FORME NORMALE DE JORDAN D'UNE MATRICE

Raymond SÉROUL
UFR de Mathématiques (Strasbourg)

Un coup d'œil à la littérature laisse la désagréable impression que la théorie de la forme normale de Jordan d'une matrice est un sujet difficile et qu'il n'est pas aisé de trouver explicitement la matrice J ; quant à la matrice de passage, c'est à peine si on ose en rêver ! On sent que les auteurs ont hâte de se débarrasser du sujet...

Vers les années 1980, j'avais découvert avec délices un article de Pittelkow et Runckel (voir [2] ou [4]) qui décrivait un algorithme pour fabriquer J et P (et démontrait du même coup leur existence). Malheureusement, cet algorithme nécessite des notations délicates ; quand on le pratique, les erreurs matérielles sont nombreuses et il faut résoudre beaucoup de systèmes linéaires ; bref, je me verrai mal enseigner cela en DEUG.

Un pas sans doute définitif vient d'être franchi : trois mathématiciens espagnols, Bujosa, Criado et Vega, viennent de publier un algorithme qui *démontre* l'existence de la réduite de Jordan et permet le calcul explicite de J et P en ne faisant appel qu'à des combinaisons linéaires de lignes et de colonnes.

Pour vous donner une idée, « l'algorithme BCV » permet de déduire J et P d'une matrice triangulaire 15×15 en une dizaine de minutes — sans stress ni mal de tête, quand on demande à Maple d'exécuter les combinaisons linéaires.

La simplicité de cet algorithme le met à la portée d'un étudiant de DEUG de première année : quelques rudiments d'algèbre linéaire suffisent. Exit les sous-espaces caractéristiques, les endomorphismes nilpotents et autres !

1. LE DÉCOR

Soit A une matrice $n \times n$ dont nous connaissons les valeurs propres $\lambda_1, \ldots, \lambda_n$; nous travaillons dans un corps k assez grand pour contenir à la fois les coefficients de la matrice et ses valeurs propres. Nous désirons trouver une matrice inversible $P \in \mathrm{Gl}(n, k)$ et une matrice de Jordan $J \in M(n, k)$ telles que

$$J = P^{-1}AP.$$

Pour éviter tout malentendu, notons que l'algorithme BCV *n'est pas un algorithme de calcul numérique* (voir [5] pour une discussion intéressante) puisque nous avons

besoin de connaître *a priori* les valeurs propres de A. Il s'agit seulement d'un algorithme «scolaire» qui établit l'existence d'une forme normale de Jordan et qui nous offre un moyen très commode de fabriquer de manière industrielle tous les exercices de réduction que nous voudrons.

La stratégie suivie par l'algorithme BCV est la suivante :

- on commence par trigonaliser A, c'est-à-dire par fabriquer une matrice P_1 telle que $T = P_1^{-1}AP_1$ est triangulaire supérieure;
- on déduit de T une matrice P_2 telle que $U = P_2^{-1}TP_2$ est diagonale par blocs triangulaires, chaque bloc ne contenant qu'une seule valeur propre (autrement dit, U correspond à une base des sous-espaces caractéristiques de A);
- on jordanise enfin U en fabriquant une matrice P_3 telle que $J = P_3^{-1}UP_3$.

On passe donc de A à J grâce à la formule

$$J = (P_1P_2P_3)^{-1}A\,(P_1P_2P_3).$$

2. OUTILS THÉORIQUES

Rappelons que manipuler les lignes et les colonnes d'une matrice A revient à pré- ou post-multiplier celle-ci par certaines matrices élémentaires :

$P^{-1}A$	AP
$L_i := L_i - \rho L_j$	$K_j := K_j + \rho K_i$
$L_i := \rho^{-1}L_i$	$K_i := \rho K_i$
$L_i \rightleftarrows L_j$	$K_i \rightleftarrows K_j$

Comme nous allons remplacer A par $P^{-1}AP$, nous avons besoin de manipuler simultanément les lignes et les colonnes; nous noterons :

- $[i,j,\rho]$, où $i \neq j$, les manipulations $K_j := K_j + \rho K_i$ et $L_i := L_i - \rho L_j$;
- $[i,i,\rho]$, les manipulations $K_i := \rho K_i$ et $L_i := \rho^{-1}L_i$ (on suppose $\rho \neq 0$);
- $[i,j]$ les manipulations $K_i \rightleftarrows K_j$ et $L_i \rightleftarrows L_j$.

L'ordre des manipulations ligne et colonne importe peu car celles-ci commutent. Si Θ est l'une des manipulations $[i,j,\rho]$ ou $[i,j]$ précédentes, nous avons donc

$$\Theta(A) = P^{-1}AP$$

où P est une matrice qu'il est inutile d'expliciter.

3. OUTILS MAPLE

Manipuler une matrice à la main est ennuyeux et source d'erreurs multiples. C'est pourquoi nous allons demander à Maple de nous seconder, de manière à nous réserver la partie noble : *comprendre et donner des ordres.*

3.1. Procédure de changement de base

Les manipulations $\Theta = [i, j, \rho]$ ou $\Theta = [i, j]$ étant stockées dans une liste L, la procédure Maple suivante se charge du calcul des matrices P et $P^{-1}AP$. L'argument P est optionnel (voir l'exemple).

```
changebase := proc(A :: matrix, L :: list(list), P :: evaln)
local B, Q;
B := copy(A);
Q := manipcol(diag(1$coldim(B)), L);
if nargs > 2 then P := copy(Q) fi ;
evalm(inverse(Q) &* B &* Q);
end :
```

La procédure *manipcol* déduit $P^{-1}AP$ de A en manipulant les lignes et colonnes.

```
manipcol := proc(A :: matrix, L :: list(list))
local ℓ, M;
M := copy(A);
for ℓ in L do
  if nops(ℓ) = 2 then M := swapcol(M, ℓ[1], ℓ[2])
  elif ℓ[1] = ℓ[2] then M := mulcol(M, ℓ[1], ℓ[3])
  else M := addcol(M, ℓ[1], ℓ[2], ℓ[3])
  fi
od ;
evalm(M)
end :
```

Exemple

Quand nous ne nous intéressons qu'à la seule matrice $B = P^{-1}AP$, nous pouvons omettre l'argument P :

```
A := matrix(4, 4, [0, 2, −2, 2, 4, 0, 2, 4, 0, 6, −2, 2, 2, 0, 6, 2]);
L := [[2, 1, 3], [3, 4], [3, 1, 1], [2, 3], [3, 3, 2], [1, 2]] :
B := changebase(A, L);
```

$$A := \begin{bmatrix} 0 & 2 & -2 & 2 \\ 4 & 0 & 2 & 4 \\ 0 & 6 & -2 & 2 \\ 2 & 0 & 6 & 2 \end{bmatrix} \qquad B := \begin{bmatrix} 0 & -4 & -4 & 8 \\ 2 & 8 & 4 & -2 \\ -1 & -8 & -6 & 4 \\ 2 & 20 & 12 & -2 \end{bmatrix}.$$

Quand nous avons besoin de la matrice de passage, nous rajoutons un paramètre supplémentaire à la procédure qui se charge alors de placer dans ce paramètre la

matrice de passage :

$$B := changebase(A, L, P);\ P;$$

Comme précédemment, nous obtenons d'abord la valeur de B puis

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 3 & 2 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}.$$

On peut s'assurer que l'on a bien $B = P^{-1}AP$ en tapant :

$$evalm(\&*\ (inverse(P), A, P) - B);$$

L'affichage d'une matrice nulle confirme que les calculs sont corrects.

3.2. Déplacement d'une colonne

Vers la fin de l'algorithme, nous déplacerons les colonnes (et les lignes) à l'aide de transpositions $\Theta = [i, i+1]$. Comme il est agaçant d'avoir à produire de longues listes de transpositions à la main, voici une petite procédure bien commode :

```
movecol := proc(a :: posint, b :: posint)
local i;
if a < b then seq([i, i + 1], i = a..b − 1)
else seq([b + a − i, b + a − i − 1], i = b..a − 1)
fi
end :
```

Exemple

Les instructions $S := movecol(5, 2)$ et $T := movecol(7, 12)$ fabriquent respectivement les suites

$$S := [5, 4], [4, 3], [3, 2]$$
$$T := [7, 8], [8, 9], [9, 10], [10, 11], [11, 12]$$

3.3. Une procédure pour un meilleur environnement de travail

Quand on travaille dans une matrice de dimension élevée, on se rend vite compte que l'on passe beaucoup de temps à compter les numéros de ligne et de colonnes. La procédure suivante, qui borde une matrice avec deux «règles graduées», évite beaucoup de fausses manœuvres.

```
graduation := proc(A :: matrix)
local regleh, reglev;
regleh := vector([seq(i, i = 0..coldim(A))]);
reglev := vector([seq(i, i = 1..coldim(A))]);
stack(regleh, concat(reglev, A));
end :
```

Exemple

La commande *graduation*(A) affiche la matrice (les règles sont grisées) :

$$\begin{bmatrix} \mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} \\ \mathbf{1} & 0 & 2 & -2 & 2 \\ \mathbf{2} & 4 & 0 & 2 & 4 \\ \mathbf{3} & 0 & 6 & -2 & 2 \\ \mathbf{4} & 2 & 0 & 6 & 2 \end{bmatrix}.$$

4. PREMIÈRE ÉTAPE : TRIGONALISATION

Nous allons progressivement *trigonaliser* la matrice A en construisant une suite de matrices inversibles $P_1, \ldots, P_n$ telles que

$$P_1^{-1}AP_1 = \begin{bmatrix} \lambda_1 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{bmatrix}, \quad P_2^{-1}AP_2 = \begin{bmatrix} \lambda_1 & * & * & * \\ 0 & \lambda_2 & * & * \\ 0 & 0 & * & * \\ 0 & 0 & * & * \end{bmatrix},$$

$$P_3^{-1}AP_3 = \begin{bmatrix} \lambda_1 & * & * & * \\ 0 & \lambda_2 & * & * \\ 0 & 0 & \lambda_3 & * \\ 0 & 0 & 0 & * \end{bmatrix}, \quad \textit{etc.}$$

Les colonnes de la matrice finale T apparaissent dans l'ordre $1, 2, \ldots, n$.

4.1. Description de l'algorithme

Posons

$$M_1 = A - \lambda_1 I_n \quad (\lambda_1 \text{ valeur propre}).$$

La matrice M_1 n'étant pas inversible (son déterminant est nul), nous savons qu'il existe une combinaison linéaire nulle des colonnes de M. Pour faire apparaître *automatiquement* cette combinaison linéaire sans résoudre (de manière apparente) un système linéaire, nous parcourons la dernière colonne de M_1 de *haut en bas* (il est essentiel de ne pas se tromper de sens !) :

• Si la dernière colonne de M_1 est nulle, nous avons terminé ; nous amenons la colonne à la place de la première colonne grâce à la manipulation $\Theta = [1, n]$ (il est inutile d'utiliser *movecol* pour cela !).

• Si la dernière colonne de M_1, n'est pas nulle, appelons $m_{i_1,n}$ le premier coefficient non nul rencontré quand on part du haut de la colonne ; appelons ce coefficient le *pivot*. Nous «nettoyons» la ligne

$$L_{i_1} = (m_{i_1,1},\ m_{i_1,2},\ \ldots\ , m_{i_1,n})$$

en tuant tous les coefficients qui se trouvent à gauche du pivot grâce à des manipulations de la forme

$$\Theta = [n, j, \rho], \quad \rho = -\frac{m_{i_1,j}}{m_{i_1,n}}, \quad 1 \leq j < n.$$

Remarques

1) Quand on travaille manuellement, on peut remplacer mentalement $\Theta = [n, j, \rho]$ par la seule manipulation colonne

$$\Theta' = \{K_j := K_j + \rho K_n\}.$$

En effet, la restriction de $\Theta = [n, j, \rho]$ à la ligne L_{i_1} est égale à Θ' ; or nous ne nous intéressons qu'à la seule ligne L_{i_1}...

2) Le sens du nettoyage dans la ligne L_{i_1} n'a aucune d'importance.

Une fois la ligne du pivot nettoyée, nous passons à la colonne $n-1$ en travaillant dans la sous-matrice formée par les colonnes $2, \ldots, n$. Nous procédons de même avec les colonnes restantes.

Les colonnes non nulles de la matrice obtenues étant — par construction — linéairement indépendantes, on voit tôt ou tard apparaître une colonne nulle qu'on amène en position 1 grâce à la manipulation $\Theta = [1, j]$.

Exemple

Considérons la matrice

$$A = \begin{bmatrix} -1 & 1 & -1 & -1 & 1 & 1 \\ -2 & 2 & -1 & 5 & 1 & 0 \\ -2 & 1 & 0 & 3 & 1 & -1 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ -4 & 1 & -1 & 0 & 3 & 3 \\ 0 & 0 & 0 & 1 & 0 & 2 \end{bmatrix}.$$

dont les valeurs propres sont $\Lambda = [1, 1, 1, 1, 2, 2]$. L'algorithme démarre

$$M_1 = A - \mathrm{Id} = \begin{bmatrix} -2 & 1 & -1 & -1 & 1 & 1 \\ -2 & 1 & -1 & 5 & 1 & 0 \\ -2 & 1 & -1 & 3 & 1 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ -4 & 1 & -1 & 0 & 2 & 3 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

• Le pivot dans la colonne 6 est $m_{1,6} = 1$; nous nettoyons cette ligne à l'aide de la liste $L_1 = [[6, 1, 2], [6, 2, -1], [6, 3, 1], [6, 4, 1], [6, 5, -1]]$ et nous obtenons

$$M_1' = \mathit{changebase}(M_1, L_1) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ -2 & 1 & -1 & 5 & 1 & 0 \\ -4 & 2 & -2 & 2 & 2 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 2 & -2 & 2 & 3 & -1 & 3 \\ 6 & -4 & 4 & 7 & -3 & 3 \end{bmatrix}.$$

• Le pivot dans la colonne 5 est $m_{2,5} = 1$; nous nettoyons cette ligne à l'aide de la liste $L_2 = [[5, 1, 2], [5, 2, -1], [5, 3, 1], [5, 4, -5]]$ et nous obtenons

$$M_1'' = \mathit{changebase}(M_1', L_2) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -8 & 2 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 21 & -2 & 2 \\ 0 & -1 & 1 & 22 & -3 & 3 \end{bmatrix}.$$

• Le pivot dans la colonne 4 est $m_{3,4} = -8$; comme il n'y à rien à nettoyer dans la ligne 3, nous recherchons le pivot $m_{5,3} = 1$ dans la colonne 3 et nous nettoyons la ligne 5 à l'aide de la liste $L_3 = [[3, 2, 1]]$, ce qui nous donne

$$M_1''' = changebase(M_1'', L_3) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -8 & 1 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 21 & -2 & 2 \\ 0 & 0 & 1 & 22 & -3 & 3 \end{bmatrix}.$$

Nous constatons alors que les colonnes 1 et 2 sont nulles. Comme $\lambda = 1$ est une valeur propre d'ordre 4, nous devons faire encore apparaître deux colonnes nulles dans la sous-matrice obtenue en supprimant les lignes et les colonnes 1,2.

• Le pivot dans la colonne 6 est $m_{3,6} = -1$; nous nettoyons la ligne 3 à l'aide de la liste $L_4 = [[6, 4, -8], [6, 5, 1]]$ et nous voyons apparaître la matrice

$$M_1^{(iv)} = changebase(M_1''', L_4) = \begin{bmatrix} 0 & 0 & 0 & -8 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 5 & 0 & 2 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

• Nous échangeons les colonnes 3 et 5, ce qui fait apparaître une nouvelle colonne nulle; la liste $L_5 = [[3, 5], [4, 5]]$ fabrique

$$M_1^{(v)} = changebase(M_1^{(iv)}, L_5) = \begin{bmatrix} 0 & 0 & 1 & 0 & -8 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 5 & 2 \\ 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

Nous changeons maintenant de valeur propre en considérant la matrice

$$M_2 = A - 2\mathrm{Id}$$

à laquelle nous appliquons d'abord les transformations précédentes, ce qui nous donne

$$M_2' = changebase(M_1^{(iv)}, [op(L_1), \ldots, op(L_5)) = \begin{bmatrix} -1 & 0 & 1 & 0 & -8 & 1 \\ 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 5 & 2 \\ 0 & 0 & 0 & -1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

On constate alors qu'il suffit d'échanger les colonnes 5 et 6 dans la sous-matrice formée par les éléments d'indice ≥ 5.

En résumé, la liste de manipulations

$$\begin{aligned} L = [&[6,1,2],[6,2,-1],[6,3,1],[6,4,1],[6,5,-1] \\ &[5,1,2],[5,2,-1],[5,3,1],[5,4,-5] \\ &[3,2,1] \\ &[6,4,-8],[6,5,1] \\ &[3,5],[4,5] \\ &[5,6]] \end{aligned}$$

trigonalise la matrice A en la matrice

$$T = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & -8 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 2 & 5 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix},$$

la matrice de passage étant

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 2 & 0 & 1 & 1 & 0 & -5 \\ 0 & 0 & 0 & 0 & 1 & -2 \end{bmatrix}.$$

Remarque

Ne vous imaginez pas que les calculs se font toujours dans $\mathbb{Z}$ comme dans cet exemple !

4.2. Procédure Maple de trigonalisation

Nous mémorisons les valeurs propres de la matrice dans la liste $\Lambda = [\lambda_1, \ldots, \lambda_n]$. Le troisième argument P et le quatrième argument L de la procédure *trigonaliser* sont optionnels ; ils permettent d'«espionner» la procédure en recueillant la matrice de passage et les manipulations.

Nous choisissons une valeur propre $\lambda_k \in \Lambda$ (boucle principale[1]) et nous travaillons dans la sous-matrice $M = A - \lambda_k I$ formée par les éléments d'indices $\geq k$ en explorant[2] les colonnes $n, n-1, \ldots, k$. La boucle[3] recherche le pivot dans la sous-colonne numéro j ; si celle-ci est nulle[4], nous appliquons la manipulation $\Theta = [k, j]$ puis nous quittons la boucle principale[5] ; dans le cas contraire[6], nous nettoyons la ligne i entre les colonnes k et $j-1$.

trigonaliser := **proc**(A :: *matrix*, Λ :: *list*, P :: *evaln*, L :: *evaln*)
local $B, M, i, j, k, n, \lambda, jj, coef, s$;
n := *coldim*(A) ; B := *copy*(A) ; s := *NULL* ;

```
for k from 1 to nops(Λ) − 1 do # voir texte(1)
| λ := Λ[k];
| M := evalm(B − λ ∗ diag(1$n));
| for j from n to k by −1 do # voir texte(2)
| | i := k;
| | while i ≤ n and M[i, j] = 0 do i := i + 1 od ; # voir texte(3)
| | if i > n then # voir texte(4)
| | | if k ≠ j then
| | | | s := s, [k, j];
| | | | M := swapcol(M, k, j);
| | | | M := swaprow(M, k, j);
| | | | break ; # voir texte(5)
| | | fi ;
| | | else # voir texte(6)
| | | | for jj from k to j − 1 do
| | | | | coef := −M[i, jj]/M[i, j];
| | | | | if coef = 0 then next fi ;
| | | | | s := s, [j, jj, coef];
| | | | | M := addcol(M, j, jj, coef);
| | | | | M := addrow(M, jj, j, −coef);
| | | | od ;
| | | fi ;
| | od ;
| | B := evalm(M + λ ∗ diag(1$n));
od ;
if nargs > 2 then P := manipcol(diag(1$n), [s]) fi ;
if nargs > 3 then L := [s] fi ;
evalm(B);
end :
```

Exemple

Sachant que $1, 1, 1, 1, 2$ sont les valeurs propres de la matrice

$$A = \begin{bmatrix} 2 & -3 & 1 & 1 & -1 \\ 1 & -2 & 1 & 1 & 2 \\ -1 & 3 & 0 & -1 & 2 \\ 2 & -9 & 2 & 4 & 7 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix},$$

la commande $T := trigonaliser(A, [1, 1, 1, 1, 2], P))$ fabrique

$$T = \begin{bmatrix} 1 & 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 & 2 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix}, \quad P = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ 3 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

5. BLOCS INDÉPENDANTS

Pour l'algorithme de trigonalisation, il n'est pas nécessaire que les valeurs propres égales soient consécutives dans la liste Λ. *Mais, pour aller plus loin, nous supposerons désormais que cette contrainte est respectée.*

Nous pouvons améliorer la trigonalisation précédente en rendant «indépendants» les blocs correspondants à des valeurs propres distinctes; de manière plus précise, si $\lambda_1, \ldots, \lambda_k$ sont les valeurs propres distinctes de A, nous recherchons maintenant une matrice triangulaire de la forme

$$U = P^{-1}TP = \begin{bmatrix} U_1 & 0 & 0 & 0 \\ 0 & U_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & U_k \end{bmatrix}$$

où U_i est une matrice triangulaire ayant la seule valeur propre λ_i de dimension égale à l'ordre de multiplicité de λ_i. (D'un point de vue théorique, il s'agit de trouver une base des sous-espaces caractéristiques.)

5.1. Description de l'algorithme

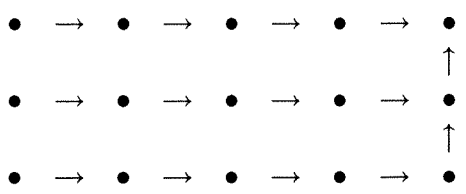
Soient $\lambda \neq \mu$ deux valeurs propres de T et supposons que $t_{i,i} = \lambda$ et $t_{j,j} = \mu$ avec $i < j$:

$$U = \begin{bmatrix} \ddots & & & & \\ & \lambda & \cdots & a_{i,j} & \\ & & \ddots & \vdots & \\ & & & \mu & \\ & & & & \ddots \end{bmatrix}.$$

L'hypothèse $i < j$ montre que l'on peut tuer le coefficient $a_{i,j}$ grâce à la manipulation

$$\Theta = [i, j, \rho], \quad \rho = \frac{a_{i,j}}{\mu - \lambda}.$$

Cette manipulation modifie la colonne K_j et la ligne L_i. Il en résulte que, pour ne pas abîmer les zéros qui remplacent les coefficients que nous détruisons, il est indispensable de parcourir les lignes *de bas en haut* et de se déplacer sur une ligne de *gauche à droite* :



Exemple

Il s'agit de se débarrasser du bloc encadré de la matrice

$$T = \begin{bmatrix} 1 & 1 & 2 & 2 & 1 & -1 & 0 & 0 & -3 \\ 0 & 1 & 1 & 4 & -1 & 0 & 5 & 2 & 2 \\ 0 & 0 & 1 & -1 & 1 & 1 & -2 & 2 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 4 & 2 & -2 \\ 0 & 0 & 0 & 0 & 2 & 0 & 1 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 2 & 1 & -1 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \end{bmatrix}.$$

La ligne la plus basse à nettoyer est la ligne 6; nous tuons $t_{6,7} = 1$, $t_{6,8} = -1$ et $t_{6,9} = 3$ *dans cet ordre* à l'aide des manipulations $[6,7,1], [6,8,-3], [6,9,-1]$. Cela fait, nous remontons pour nettoyer la ligne 5 entre les colonnes 7 à 9, *etc.* La liste complète des manipulations est :

$$\begin{aligned} K := [&[6,7,1], [6,8,-3], [6,9,-1] \\ &[5,7,1], [5,8,-2], [5,9,-1] \\ &[4,5,1], [4,6,1], [4,7,3], [4,8,-\tfrac{9}{2}], [4,9,-\tfrac{23}{4}] \\ &[3,7,-\tfrac{3}{2}], [3,8,\tfrac{9}{4}], [3,9,\tfrac{17}{4}] \\ &[2,5,3], [2,6,4], [2,7,\tfrac{29}{4}], [2,8,-\tfrac{105}{8}], [2,9,-\tfrac{289}{16}] \\ &[1,5,6], [1,6,5], [1,7,\tfrac{41}{8}], [1,8,-\tfrac{215}{16}], [1,9,-\tfrac{341}{16}]] \end{aligned}$$

et la nouvelle matrice $U = \mathit{changebase}(T, K)$ est égale à

$$U = \begin{bmatrix} 1 & 1 & 2 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \end{bmatrix}.$$

La matrice de passage et les manipulations s'obtiennent comme d'habitude.

5.2. La procédure blocs_independants

Là encore, la traduction en code Maple de cet algorithme ne présente pas de difficulté particulière. Le second argument P et le troisième argument L sont optionnels.

blocs_independants := **proc**(A :: *matrix*, P :: *evaln*, L :: *evaln*)
local $B, coef, i, j, k, n, s$;
$n := coldim(A)$; $B := copy(A)$; $s := NULL$;
for k **from** 1 **to** $n-1$ **do**

```
  for i from 1 to n − k do
    j := i + k;
    if B[i,i] = B[j,j] or B[i,j] = 0 then next fi ;
    coef := −B[i,j]/(B[i,i] − B[j,j]) :
    B := addcol(B,i,j,coef); B := addrow(B,j,i,−coef);
    s := s,[i,j,coef];
  od ;
od ;
if nargs > 1 then P := manipcol(diag(1$n),[s]) fi ;
if nargs > 2 then L := [s] fi ;
evalm(B);
end :
```

Ne pas oublier : cette procédure ne fonctionne correctement que si A est déjà triangulaire avec des valeurs propres égales consécutives.

Exemple

Considérons la matrice

$$T = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}.$$

La commande $U := blocs\_independants(T, P)$ fournit la matrice

$$U = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

et la matrice de passage

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 & 8 & -32 \\ 0 & 1 & 0 & 0 & 4 & -12 \\ 0 & 0 & 1 & 0 & 2 & -4 \\ 0 & 0 & 0 & 1 & 1 & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

6. FORME NORMALE DE JORDAN

6.1. Cas d'une seule valeur propre

Soit M une matrice triangulaire supérieure ayant la seule valeur propre λ et dont les $(n-1)$ première colonnes forment déjà une matrice de Jordan ; en revanche, nous ne faisons aucune hypothèse sur la dernière colonne que nous allons manipuler.

Les dernières lignes de blocs de Jordan de M jouent un rôle spécial : si la dernière ligne d'un bloc a pour numéro i, nous dirons que ce bloc est le *bloc de Jordan associé* à l'élément $m_{i,n}$ de la dernière colonne. Dans l'exemple qui suit, il y a trois blocs de Jordan ; les éléments associés (dans la dernière colonne) sont $m_{3,10} = a$, $m_{7,10} = x$ et $m_{9,10} = u$.

$$M = \begin{bmatrix} \boxed{\begin{matrix}\lambda & 1 & 0\\ 0 & \lambda & 1\\ 0 & 0 & \lambda\end{matrix}} & 0 & 0 & 0 & 0 & 0 & 0 & \boxed{\begin{matrix}0\\0\\a\end{matrix}} & \leftarrow m_{3,10} \\ 0 & 0 & 0 & \boxed{\begin{matrix}\lambda & 1 & 0 & 0\\ 0 & \lambda & 1 & 0\\ 0 & 0 & \lambda & 1\\ 0 & 0 & 0 & \lambda\end{matrix}} & 0 & 0 & 0 & \leftarrow m_{7,10} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \boxed{\begin{matrix}\lambda & 1\\ 0 & \lambda\end{matrix}} & \boxed{\begin{matrix}0\\u\end{matrix}} & \leftarrow m_{9,10} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda \end{bmatrix}$$

Pour transformer M en une matrice de Jordan, nous allons modifier la dernière colonne en annulant tous ses coefficients sauf un au plus. Les manipulations qui suivent ne concernent donc que les $m_{i,n}$ pour $i = 1, \ldots, n-1$ (le dernier coefficient, qui est la valeur propre, n'est pas concerné). Le processus comprend quatre étapes.

Première étape. — Nous annulons tous les coefficients de la dernière colonne qui se trouvent «en face» d'un '1' grâce à des manipulations de la forme

$$\Theta = [j, n, \rho], \quad \rho = -m_{i,n}.$$

Quand on travaille à la main, on trouve facilement le coefficient ρ en s'imaginant qu'on utilise la seule manipulation $K_n := K_n + \rho K_j$. Signalons une erreur fréquente à ne pas commettre : si $\lambda = 1$ est une valeur propre, n'essayez pas de vous servir de λ comme pivot... Les '1' qui nous intéressent sont *au-dessus* de la diagonale.

Deuxième étape. — Nous *normalisons* (ce qui signifie que nous remplaçons par des '1') chaque coefficient non nul de la dernière colonne (après la première étape, chacun de ces coefficients est associé à un bloc de Jordan). Si $m = m_{i,n}$ est le coefficient à normaliser, on applique les manipulations

$$\Theta = [\ell, \ell, m], \quad (m = m_{i,n} \text{ coef. à normaliser})$$

à chaque ligne du bloc de Jordan associé au coefficient. Dans l'exemple précédent, la normalisation des coefficients $m_{3,10}$, $m_{7,10}$ et $m_{9,10}$ exige donc trois vagues de manipulations (une vague par bloc) :

$$\begin{aligned} L = \big[& [3,3,a], [2,2,b], [1,1,c], \\ & [7,7,x], [6,6,y], [5,5,z], [4,4,t], \\ & [9,9,u], [8,8,v] \big]. \end{aligned}$$

Dans la pratique, on a $a = b = c$, $x = y = z = t$ et $u = v$; les lettres supplémentaires ne servent qu'à mettre en évidence le phénomène de propagation à l'intérieur des blocs de Jordan.

$$changebase(M, L) = \begin{bmatrix} \lambda & \frac{b}{c} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \lambda & \frac{a}{b} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \lambda & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & \lambda & \frac{z}{t} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \lambda & \frac{y}{z} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \lambda & \frac{x}{y} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \lambda & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda & \frac{u}{v} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda \end{bmatrix}$$

Troisième étape. — Quand la dernière colonne contient au moins deux '1' , ceux-ci «s'affrontent» de manière à ne laisser qu'un seul survivant; celui qui reste est celui qui est associé à un bloc de Jordan de dimension maximum. Pour être plus précis, soient s (pour *source*) et t (pour *but* = *target*) les lignes sur lesquelles se trouvent deux '1'; supposons que le plus gros bloc de Jordan associé à ces coefficients est celui de la ligne s. Le gros bloc «attaque» le plus petit grâce aux manipulations

$$\Theta = [t - i, s - i, 1], \quad i = 0, 1, \ldots$$

de manière que chaque ligne du plus petit bloc soit atteinte. Considérons par exemple la matrice :

$$M = \begin{bmatrix} \lambda & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \lambda & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \lambda & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & \lambda & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \lambda & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \lambda & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \lambda & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda \end{bmatrix} \begin{matrix} \\ \\ \leftarrow m_{3,10} \\ \\ \\ \\ \leftarrow m_{7,10} \\ \\ \leftarrow m_{9,10} \\ \end{matrix}$$

Les sous-blocs de Jordan associés aux éléments non nuls de la dernière colonne sont de dimensions $3, 4, 2$; il en résulte que c'est le coefficient $m_{7,10}$ qui se «débarrasse» des autres grâce aux manipulations

$$L = \big[\, [3, 7, a], [2, 6, b], [1, 5, c], \\ [9, 7, x], [8, 6, y] \,\big]$$

qui transforment M en la matrice $M' = \mathit{changebase}(M, L)$ égale à :

$$M' = \begin{bmatrix} \lambda & 1 & 0 & 0 & 0 & -c+b & 0 & 0 & 0 & 0 \\ 0 & \lambda & 1 & 0 & 0 & 0 & -b+a & 0 & 0 & 0 \\ 0 & 0 & \lambda & 0 & 0 & 0 & 0 & 0 & 0 & -a+1 \\ 0 & 0 & 0 & \lambda & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \lambda & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \lambda & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \lambda & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -y+x & \lambda & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda & -x+1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda \end{bmatrix}$$

Dans la réalité, on a $a = b = c = x = y = 1$; les lettres servent seulement à mettre en évidence un phénomène de propagation à l'intérieur des blocs de Jordan.

Quatrième étape. — Si la dernière colonne est nulle, nous avons une matrice de Jordan. Dans le cas contraire, elle contient un unique '1' et nous déplaçons la dernière colonne vers la gauche à l'aide des manipulations

$$\Theta = [i, i-1], \quad i = n, n-1, \ldots .$$

jusqu'à ce que le '1' se retrouve à côté d'une valeur propre (on se servira de la procédure *movecol*). Dans l'exemple précédent, nous ferions reculer de deux crans la dernière colonne avec la liste

$$L = \big[[10, 9], [9, 8]\big].$$

6.2. Le cas général

Le cas général consiste à partir d'une matrice triangulaire U *à blocs indépendants* et à traiter successivement les colonnes $2, 3, \ldots, n$ selon la méthode qui vient d'être décrite. La présence éventuelle de plusieurs valeurs propres n'est pas un obstacle car les blocs sont indépendants.

$$U = \begin{bmatrix} 2 & 0 & 1 & 2 & 2 & 0 & 0 & 0 \\ 0 & 2 & 1 & 1 & 3 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 1 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \end{bmatrix}$$

Voici la liste complète des manipulations

$L := [[1,2,1]$ # colonne 3 : affrontement
$,[3,4,-1]$ # colonne 4 : annulation (phase 1)
$,[3,3,-1],[2,2,-1]$ # colonne 4 : normalisation
$,[1,3,1]$ # colonne 4 : affrontement
$,movecol(4,2)$ # colonne 4 : phase 4
$,[2,5,-8],[4,5,3]$ # colonne 5 : annulation (phase 1)
$,[2,2,5],[1,1,5],[4,4,-9],[3,3,-9]$ # colonne 5 : normalisation
$,[2,4,1],[1,3,1]$ # colonne 5 : affrontement
$,[7,8,-5]$ # colonne 8 : annulation (phase 1)
$,[7,7,4],[6,6,4]$ # colonne 8 : normalisation
$]$

qui transforment U en la matrice de Jordan

$$J = \begin{bmatrix} 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \end{bmatrix}.$$

7. PROGRAMMATION DE LA RÉDUCTION DE JORDAN

Nous commençons par trigonaliser A (matrice T); nous rendons ensuite les blocs indépendants (matrice U), puis nous jordanisons chaque bloc (matrice J). Une fois de plus, les paramètres P et L sont optionnels.

forme_normale_jordan := **proc**(A :: *matrix*, Λ :: *list*, P :: *evaln*, L :: *evaln*)
local $B, T, U, J, Q_1, Q_2, Q_3, L_1, L_2, L_3$;
B := *copy*(A);
T := *trigonaliser*(A, Λ, Q_1, L_1);
U := *blocs_independants*(T, Q_2, L_2);
J := *jordaniser_blocs*(U, Q_3, L_3);
if *nargs* > 2 **then** P := *evalm*(Q_1 &* Q_2 &* Q_3) **fi** ;
if *nargs* > 3 **then** L := [*op*(L_1), *op*(L_2), *op*(L_3)] **fi** ;
evalm(J)
end :

La procédure *jordaniser_blocs* traite les colonnes $2, 3, \ldots, n$ selon la méthode décrite au paragraphe précédent. Nous n'avons pas à nous préoccuper des différentes valeurs propres car les blocs sont indépendants.

```
jordaniser_blocs := proc(A :: matrix, P :: evaln, L :: evaln)
local B, k, n, Q, L1, L2, s;
B := copy(A); n := coldim(A) : L1 := [];
for k from 2 to n do
| B := traiter_colonne(B, k, Q, L2);
| L1 := [op(L1), op(L2)];
od :
if nargs > 1 then P := manipcol(diag(1$coldim(A)), L1) fi ;
if nargs > 2 then L := L1 ; fi ;
evalm(B)
end :
```

Le traitement de la colonne k comporte les quatre phases décrites dans le paragraphe précédent.

```
traiter_colonne := proc(A :: matrix, k :: posint, P :: evaln, L :: evaln)
local B, B1, B2, B3, B4, Q1, Q2, Q3, Q4, L1, L2, L3, L4;
B := copy(A);
B1 := annuler_coefs_colonne(B, k, Q1, L1);
B2 := normalise_coefs_colonne(B1, k, Q2, L2);
B3 := confrontation_coefs(B2, k, Q3, L3);
B4 := deplace_colonnes(B3, k, Q4, L4);
if nargs > 2 then P := evalm(P1 &* P2 &* P3 &* P4) fi ;
if nargs > 3 then L := [op(L1), op(L2), op(L3), op(L4)] fi ;
evalm(B4)
end :
```

Dans la phase 1, nous annulons les coefficients qui se trouvent sur les lignes contenant un coefficient '1' de la matrice de Jordan formée par les colonnes $1, \ldots, k-1$.

```
annuler_coefs_colonne := proc(A :: matrix, k :: posint, P :: evaln, L :: evaln)
local B, i, coef, s;
B := copy(A); s := NULL;
for i from 1 to k - 2 do
| if B[i, k] <> 0 and B[i, i + 1] <> 0 then
| | coef := -B[i, k]/B[i, i + 1];
| | B := addcol(B, i + 1, k, coef); B := addrow(B, k, i + 1, -coef);
| | s := s, [i + 1, k, coef];
| fi
od ;
if nargs > 2 then P := manipcol(diag(1$coldim(A)), [s]) fi ;
if nargs > 3 then L := [s] fi ;
evalm(B)
end :
```

Pour normaliser un coefficient, il faut connaître le début du bloc de Jordan qui

lui est associé : la fonction *limite_sous_bloc* s'en occupe.

```
normalise_coefs_colonne := proc(A :: matrix, k :: posint, P :: evaln, L :: evaln)
local B, i, coef, debut_bloc, fin_bloc, s;
B := copy(A); s := NULL;
for fin_bloc from 1 to k - 1 do
| coef := B[fin_bloc, k];
| if coef ≠ 0 and coef ≠ 1 then
| | debut_bloc := limite_bloc_jordan(B, fin_bloc);
| | for i from fin_bloc to debut_bloc by - 1 do
| | | s := s, [i, i, coef];
| | | B := mulcol(B, i, coef); B := mulrow(B, i, 1/coef);
| | od
| fi
od ;
if nargs > 2 then P := manipcol(diag(1$coldim(A)), [s]) fi ;
if nargs > 3 then L := [s] fi ;
evalm(B)
end :
```

Pour connaître la première ligne d'un bloc de Jordan, nous remontons le long de la diagonale de la matrice.

```
limite_bloc_jordan := proc(A :: matrix, ℓ :: posint)
local i, debut_bloc;
debut_bloc := ℓ :
for i from ℓ - 1 to 1 by - 1 do
| if A[i, i + 1] ≠ 0 then debut_bloc := i else break fi
od ;
debut_bloc
end :
```

La procédure la plus compliquée s'occupe de la confrontation des coefficients. Nous commençons par rechercher[1] le premier '1' dans la colonne k; quand il y en a un[2], nous cherchons s'il y en a d'autres. Quand il y a bataille[3], on détermine la taille des blocs de Jordan associés aux deux '1' que l'on vient de découvrir. Le '1' associé au plus gros bloc[4],[5] élimine l'autre et prend sa place.

```
confrontation_coefs := proc(A :: matrix, k :: posint, P :: evaln, L :: evaln)
local B, i, ii, debut_bloc1, debut_bloc2, fin_bloc1, fin_bloc2, coef, s;
B := copy(A);
s := NULL;
debut_bloc1 := 0; fin_bloc1 := 0;
debut_bloc2 := 0; fin_bloc2 := 0;
for i from 1 to k - 1 do # voir texte(1)
| if B[i, k] ≠ 0 then fin_bloc1 := i; break fi ;
od ;
```

```
if fin_bloc1 > 0 then # voir texte(2)
| debut_bloc1 := limite_bloc_jordan(B,fin_bloc1);
| for i from fin_bloc1 + 1 to k − 1 do
| | if B[i,k] ≠ 0 then # voir texte(3)
| | | fin_bloc2 := i;
| | | debut_bloc2 := limite_bloc_jordan(B,fin_bloc2);
| | | if fin_bloc2 − debut_bloc2 ≥ fin_bloc1 − debut_bloc1 then # voir texte(4)
| | | | for ii from fin_bloc1 to debut_bloc1 by − 1 do
| | | | | B := addcol(B,ii,fin_bloc2 + ii − fin_bloc1,1);
| | | | | B := addrow(B,fin_bloc2 + ii − fin_bloc1,ii,−1);
| | | | | s := s,[ii,fin_bloc2 + ii − fin_bloc1,1];
| | | | od ;
| | | | debut_bloc1 := debut_bloc2;
| | | | fin_bloc1 := fin_bloc2;
| | | else # voir texte(5)
| | | | for ii from fin_bloc2 to debut_bloc2 by − 1 do
| | | | | B := addcol(B,ii,fin_bloc1 + ii − fin_bloc2,1);
| | | | | B := addrow(B,fin_bloc1 + ii − fin_bloc2,ii,−1);
| | | | | s := s,[ii,fin_bloc1 + ii − fin_bloc2,1];
| | | | od ;
| | | fi ;
| | fi ;
| od ;
fi ;
if nargs > 2 then P := manipcol(diag(1$coldim(A)),[s]) fi ;
if nargs > 3 then L := [s] fi ;
evalm(B)
end :
```

Pour déplacer une colonne k qui contient un seul '1', nous utilisons de manière répétée les manipulations $\Theta = [j-1, j]$ de manière à ce que le '1' se retrouve juste après une valeur propre.

```
deplace_colonnes := proc(A :: matrix,k :: posint,P :: evaln,L :: evaln)
local B,i,j,ℓ,s;
B := copy(A);
s := NULL;
ℓ := 0;
# recherche de l'unique '1'
for i from 1 to k − 1 do if B[i,k] ≠ 0 then ℓ := i; break fi od ;
if ℓ > 0 then # le '1' existant, on le déplace
| for j from k to ℓ + 2 by − 1 do
| | B := swapcol(B,j,j − 1); B := swaprow(B,j,j − 1);
| | s := s,[j,j − 1];
| od ;
```

```
fi ;
if nargs > 2 then P := manipcol(diag(1$coldim(A)), [s]) fi ;
if nargs > 3 then L := [s] fi ;
evalm(B)
end :
```

8. CONCLUSION

On peut énoncer la morale de cette histoire de la manière suivante : *la réduite de Jordan est un algorithme, pas un théorème*! C'est parce qu'elle n'était pas présentée dans le bon contexte que cette théorie était aussi rébarbative.

9. APPENDICE

Quand nous voulons fabriquer des exercices à pratiquer à la main, nous nous tournons spontanément vers des matrices à coefficients entiers et à valeurs propres entières. Pour obtenir de telles matrices, il est naturel de partir d'une matrice triangulaire T à coefficients entiers qu'on brouille par manipulations des lignes et des colonnes, opération qui remplace T par $A = U^{-1}TU$ où U est une matrice unimodulaire(*).

Si les entiers vous enchantent, vous vous demanderez certainement si cette méthode est capable de fabriquer toutes les matrices $A \in M(n, \mathbb{Z})$ à valeurs propres entières.

Théorème (Leavitt et Whaples, 1948). — *Soit A une matrice $n \times n$ à coefficients entiers. Les valeurs propres de A sont entières si et seulement si il existe une matrice unimodulaire U telle que $U^{-1}AU$ soit triangulaire supérieure.*

Démonstration. — Soit A une matrice à coefficients et à valeurs propres entières. Comme les valeurs propres sont rationnelles, nous savons déjà trouver (explicitement) une matrice $P \in \mathrm{Gl}(n, \mathbb{Q})$ telle que $T_1 = P^{-1}AP$ soit triangulaire supérieure. Écrivons cette égalité sous la forme

$$AP = PT_1$$

et multiplions P par un entier convenable de manière à ce qu'elle devienne à coefficients entiers.

En manipulant les lignes de P grâce à l'algorithme de Blankinship (qui est le pivot de Gauss adapté aux entiers, voir [3] ou [4]), nous savons trouver (explicitement)

(*) Rappelons qu'une matrice *unimodulaire* est une matrice à coefficients entiers, inversible et dont l'inverse est aussi à coefficients entiers, cette dernière propriété étant équivalente à $\det \pm 1$ (la preuve est très facile).

une matrice unimodulaire U telle que $T_2 = UP$ soit triangulaire supérieure. Nous déduisons facilement de l'égalité $UAP = UAU^{-1}UP = UPT_1$ que nous avons

$$UAU^{-1} = T_2T_1T_2^{-1}.$$

Alors UAU^{-1} est à coefficients entiers parce que U est unimodulaire ; d'autre part, c'est une matrice triangulaire car elle est le produit de trois matrices triangulaires supérieures. La réciproque est évidente.

Une question. — De manière analogue, peut-on caractériser les matrices A à coefficients entiers, à valeurs propres entières et pour lesquelles il existe une matrice unimodulaire U telle que $U^{-1}AU$ soit une matrice de Jordan ?

J'ai fait de multiples expériences, mais je n'ai pas trouvé de loi...

Bibliographie

[1] A. BUJOSA, R. CRIADO, C. VEGA, *Jordan normal form via elementary transformations*, Siam Review, vol. 40 (1998), p.947–956.
[2] U. PITTELKOW, H.-J. RUNCKEL, *A short and constructive approach to the Jordan canonical form of a matrix*, Serdica, vol. 7 (1981), p. 348–359.
[3] E.J. PUTZER, *Avoiding the Jordan canonical form in the discussion of linear systems with constant coefficients*, American Mathematical Monthly (1966), p. 2–7.
[4] R. SÉROUL, *math_info*, InterÉditions, 1997.
[5] C. Moler et C. Van Loan, *Nineteen dubious ways to compute the exponential of a matrix*, SIAM Review , vol. 20, n° 4 (1978).