sont formés des vecteurs extrêmes de K_n qui sont dans la classe k^+, augmentés des $\boldsymbol{k}_{ij}$ que nous venons de calculer par la méthode ci-dessus. L'algorithme s'arrête avec la dernière inégalité, c'est-à-dire lorsque l'on a trouvé les vecteurs extrêmes de $K_{n_\star}$. Ces vecteurs normalisés sont toutes les populations stellaires extrêmes qui permettent de synthétiser les largeurs équivalentes de la galaxie. Toute population stellaire qui est dans l'enveloppe convexe de ces populations extrêmes synthétise aussi ces mêmes largeurs équivalentes.

3.5.2 Le cône initial

Pour que la récurrence puisse démarrer, il faut construire un cône de départ. Là aussi il existe plusieurs méthodes, celle que nous proposons permet un traitement rapide des différentes phases de la méthode progressive.

Il est, en théorie, aisé de construire le cône K_{n_0}, c'est-à-dire un cône tel que les n_0 composantes de ses vecteurs extrêmes soient positives ou nulles. En effet, il est toujours possible d'extraire n_0 lignes linéairement indépendantes de $\mathbf{K}$ car cette matrice, qui est formée de n_0 vecteurs propres linéairement indépendants, est nécessairement de rang n_0. On extrait ensuite $n_0 - 1$ lignes de ce sous système, il y a n_0 façons de le faire. On ajoute enfin la condition de normalisation afin d'obtenir n_0 systèmes linéaires de n_0 équations à n_0 inconnues. Le second membre de ces systèmes est toujours formé de $n_0 - 1$ valeurs nulles (pour définir des vecteurs extrêmes), la dernière valeur valant 1 (afin qu'ils soient normalisés). Les n_0 solutions de ces systèmes forment les n_0 vecteurs extrêmes du cône K_{n_0} qui permet de démarrer la récurrence.

4 Mise en œuvre numérique

On organisera le travail en plusieurs tâches indépendantes sous forme de modules. Ces modules doivent « encapsuler » leur mécanique interne de façon à ce qu'elle soit inaccessible aux utilisateurs terminaux. Par exemple, l'utilisateur n'a pas besoin de savoir que telle méthode de diagonalisation de matrice a besoin d'un tableau de travail de telle dimension, la méthode doit créer dynamiquement ce tableau et le détruire une fois sa tâche accomplie. L'utilisateur ignore tout de ce tableau et n'y a pas accès.

On peut identifier les tâches et les modules suivants :

- Programme principal, coordination du travail des groupes, structure des données (module de déclaration des types).
- Un module d'entrées/sorties des données et de graphique.
- Un module des conditions nécessaires de « synthétisabilité ».
- Un module de résolution du problème par la méthode naïve.
- Un module de résolution du problème par la méthode progressive.
- Un module contenant les opérations d'algèbre linéaire de base.

– Tests, simulations, contrôle qualité.

Nous donnons maintenant plus de détails sur le travail à accomplir.

4.1 Le module de déclaration des types

Il est indispensable de structurer les données afin de faire référence à des types de données mieux adaptés que ceux prévus par le langage de base. Par exemple : une longueur d'onde peut être considérée comme une entité constituée d'un couple formé d'une chaîne de caractères servant d'identification et d'une valeur numérique. Il faudra donc prévoir dans le module de déclaration un type disons « `lambda` » correspondant à cette structure. En Fortran 95, le module de déclaration pourrait ressembler à l'exemple donné figure 5, page suivante.

Si l'entité « `raies` » est un tableau de type `lambda`, on doit le déclarer et l'utiliser d'une manière analogue à ce qui suit :

```
Type(lambda), Dimension(100) :: raies
raies(1)%ID  = "H alpha"
raies(1)%val = 6563.
```

On peut envisager d'autres types comme : `star`, `galaxy` ou `base` comme on le voit sur la figure 5 page suivante.

Les étudiants chargés de l'écriture de ce module doivent aussi réaliser le programme principal.

4.2 Le module d'entrées/sorties et de graphique

4.2.1 La base de données

Une base de données est un fichier `ascii` organisé en lignes, la fin du fichier est signalée par une ligne vide. Un tel fichier peut être créé à l'aide d'un éditeur ou au moyen d'instructions d'écritures formatées. Afin de rendre ce fichier plus lisible, on a prévu des commentaires. Un commentaire commence par le caractère « ! » et se termine en fin de ligne. La figure 6, page 12, est un exemple de base de données. On fournira, suivant le même schéma, une base de données astrophysique réelle.

Il est souhaitable de définir une structure de données de type « base » susceptible d'accueillir les différentes bases de données que nous aurons à traiter. En Fortran 95, par exemple, le type « base » pourrait être semblable à celui donné figure 5 page suivante.

4.2.2 La galaxie observée

Les largeurs équivalentes observées : $W_{\mathrm{obs}j}$ ainsi que d'autres informations complémentaires sont contenue dans un fichier dont l'organisation est semblable à celle du fichier de la base de données. En particulier il contient le nom de la base stellaire qui sert a l'analyser, un exemple de fichier correspondant à une galaxie test est donné figure 7, page 12.

4.2.3 La galaxie synthétique

On pourra calquer l'organisation du fichier résultat contenant les $W_{\mathrm{syn}j}$ sur celle du fichier de la galaxie. Une bonne idée est d'y inclure les date et heure de création

```
Module Types
  Implicit None
  Private
  Public :: SP, DP, lambda, star, base

  Integer, Parameter :: SP=Kind(1.0), DP=Kind(1.0d0)
  Integer, Parameter :: ncc=30

  ! What we mean by 'lambda'
  Type lambda
     Character(len=ncc) :: ID=" "
     Real(SP)           :: val=0.0
  End Type lambda

  ! What we mean by 'star'
  Type star
     Character(len=ncc) :: ID=" "  ! IDentification.
     Character(len=ncc) :: ST=" "  ! Spectral Type.
     Real(DP), Dimension(:), Pointer :: W => Null()  ! Eqwidths.
     Real(DP), Dimension(:), Pointer :: I => Null()  ! Continua.
  End Type star

  ! What we mean by 'DataBase'
  Type base
     Integer :: nl=0  ! Number of wavelengths
     Integer :: ns=0  ! Number of stars in database.
     Type(lambda), Dimension(:), Pointer :: lambda => Null()
     Type(star),   Dimension(:), Pointer :: star => Null()
  End Type base

  ! suivent les autres declarations de types...
End Module Types
```

FIG. 5 – Exemple de déclaration Fortran 95 pour définir certains types nécessaires au programme.

du fichier a été créé, en Fortran 95 on obtient ces informations suite à l'appel : `Call Date_and_time`. Il est aussi conseillé d'y mettre la plupart des données qui ont servi à créer ce fichier, par exemple : le nom du fichier galaxie étudié et le nom de celui de la base stellaire utilisée.

4.2.4 Le graphique

Afin d'acquérir une certaine intuition sur un problème, il s'avère souvent utile de tracer quelques graphiques. En ce qui nous concerne, pour que les graphiques soient facilement compréhensibles, il faudra se limiter à la représentation d'un couple de longueurs d'ondes comme il a été fait sur la figure 8 page 13.

Il existe de nombreux logiciels graphiques, le graphique de la figure 8, par exemple, a été fait à l'aide du logiciel `Pgplot` qui est disponible au Service Informatique de l'Observatoire (SIO). Sur les systèmes `unix` du SIO, pour avoir accès à ce logiciel il faut ajouter l'option `-lpgplot` à l'ordre de compilation, par exemple : `f95 prog.f90 -lpgplot`.

4.3 Le module des conditions nécessaires

Nous n'exposerons pas ici la méthode du simplex, on trouvera un bon exposé dans la section 10.8 pages 423–436 de l'ouvrage "Numerical Recipes" de Press et al. [2]. On pourra utiliser les programmes `simplx, simp1, simp2` et `simp3` de ce livre en prenant garde à ce qu'ils sont en `fortran 77` et en simple précision alors que

```
!
!   DataBase/test
!
          3    ! Number of lines nl
! Lambda ident
'lambda1' 'lambda2' 'lambda3'
! Lambda Angstrom
   1.0    2.0    3.0
!
          8    ! Number of stars ns
! EW and continua.
!
'Test cluster Nr.  1' ''
   3.00    2.00    1.00
   4.00    1.00    1.00
!
'Test cluster Nr.  2' ''
   9.00    3.00    6.00
   1.00    2.00    3.00
!
'Test cluster Nr.  3' ''
   7.00    7.00    8.00
   3.00    1.00    2.00
!
'Test cluster Nr.  4' ''
   4.00    8.00    9.00
   1.00    3.00    3.00
!
'Test cluster Nr.  5' ''
   6.00    4.00    2.00
   1.00    1.00    1.50
!
'Test cluster Nr.  6' ''
   8.50    5.00    3.00
   2.50    2.00    1.50
!
'Test cluster Nr.  7' ''
  10.00    9.00    9.00
   3.00    2.00    1.00
!
'Test cluster Nr.  8' ''
   6.00   10.00    4.00
   1.10    1.20    2.00
```

FIG. 6 – Base de données stellaires artificielle `test.dat` qui peut être utilisée en vue de tests. Ce fichier comporte 44 lignes, la dernière doit être vide.

```
"Test"                                ! Identification.
    3                                 ! Number of lines (nl).
!-lambdaID-   lambda       Wobs   WobsErr
!----------   ------       ----   -------
"lambdaID 1"    0.00       6.00     0.200
"lambdaID 2"    0.00       3.40     0.300
"lambdaID 3"    0.00       3.00     0.300
!
! Name of the database used to synthesize these data:
"test.dat"
```

FIG. 7 – Fichier : `test1.gal`, contenant les valeurs $W_{\mathrm{obs}\,j}$ d'une galaxie artificielle utilisée en vue de tests. Ce fichier comporte 12 lignes, la dernière doit également être vide.

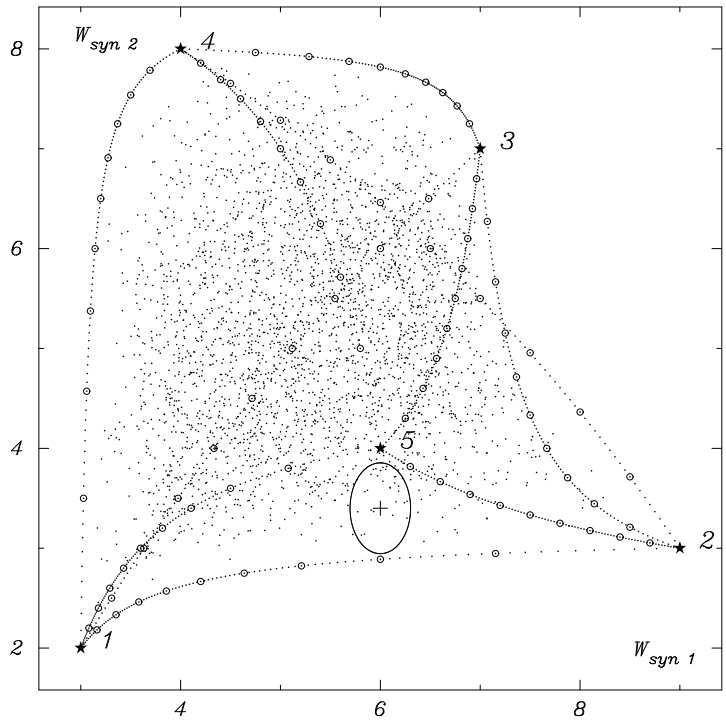

FIG. 8 – Exemple de représentation graphique des 5 premières étoiles de la base de données `test.dat` dans l'espace des 2 premières longueurs d'ondes. La « surface synthétique », c'est-à-dire l'ensemble de toutes les largeurs équivalentes qui peuvent être synthétisées par cette base, est le polygone curviligne rempli de façon non uniforme par des points. Une observation synthétisable est représentée par une croix entourée de son ellipse d'erreur. Cette observation est synthétisable par les étoiles : (1,2,5), (1,2,3) et (1,2,4) ou par tout autre combinaison barycentrique de ces trois solutions. Ces trois solutions sont les solutions *extrêmes* du problème, elles possèdent au moins : $n_\star - n_\lambda - 1 = 2$ proportions stellaires nulles.

nous travaillons en Fortran 95 ou C++ et en double précision. Si le temps le permet, on pourra modifier `simplx` de façon à ce qu'il s'arrête dès qu'une solution réalisable est trouvée.

4.4 La méthode naïve

Cette méthode peut être appliquée directement au système $\mathbf{A}\boldsymbol{k} = \mathbf{0}$ ou au système $\mathbf{K}\boldsymbol{x} \geq \mathbf{0}$ comme il a été expliqué plus haut. Dans le premier cas on n'a pas besoin de calculer une base de $\ker \mathbf{A}$.

Pour les deux méthodes (sans ou avec calcul d'une base de $\ker \mathbf{A}$) on doit être capable de générer toutes les façons d'extraire p éléments parmi n. On doit veiller à ce que *toutes* les combinaisons soient générées et qu'elles ne le soient qu'une *seule* fois. Par exemple, si $p = 3$ et $n = 5$, le premier appel à la procédure doit pouvoir fournir le tableau $\{1, 2, 3\}$, le deuxième appel $\{1, 2, 4\}$ puis ensuite : $\{1, 2, 5\}, \{1, 3, 4\}, \{1, 3, 5\}$, $\{1, 4, 5\}, \ldots, \{3, 4, 5\}$.

4.4.1 Méthode directe

Il suffit d'extraire $n_\star - 1$ lignes du système $\mathbf{A}\boldsymbol{k} = \mathbf{0}$ et d'ajouter la normalisation de façon à obtenir un système linéaire de $n_\star$ équations à $n_\star$ inconnues. Le système sera résolu par la méthode classique de Gauss (décomposition LU). Le programme devra être capable de reconnaître si la sous-matrice ainsi extraite est bien inversible et si la solution trouvée est effectivement positive. Pour ce dernier cas, le problème est de décider quand un nombre proche de zéro est positif. Il est fort possible, en effet, que les erreurs d'arrondis rendent négatif un nombre qui mathématiquement est nul ou proche de zéro mais positif. Il faudra admettre une certaine tolérance que l'on tâchera de justifier.

4.4.2 Calcul de la matrice K par diagonalisation

Le calcul de $\mathbf{K}$ est nécessaire pour appliquer la deuxième manière de la méthode naïve et il est obligatoire pour mettre en œuvre la méthode progressive.

La matrice $\mathbf{A}^T\mathbf{A}$ à diagonaliser est une matrice carrée symétrique définie non-négative (c'est-à-dire que ses valeurs propres sont positives ou nulles). Il faudra employer des méthodes de diagonalisation adaptées à ce genre de matrices. Ici aussi la difficulté est de reconnaître quand une valeur propre doit être considérée comme nulle. De nouveau il y a fort peu de chances que des valeurs propres *mathématiquement* nulles soient trouvée *numériquement* nulles. Pour tenter de discerner les valeurs propres nulles, on s'aidera d'une estimation *a posteriori* basée sur un résultat exposé par Wilkinson concernant les matrices symétriques (voir [3] §49 p.139).

Soit $\mathbf{M}$ une matrice réelle symétrique, λ une valeur propre calculée et $\boldsymbol{x}$ le vecteur propre calculé correspondant. On construit le vecteur résidu : $\boldsymbol{\eta} = \mathbf{M}\boldsymbol{x} - \lambda\boldsymbol{x}$. Si λ et $\boldsymbol{x}$ étaient exacts $\boldsymbol{\eta}$ serait nul. Soit ϵ la norme euclidienne du résidu : $\epsilon = ||\boldsymbol{\eta}||_2 = (\boldsymbol{\eta}^T\boldsymbol{\eta})^{\frac{1}{2}}$, on montre qu'il existe au moins une valeur propre exacte λ_i dans l'intervalle : $\lambda \pm \epsilon$ (c'est-à-dire, $\min_i |\lambda_i - \lambda| \leq \epsilon$). Si zéro est dans cet intervalle on peut alors avancer que λ est la valeur calculée d'une valeur propre qui est en fait nulle. Ce test peut être pris en défaut, en effet le calcul de ϵ n'est lui-même pas exact et il peut y avoir une valeur propre très petite mais non nulle dans l'intervalle en question, mais il n'y a pas grand-chose de mieux à faire. Quoi qu'il en soit, on aura intérêt à vérifier que la

dimension de $\ker$ **A** n'est pas plus petite que la valeur absolue de la différence entre le nombre de lignes de **A** et son nombre de colonnes.

4.4.3 Calcul de K par décomposition QR

La factorisation QR d'une matrice m-par-n **M** est donnée par $\mathbf{M} = \mathbf{QR}$ où **Q** est une matrice orthogonale m-par-m et **R** une matrice triangulaire supérieure de même format que **M** (voir Golub et Van Loan [1] §5.2 et §5.4). Si m, le nombre de lignes, est plus grand que n, le nombre de colonnes, et si les colonnes de **M** sont linéairement indépendantes, c'est-à-dire si le *rang* de **M** est égal à n, alors on montre que les colonnes de **Q** forment une base de l'espace *image* de **M**. En d'autres termes, la décomposition QR de **M** permet d'obtenir une base orthonormée de l'espace engendré par les colonnes de **M**.

Si les colonnes de **M** ne sont pas linéairement indépendantes (le rang de **M** est inférieur à n) alors il faut faire appel à une variante de la décomposition QR adaptée au cas où la matrice **M** est, comme on dit, de rang déficient. On montre, qu'après une permutation des colonnes de **M** effectuée par une matrice de permutation $\mathbf{\Pi}$, il est toujours possible d'écrire $\mathbf{M\Pi} = \mathbf{QR}$. Si r désigne le rang de la matrice **M**, alors les r premières colonnes de **Q** forment une base orthonormée de l'image de **M** et les $n - r$ suivantes une base orthonormée de l'espace qui lui est supplémentaire (c'est celui qui nous inTéresse). Toute la difficulté réside dans la détermination de r. Pour ce faire, remarquons que les dernières colonnes de **Q** doivent être des vecteurs propres de **M** pour la valeur propre zéro, par conséquent on pourra s'inspirer des remarques du chapitre précédent pour trouver où se situe la ligne de démarcation dans les colonnes de la matrice **Q** entre la base de l'espace image de **A** et la base de son supplémentaire.

4.5 La méthode progressive

4.5.1 Détermination du premier cône

Comme il est dit plus haut, pour trouver un premier cône il « suffit » d'extraire de **K** n_0 lignes linéairement indépendantes. Mais si l'on extrait n_0 lignes au hasard, il est possible qu'elles ne soient pas linéairement indépendantes. De nouveau la décomposition QR va nous aider, cette fois-ci c'est la matrice de permutation $\mathbf{\Pi}$ qui va indiquer où sont les colonnes de la matrice **M** qui ont servi à trouver la base de son image. Prenons à titre d'exemple la matrice **M** de rang 2 suivante :

$$\mathbf{M} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 5 & 6 \\ 1 & 8 & 9 \\ 1 & 11 & 12 \end{bmatrix} .$$

La décomposition QR peut donner :

$$\mathbf{\Pi} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

avec

$$\mathbf{QR} = \begin{bmatrix} -.182 & -.816 & .514 & .191 \\ -.365 & .408 & -.827 & .129 \\ .548 & .000 & .113 & -.829 \\ -.730 & .408 & .200 & .510 \end{bmatrix} \begin{bmatrix} -16.4 & -14.600 & -1.820 \\ 0.0 & .816 & -.816 \\ 0.0 & 0.0 & .000 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} .$$

La matrice **Π** indique que les deux dernières colonnes de **M** sont linéairement indépendantes.

Les vecteurs extrêmes du cône de départ seront définis à l'aide d'un mot de 32-bits (un entier par exemple) où le bit numéro i sera mis à 1 si on a extrait la ligne $i+1$ de **K** pour définir ce vecteur extrême, il sera mis à 0 dans le cas contraire. Le décalage d'une unité vient de la convention qui veut que le premier bit d'un mot porte le numéro zéro. Pour que cette procédure ait un sens il faut naturellement que le nombre d'étoiles de la base stellaire soit inférieur ou égal à 32. Tous les vecteurs extrêmes que nous seront amenés à trouver au cours de l'étape suivante seront caractérisés de façon identique.

4.5.2 Tri entre les classes k^+ et k^-

Les vecteurs extrêmes du cône K_n sont donc stockés dans un tableau d'entiers de 32 bits (ou de 64 bits si nécessaire et si le compilateur le permet), la taille de ce tableau doit être prévue assez grande car il n'est pas rare que le nombre de vecteurs extrêmes dépasse 10 000.

Afin d'illustrer notre propos, supposons que la base stellaire comporte 10 étoiles ($n_\star = 10$) et que l'on cherche à synthétiser 5 largeurs équivalentes ($n_\lambda = 5$). Supposons en outre que le noyau possède la dimension $n_0 = 10 - 5 = 5$ (base non-dégénérée). Un entier définissant un rayon extrême comportera $n_0 - 1 = 4$ bits à 1, les autres étant à zéro, par exemple :

numéro du bit :	0 1 2 3 4 5 6 7 8 9 ... 31
vecteur n° 1	0 1 0 1 1 0 1 0 0 0 ⋯ 0
vecteur n° 2	1 0 0 1 1 0 1 0 0 0 ⋯ 0
vecteur n° 3	0 0 1 1 1 0 0 0 0 1 ⋯ 0
etc...	

Comme l'indiquent les entiers ci-dessus, le premier vecteur est défini en extrayant les lignes 2,4,5 et 7 de la matrice **K** ; le deuxième les lignes 1,4,5 et 7 et le troisième les lignes 3,4,5 et 10. Les vecteurs n° 1 et n° 2 sont sur la même facette du cône (ou de façon équivalente sur la même arête du polytope) car ils ont $n_0 - 2 = 3$ égalités en commun, à savoir : les égalités 4,5 et 7 ou, si l'on préfère, les bits qui les définissent sont les mêmes à deux positions près : les positions 0 et 1.

Pour savoir si les entiers `vecteur(i)` et `vecteur(j)` sont « sur la même arête » il suffit de combiner ces deux valeurs à l'aide d'un « ou exclusif », c'est-à-dire en Fortran 95 : `kij=Ieor(vecteur(i),vecteur(j))`. Si le résultat `kij` ne possède que deux bits à 1 alors `vecteur(i)` et `vecteur(j)` sont sur la même arête, dans le cas contraire ils ne le sont pas. La majeure partie du temps est consommée par ce test, il est donc indispensable de compter les bits de `kij` de la façon la plus rapide possible.

Si le nombre d'étoiles n'est pas trop grand, on peut repérer à l'avance quels sont les nombres qui ne sont formés que de deux bits et s'y référer en temps utile. On peut, par exemple, déclarer un tableau de type logique et de dimension $2^{n_\star}$ initialisé à `.False.`, soit :

```
Logical, Dimension(2**ns) : : TwoBits=.False.
```

Il faut ensuite générer tous les nombres de deux bits parmi $n_\star$ (il y en a $C^2_{n_\star} = n_\star(n_\star-1)/2$) et de mettre à `.True.` l'élément correspondant du tableau `TwoBits`. Une simple référence à `Twobits` permet alors de savoir si `kij` est sur une arête du polytope, ce qui est le cas si l'ordre suivant est vrai :

```
TwoBits(Ieor(vecteur(i),vecteur(j)))
```

Il ne faut cependant pas oublier que $2^{n_\star}$ peut être exagérément trop grand : $2^{16} = 65\,536$ ce qui est raisonnable, mais $2^{32} = 4\,294\,967\,296$ ce qui ne l'est pas. Si $n_\star$ est trop grand, il faudra trouver une procédure rapide pour compter les bits, cet exercice est laissé aux étudiants.

4.6 Le module d'algèbre linéaire

Afin de résoudre notre problème nous avons besoin de pratiquer les opérations suivantes, relevant toutes de l'algèbre linéaire :

1. Diagonaliser une matrice symétrique définie non-négative.
2. Procéder à la décomposition QR adaptée à une matrice de rang déficient.
3. Résoudre un système linéaire de n équations à n inconnues par la méthode de Gauss.

La bibliothèque `lapack` disponible en copyleft sur « la toile » à l'adresse :

```
http ://www.netlib.org/lapack95/
```

est particulièrement recommandée pour mener à bien ce genre de tâches, l'excellent ouvrage de Golub et Van Loan [1], par exemple, y fait constamment référence.

On trouvera en appendice, page 19, toute la documentation de la bibliothèque `lapack`, c'est beaucoup plus qu'il ne vous en faut mais ce document pourra vous servir plus tard. Sur les stations Alpha du SIO, on obtient la documentation détaillée de `lapack` par la commande `man` suivie du nom de la routine qui vous intéresse. Par exemple : `man dgesv` doit produire en sortie un fichier similaire à celui représenté sur la figure 9 page suivante.

Pour pouvoir utiliser les routines `lapack` sur les stations Alpha sous `unix`, il faut inclure l'option `-ldxml`, par exemple : `f95 prog.f90 -ldxml`.

Références

[1] Golub G., Van Loan Ch., 1996, *Matrix Computations*, Third edition, The John Hopkins University Press, Baltimore and London

[2] Press W., Teukolsky S., Vetterling W., Flannery B., 1992, *Numerical Recipes, Second edition*, Cambridge University Press, Cambridge

[3] Wilkinson J. H., 1963, *Rounding Errors in Algebraic Processes*, Her Majesty's Stationery Office, London

```
DGESV(l)      LAPACK driver routine (version 2.0)      DGESV(l)

NAME
  DGESV - compute the solution to a real system of linear equations  A * X = B,

SYNOPSIS

  SUBROUTINE DGESV( N, NRHS, A,LDA, IPIV, B, LDB, INFO )

      INTEGER     INFO, LDA, LDB, N, NRHS

      INTEGER     IPIV( * )

      DOUBLE     PRECISION A( LDA, * ), B( LDB, * )

PURPOSE
  DGESV computes the solution to a real system of linear equations
     A * X = B,where A is an N-by-N matrix and X and B are N-by-NRHS
  matrices.

  The LU decomposition with partial pivoting and row interchanges is used to
  factor A as
     A = P * L * U,
  where P is a permutation matrix, L is unit lower triangular, and U is upper
  triangular.  The factored form of A is then used to solve the system of
  equations A * X = B.

ARGUMENTS

  N   (input) INTEGER
  The number of linear equations, i.e.,the order of the matrix A.
          N >= 0.

  NRHS   (input) INTEGER
  The number of right hand sides, i.e.,the number of columns of the
  matrix B.  NRHS >= 0.

  A   (input/output) DOUBLE PRECISION array, dimension (LDA,N)
  On entry, the N-by-N coefficient matrix A.  On exit, the factors L
  and U from the factorization A = P*L*U; the unit diagonal elements
  of L are not stored.

  LDA   (input) INTEGER
  The leading dimension of the array A.  LDA >= max(1,N).

  IPIV   (output) INTEGER array, dimension (N)
  The pivot indices that define the permutation matrix P; row i of
  the matrix was interchanged with row IPIV(i).

  B   (input/output) DOUBLE PRECISION array, dimension (LDB,NRHS)
  On entry, the N-by-NRHS matrix of right hand side matrix B.  On
  exit,if INFO = 0, the N-by-NRHS solution matrix X.

  LDB   (input) INTEGER
  The leading dimension of the array B.  LDB >= max(1,N).

  INFO   (output) INTEGER
  = 0: successful exit
  < 0: if INFO = -i, the i-th argument had an illegal value
  > 0: if INFO = i, U(i,i) is exactly zero.  The factorization has
  been completed, but the factor U is exactly singular,so the solu-
  tion could not be computed.
```

FIG. 9 – Documentation en ligne, suite à la commande `man dgesv` sur les stations alphas du SIO.

A `lapack` - A library of linear algebra routines

A.1 Description

LAPACK (Linear Algebra Package) is a new library of dense linear and eigenproblem solvers that supercedes LINPACK and EISPACK, offering better performance and accuracy.

DXML V3.1 includes a compiled and optimized version of LAPACK 2.0, which was released in September 1994.

LAPACK includes subroutines for solving the most common problems in numerical linear algebra :

- Solving systems of simultaneous linear equations
- Finding least squares solutions of overdetermined systems of equations
- Solving eigenvalue problems
- Solving singular value problems

The extensive functionality provided by LAPACK includes routines for the following matrix factorizations :

LU, Cholesky, QR, SVD, Schur and Generalized Schur.

Where appropriate, these functions are provided for the following matrices :

- General ; General band ; General tridiagonal
- Symmetric ; Symmetric band ; Symmetric tridiagonal ; Symmetric, packed storage
- Symmetric positive definite ; Symmetric positive definite band ; Symmetric positive definite, tridiagonal
- Triangular ; Triangular band ; Triangular, packed storage

LAPACK extends the functionality of LINPACK and EISPACK by including equilibration, iterative refinement, error bounds, and driver routines for linear systems, routines for computing and re-ordering the Schur factorization, and condition estimation routines for eigenvalue problems. LAPACK improves on the accuracy of the standard algorithms in EISPACK by including high accuracy algorithms for finding singular values and eigenvalues of bidiagonal and tridiagonal matrices respectively that arise in SVD and symmetric eigenvalue problems.

The performance of the public-domain LAPACK routines on Alpha AXP platforms is improved through the use of the optimized BLAS subprograms.

A.2 Naming scheme

The name of each LAPACK routine is a coded specification of its function (within the very tight limits of standard Fortran 77 6-character names). All driver and computational routines have names of the form `xyyzzz`, where for some driver routines the 6th character is blank.

The first letter, `x`, indicates the data type as follows :

- `s Real`
- `d Double Precision`
- `c Complex`
- `z Complex*16 or Double Complex`

The next two letters, `yy`, indicate the type of matrix (or of the most significant matrix). Most of these two-letter codes apply to both real and complex matrices ; a few apply specifically to one or the other.

The last three letters `zzz` indicate the computation performed. For example, `sgebrd` is a single precision routine that performs a bidiagonal reduction (`brd`) of a real general matrix.

A.3 List of routines

LAPACK includes both computational routines that perform a distinct algorithmic task (such as `dgetrf` performing an LU factorization) as well as driver routines that solve a complete problem (such as `dgesv` solving a system of linear equations). The routines are listed following the chapters of the book "Matrix Computations" of G. H. Golub and Ch. F. Van Loan [1].

The Subprogram Name is the name of the manual page containing documentation on the subprogram. On `unix` systems, the manual page is available through the `man` command.

A.3.1 General Linear Systems

Chapter 3 of Golub and Van Loan.

General Linear Systems	
`_gesv`	Solve $AX = B$
`_gecon`	Condition estimate via $PA = LU$
`_gerfs`	Improve $AX = B$, $A^T X = B$, $A^H X = B$ solutions with error bounds
`_gesvx`	Solve $AX = B$, $A^T X = B$, $A^H X = B$ with condition estimate
`_getrf`	$PA = LU$
`_getrs`	Solve $AX = B$, $A^T X = B$, $A^H X = B$ via $PA = LU$
`_getri`	A^{-1}
`_geequ`	Equilibration

Triangular Systems	
`_trsv`	Solves $Ax = b$
`_trsm`	Solves $AX = B$
`_trcon`	Condition estimate
`_trrfs`	Solve $AX = B$, $A^T X = B$ with error bounds
`_trtrs`	Solve $AX = B$, $A^T X = B$
`_trtri`	A^{-1}

A.3.2 Special Linear Systems

Chapter 4 of Golub and Van Loan.

General Band Matrices	
_gbsv	Solves $AX = B$
_gbcon	Condition estimator
_gbrfs	Improve $AX = B$, $A^T X = B$, $A^H X = B$ solution with error bounds
_gbsvx	Solve $AX = B$, $A^T X = B$, $A^H X = B$ with condition estimate
_gbtrf	$PA = LU$
_gbtrs	Solve $AX = B$, $A^T X = B$, $A^H X = B$ via $PA = LU$
_gbequ	Equilibration

General Tridiagonal Matrices	
_gtsv	Solves $AX = B$
_gtcon	Condition estimator
_gtrfs	Improve $AX = B$, $A^T X = B$, $A^H X = B$ solution with error bounds
_gtsvx	Solve $AX = B$, $A^T X = B$, $A^H X = B$ with condition estimate
_gttrf	$PA = LU$
_gttrs	Solve $AX = B$, $A^T X = B$, $A^H X = B$ via $PA = LU$

Full Symmetric Positive Definite	
_posv	Solves $AX = B$
_pocon	Condition estimate via $PA = LU$
_porfs	Improve $AX = B$ solution with error bounds
_posvx	Solve $AX = B$ with condition estimate
_potrf	$A = GG^T$
_potrs	Solve $AX = B$ via $A = GG^T$
_potri	A^{-1}
_poequ	Equilibration
_pp___	Same, packed storage
_pbstf	$A = GG^T$ (A banded storage)

Banded Symmetric Positive Definite	
_pbsv	Solves $AX = B$
_pbcon	Condition estimate via $A = GG^T$
_pbrfs	Improve $AX = B$ solution with error bounds
_pbsvx	Solve $AX = B$ with condition estimate
_pbtrf	$A = GG^T$
_pbtrs	Solve $AX = B$ via $A = GG^T$
_pbequ	Equilibration

Tridiagonal Symmetric Positive Definite	
_ptsv	Solves $AX = B$
_ptcon	Condition estimate via $A = LDL^T$
_ptrfs	Improve $AX = B$ solution with error bounds
_ptsvx	Solve $AX = B$ with condition estimate
_pttrf	$A = LDL^T$
_pttrs	Solve $AX = B$ via $A = LDL^T$

Full Symmetric Indefinite	
_sysv	Solves $AX = B$
_sycon	Condition estimate via $PAP^T = LDL^T$
_syrfs	Improve $AX = B$ solution with error bounds
_sysvx	Solve $AX = B$ with condition estimate
_sytrf	$PAP^T = LDL^T$
_sytrs	Solve $AX = B$ via $PAP^T = LDL^T$
_sytri	A^{-1}
_sp___	Same (A packed storage)
_he___	Same (A complex Hermitian)
_hp___	Same (A complex Hermitian, packed)

Triangular Banded Matrices	
_tbcon	Condition estimate
_tbrfs	Improve $AX = B$, $A^T X = B$ solution with error bounds
_tbtrs	Solve $AX = B$, $A^T X = B$
_tp___	Same, A packed storage
_tptri	A^{-1}, packed storage

A.3.3 Orthogonalization and Least Square

Chapter 5 of Golub and Van Loan.

Householder/Givens Tools	
_larfg	Generates a Householder matrix
_larf	Householder times matrix
_larfx	Small n Householder times matrix
_larfb	Block Householder times matrix
_larft	Computes $I - VTV^H$ block reflector representation
_lartg	Generates a plane rotation
_largv	Generates a vector of plane rotations
_lartv	Applies a vector of plane rotations to a vector pair
_lasr	Applies rotation sequence to matrix
_rot	Performs Givens plane rotation
csrot	Real rotation times complex vector pair
zdrot	Same, double precision

Orthogonal Factorizations	
_geqrf	$A = QR$ (orthogonal)(upper triangular)
_geqpf	$A\Pi = QR$
_ormqr	Q (factored form) times matrix (real case)
_unmqr	Q (factored form) times matrix (complex case)
_orgqr	Generates Q from _geqrf output (real case)
_ungqr	Generates Q from _geqrf output (complex case)
_gerqf	$A = RQ =$ (upper triangular)(orthogonal)
_gelqf	$A = LQ =$ (lower triangular)(orthogonal)
_geqlf	$A = QL =$ (orthogonal)(lower triangular)
_tzrqf	$A = RQ$ where A is upper trapezoidal
_orgrq	Generates Q from _gerqf output
_ormrq	Multiplies by Q from _gerqf output
____lq	Same but for _gelqf output
____ql	Same but for _geqlf output
_gesvd	$A = U\Sigma V^T$
_bdsqr	SVD of real bidiagonal matrix
_gebrd	Bidiagonalization of general matrix
_orgbr	Generates the orthogonal transformations
_gbbrd	Bidiagonalization of band matrix

Least squares	
_gels	Full rank $\min \|\|AX - B\|\|_{2F}$ or $\min \|\|A^H X - B\|\|_F$
_gelss	SVD solution to $\min \|\|AX - B\|\|_F$
_gelsx	Complete orthogonal decomposition solution to $\min \|\|AX - B\|\|_F$
_geequ	Equilibrates general matrix to reduce condition
_ggglm	Solves the Generalized Linear Regression Model

A.3.4 The Unsymmetric Eigenvalue Problem

Chapter 7 of Golub and Van Loan.

Unsymmetric Eigenproblem	
_gebal	Balance transform
_gebak	Undo balance transform
_gehrd	Hessenberg reduction $U^H AV = H$
_ormhr	U (factored form) times matrix (real case)
_orghr	Generates U (real case)
_unmhr	U (factored form) times matrix (complex case)
_unghr	Generates U (complex case)
_hseqr	Schur decomposition of Hessenberg matrix
_hsein	Eigenvectors of Hessenberg matrix by inverse iteration
_gees	Schur decomp of general matrix with e.values ordering
_geesx	Same but with condition estimate
_geev	Eigenvalues and left and right eigenvectors of general matrix
_geevx	Same but with condition estimate

Unsymmetric Eigenproblem (continued)	
_trevc	Selected eigenvectors of upper quasitriangular matrix
_trsna	Cond. estimates of selected eigenvalues of upper quasitriangular matrix
_trexc	Unitary reordering of Schur decomposition
_trsen	Same but with condition estimate
_trsyl	Solve $AX + XB = C$ for upper quasitriangular A and B

Unsymmetric Generalized Eigenproblem	
_ggbal	Balance transform
_ggbak	Undo balance transform
_gghrd	Reduction to Hessenberg-Triangular form
_hgeqz	Generalized Schur decomposition
_tgevc	Some or all eigenvectors
_gegv	All generalized eigenvalues and eigenvectors
_gegs	All generalized eigenvalues, Schur forms and Schur vectors

A.3.5 The Symmetric Eigenvalue Problem

Chapter 8 of Golub and Van Loan.

Symmetric Eigenproblem	
_syev	All eigenvalues and eigenvectors
_syevd	Same but uses divide and conquer for eigenvectors
_syevx	Selected eigenvalues and vectors
_sp___	Same, packed storage
_stev	All eigenvalues and vectors of tridiagonal (A real)
_steqr	All eigenvalues and vectors of tridiagonal by implicit QR
_stedc	All eigenvalues and vectors of tridiagonal by divide and conquer
_stevd	Same (A real)
_sterf	All eigenvalues of tridiagonal by root-free QR
_pteqr	All eigenvalues and eigenvectors of positive definite tridiagonal
_stebz	Selected eigenvalues of tridiagonal by bisection
_stein	Selected eigenvectors of tridiagonal by inverse iteration
_sbev	All eigenvalues and eigenvectors of band matrix
_sbevd	Same but uses divide and conquer for eigenvectors
_sbevx	Selected eigenvalues and vectors (real case)
_hbevx	Selected eigenvalues and vectors (complex case)

Symmetric-Definite Eigenproblem	
_sygst	Converts $A - \lambda B$ to $C - \lambda I$ form
_pbstf	Split Cholesky factorization
_sbgst	Converts banded $A - \lambda B$ to $C - \lambda I$ form via split Cholesky
_sbgv	All eigenvalues and eigenvectors (A banded)

Householder Tridiagonalisation	
_sytrd	Householder tridiagonalisation (A real)
_hetrd	Householder tridiagonalisation (A complex Hermitian)
_sbtrd	Householder tridiagonalisation (A real banded)
_orgtr	Generates Q from _sytrd/_hetrd output
_opmtr	Multiplies by Q as coded from _sytrd/_hetrd
_sptrd	Householder tridiagonalisation (A real, packed)
_hptrd	Householder tridiagonalisation (A complex Hermitian, packed)
_opgtr	Generates Q from _sptrd/_hptrd output
_opmtr	Multiplies by Q as coded from _sptrd/_hptrd

Symmetric Generalized Eigenproblem	
_sygv	All eigenvalues and eigenvectors (A, B symmetric, $B > 0$)
_spgv	Same (A, B packed)

SVD	
_gesvd	$A = U\Sigma V^T$ (A general rectangular)
_bdsqr	SVD of real bidiagonal matrix
_gebrd	bidiagonalization of general matrix
_orgbr	generates the orthogonal transformations
_ormbr	Multiplies by one of _gebrd output matrices
_gbbrd	bidiagonalization of band matrix

The generalized Singular Value Problem	
_ggsvd	Computes GSVD of a pair of matrices
_ggsvp	Converts $A^T A - \mu^2 B^T B$ to triangular $A_1^T A_1 - \mu^2 B_1^T B_1$
_tgsja	Computes GSVD of a pair of triangular matrices

A.3.6 Special Topics

Chapter 12 of Golub and Van Loan.

Tools for Generalized/Constrained LS Problems	
`_gglse`	Solve the equality constrained LS problem
`_ggqrf`	Computes the generalized QR factorization of a matrix pair
`_ggrqf`	Computes the generalized RQ factorization of a matrix pair
`_ggsvp`	Converts the GSVD problem to triangular form
`_tgsja`	Computes the GSVD problem of a pair of triangular matrices