Algèbre linéaire

Bernard Parisse
parisse@fourier.ujf-grenoble.fr
Institut Fourier
UMR 5582 du CNRS
Université de Grenoble I

6 février 2004

On présente ici des algorithmes autour de la résolution exacte de systèmes (réduction des matrices sous forme échelonnée) et la recherche de valeurs propres et de vecteurs propres (diagonalisation et jordanisation des matrices).

1 Résolution de systèmes, calcul de déterminant.

1.1 La méthode du pivot de Gauß.

- Le pivot : on détermine à partir d'une ligne i la ligne j où apparait le premier coefficient non nul p dans la colonne à réduire. On échange les lignes i et j. Puis pour $j > i$ (réduction sous-diagonale) ou $j \neq i$ (réduction complète), on effectue l'opération $L_j \leftarrow L_j - \frac{p_j}{p} L_i$.
  Inconvénient : avec des données exactes de taille non bornée, la complexité des coefficients augmente plus vite qu'en choisissant le pivot le plus simple possible, (remarque, lorsque les données sont approchées, on n'utilise pas non plus cette méthode pour des raisons de stabilité numérique). Le domaine d'utilisation naturel concerne donc les coefficients dans un corps fini (par exemple $\mathbb{Z}/n\mathbb{Z}$).
- Le pivot partiel. On choisit le meilleur coefficient non nul de la colonne, où meilleur dépend du type de coefficient : avec des données exactes, on choisirait le coefficient de taille la plus petite possible, avec des données approximatives, on choisit le coefficient de plus grande norme dans la colonne. Le domaine d'utilisation naturel concerne les coefficients approchés. Pour les coefficients exacts, on remplacerait la réduction par $L_j \leftarrow pL_j - p_j L_i$ pour ne pas effectuer de division. Mais avec cette méthode, la taille des coefficients augmente de manière exponentielle. On peut améliorer la taille des coefficients intermédiaires en divisant chaque ligne par le PGCD de ses coefficients, mais comme pour le calcul du PGCD par l'algorithme du sous-résultant, il existe une méthode plus efficace présentée ci-dessous.

- La méthode de Bareiss : on initialise un coefficient b à 1. On remplace l'étape de réduction ci-dessus par $L_j \leftarrow (pL_j - p_j L_i)/b$. À la fin de l'étape de réduction, on met le coefficient b à la valeur du pivot p. L'intérêt de la méthode est que la division se fait sans introduire de fraction (la preuve peut se faire avec un système de calcul formel, comme ci-dessous). On peut utiliser cette méthode aussi bien pour la réduction sous-diagonale que pour la réduction complète (les lignes intervenant dans la combinaison linéaire subissent des modifications identiques dans les deux cas).

Montrons avec MuPAD ou xcas en mode mupad (commande `maple_mode(2)`) qu'en effet, on n'introduit pas de dénominateur dans la méthode de Bareiss. Sans restreindre la généralité, il suffit de le montrer avec une matrice 3x3 à coefficients symboliques génériques.

```
pivot:=proc (M,n,m,r) // n ligne du pivot, m colonne, r ligne a modifier
      local col,i,a,b;
       begin
         col:=ncols(M);
         a:=M[n,m];
         b:=M[r,m];
         for i from 1 to col do
           // print(i,a,b,n,m,r);
           M[r,i]:=a*M[r,i]-b*M[n,i];
         end_for;
         return(M);
       end_proc; /* End of pivot */
A:=matrix(3,3,[[a,b,c],[d,e,f],[g,h,j]]);
A:=pivot(A,1,1,2); A:=pivot(A,1,1,3); /* reduction 1ere colonne */
A:=pivot(A,2,2,3); A:=pivot(A,2,2,1); /* reduction 2eme colonne */
factor(A[3,3]);
```

Ce qui met bien en évidence le facteur a dans $A_{3,3}$.

1.2 Le déterminant.

On peut bien sûr appliquer les méthodes ci-dessus en tenant compte des pivots utilisés et du produit des coefficients diagonaux. Dans le cas de la méthode de Bareiss, si on effectue la réduction sous-diagonale uniquement, il n'est pas nécessaire de garder une trace des pivots et de calculer le produit des coefficients diagonaux, montrons que la valeur du déterminant est égal au dernier coefficient diagonal : en effet si R désigne la matrice réduite et que l'on pose $R_{0,0} = 1$, alors la réduction par la méthode de Bareiss de la colonne i a pour effet de multiplier le déterminant de la matrice initiale M par $(R_{i,i}/(R_{i-1,i-1})^{n-i}$. Donc :

$$\begin{aligned}
\det(R) &= \det(M) \prod_{i=1}^{n-1} (R_{i,i}/(R_{i-1,i-1})^{n-i} \\
\prod_{i=1}^{n} R_{i,i} &= \det(M) \prod_{i=1}^{n-1} R_{i,i} \\
R_{n,n} &= \det(M)
\end{aligned}$$

Pour les matrices à coefficients entiers, on peut aussi utiliser une méthode modulaire : on calcule une borne à priori sur le déterminant et on calcule le déterminant modulo suffisamment de petits nombres premiers pour le reconstruire par les restes chinois. L'avantage de cet algorithme est qu'il est facile à paralléliser.

On utilise souvent la borne d'Hadamard sur le déterminant :

$$|\det(M)| \leq \prod_{1\leq i\leq n} \sqrt{\sum_{1\leq j\leq n} |m_{i,j}|^2}$$

Preuve de la borne : on majore le déterminant par le produit des normes des vecteurs colonnes de M.

Remarque :
Si on veut juste prouver l'inversibilité d'une matrice à coefficients entiers, il suffit de trouver un nombre premier p tel que le déterminant de cette matrice modulo p soit non nul.

Développement par rapport à une ligne ou une colonne
On a tendance à oublier ce type de méthode car le développement complet du déterminant (faisant intervenir une somme sur toutes les permutations du groupe symétrique) nécessite d'effectuer $n!$ produits de n coefficients et $n!$ additions ce qui est gigantesque. Or on peut "factoriser" une partie des calculs et se ramener à $n.2^n$ opérations élémentaires au lieu de $n.n!$. Remarquons aussi que le nombre d'opérations élémentaires n'a guère de sens si on ne tient pas compte de la complexité des expressions, l'avantage principal de la méthode de développement étant d'éviter d'effectuer des divisions.

Calcul du déterminant par développement de Laplace
On calcule d'abord tous les mineurs 2x2 des colonnes 1 et 2 que l'on place dans une table de mineurs, puis on calcule les mineurs 3x3 des colonnes 1 à 3 en développant par rapport à la colonne 3 et en utilisant les mineurs précédents, puis les mineurs 4x4 avec les mineurs 3x3, etc.. On évite ainsi de recalculer plusieurs fois les mêmes mineurs. Cf. par exemple l'implémentation en C++ dans giac/xcas (`www-fourier.ujf-grenoble.fr/~parisse/giac.html`) qui utilise le type générique `map<>` de la librairie standard C++ (STL) pour stocker les tables de mineurs (fonction `det_minor` du fichier `vecteur.cc`).
Nombre d'opérations élémentaires : il y a $\binom{n}{2}$ mineurs d'ordre 2 à calculer nécessitant chacun 2 multiplications (et 1 addition), puis $\binom{n}{3}$ mineurs d'ordre 3

nécessitant 3 multiplications et 2 additions, etc. donc le nombre de multiplications est de $2\binom{n}{2}+3\binom{n}{3}+...+n\binom{n}{n}$, celui d'additions est $\binom{n}{2}+2\binom{n}{3}+...+(n-1)\binom{n}{n}$ soit un nombre d'opérations élémentaires majoré par $n.2^n$.

On observe "expérimentalement" que cet algorithme est intéressant lorsque le nombre de paramètres dans le déterminant est grand et que la matrice est plutôt creuse (majorité de coefficients nuls). Il existe des heuristiques de permutation des lignes ou des colonnes visant à optimiser la position des zéros (par exemple, les auteurs de GiNaC (`www.ginac.de`) suite à des expérimentations privilégient la simplification des petits mineurs en mettant les colonnes contenant le maximum de zéros à gauche selon la description faite ici).

Pour se convaincre de l'intérêt de cet algorithme, on peut effectuer le test O1 de Lewis-Wester
`http://www.bway.net/~lewis/calatex.html`
il s'agit de calculer un déterminant de taille 15 avec 18 paramètres.

1.3 Systèmes linéaires

On peut appliquer la méthode du pivot de Gauß ou les règles de Cramer. Pour les systèmes à coefficients entiers non singuliers, on peut aussi utiliser une méthode p-adique asymptotiquement plus efficace. On calcule d'abord une borne sur les coefficients des fractions solutions de l'équation $Ax = b$ en utilisant les règles de Cramer et la borne d'Hadamard. On calcule ensuite l'inverse de A modulo p (en changeant de p si A n'est pas inversible modulo p), puis, si

$$x = \sum_i x_i p^i, \quad A(\sum_{i<k} x_i p^i) = b \pmod{p^k}$$

on ajoute $x_k p^k$ et on obtient l'équation :

$$Ax_k = \frac{b - \sum_{i<k} x_i p^i}{p^k} \pmod{p}$$

qui détermine x_k. On s'arrête lorsque k est suffisamment grand pour pouvoir reconstruire les fractions à l'aide de l'identité de Bézout (cf. l'appendice).

1.4 Base du noyau

On commence bien sûr par réduire la matrice (réduction complète en-dehors de la diagonale), et on divise chaque ligne par son premier coefficient non nul (appelé pivot). On insère alors des lignes de 0 pour que les pivots (non nuls) se trouvent sur la diagonale. Puis en fin de matrice, on ajoute ou on supprime des lignes de 0 pour avoir une matrice carrée de dimension le nombre de colonnes de la matrice de départ. On parcourt alors la matrice en diagonale. Si le i-ième coefficient est non nul, on passe au suivant. S'il est nul, alors tous les coefficients d'indice supérieur ou égal à i du i-ième vecteur colonne v_i sont nuls (mais pas forcément pour les indices inférieurs à i). Si on remplace le i-ième coefficient de v_i par -1, il est facile de se convaincre que c'est un vecteur du noyau, on le rajoute donc à la base du noyau. On voit facilement que tous les vecteurs de ce type forment une famille libre de la bonne taille, c'est donc bien une base du noyau.

2 Réduction des endomorphismes

2.1 Le polynôme minimal

On prend un vecteur v au hasard et on calcule la relation linéaire de degré minimal entre v, Av, ..., $A^n v$ en cherchant le premier vecteur w du noyau de la matrice obtenue en écrivant les vecteurs v, Av, etc. en colonne dans cet ordre. Les coordonnées de w donnent alors par ordre de degré croissant un polynôme P de degré minimal tel que $P(A)v = 0$ donc P divise le polynôme minimal M. Donc si P est de degré n, $P = M$. Sinon, il faut vérifier que le polynôme obtenu annule la matrice A. On peut aussi calculer en parallèle le polynôme P précédent pour quelques vecteurs aléatoires et prendre le PPCM des polynômes obtenus.

Exemple 1

Polynôme minimal de $\begin{pmatrix} 1 & -1 \\ 2 & 4 \end{pmatrix}$. On prend $v = (1, 0)$, la matrice à réduire est alors :

$$\begin{pmatrix} 1 & -1 & -11 \\ 2 & 10 & 38 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & -6 \\ 0 & 1 & 5 \end{pmatrix}$$

Le noyau est engendré par $(-6, 5, -1)$ donc $P = -x^2 + 5x - 6$.

Exemple 2

$$A = \begin{pmatrix} 3 & 2 & -2 \\ -1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

en prenant $v = (1, 0, 0)$ on obtient la matrice :

$$A = \begin{pmatrix} 1 & 3 & 5 & 7 \\ 0 & -1 & -2 & -3 \\ 0 & 1 & 2 & 3 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & -1 & -2 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

le permier vecteur du noyau est $(-1, 2, -1)$ d'où un polynôme divisant le polynôme minimal $-x^2 + 2x - 1$.

2.2 Le polynôme caractéristique

Pour une matrice générique, le polynôme caractéristique est égal au polynôme minimal, il est donc intéressant de chercher si le polynôme annulateur de A sur un vecteur aléatoire est de degré n, car le temps de calcul du polynôme caractéristique est alors en $O(n^3)$. Si cette méthode probabiliste échoue, on se rabat sur une des méthode déterministe ci-dessous :

- on utilise la formule $\det(\lambda I - A)$ déterminé par une des méthodes de calcul de déterminant ci-dessus. Cela nécessite $O(n^3)$ opérations mais avec des coefficients polynômes en λ.
- on fait une interpolation de Lagrange en donnant $n+1$ valeurs distinctes à λ. Ce qui nécessite $O(n^4)$ opérations mais avec des coefficients indépendants de λ, de plus cette méthode est facile à programmer de manière parallèle.
- si la matrice est à coefficients entiers on peut utiliser la méthode de Hessenberg (voir ci-dessous), on calcule une borne à priori sur les coefficients du polynôme caractéristique (cf. Cohen p.58-59) :

  $$|P_k| \leq \left(\begin{array}{c} n \\ n-k \end{array} \right) (n-k)^{(n-k)/2} |M|^{n-k} \ ,$$

  on calcule le polynôme caractéristique modulo suffisamment de petits entiers puis on remonte par les restes chinois.

2.3 La méthode de Hessenberg

Pour les matrices à coefficients de taille bornée (modulaires par exemple) on préfère la méthode de Hessenberg qui est plus efficace, car elle nécessite de l'ordre de n^3 opérations sur les coefficients.

On se raméne d'abord à une matrice triangulaire supérieure à une diagonale près qui est semblable à la matrice de départ puis on applique une formule de récurrence pour calculer les coefficients du polynôme caractéristique.

Algorithme de réduction de Hessenberg :
Dans une colonne m donnée de la matrice H, on cherche à partir de la ligne $m+1$ un coefficient non nul. S'il n'y en a pas on passe à la colonne suivante. S'il y en a un en ligne i, on échange les lignes $m+1$ et i et les colonnes $m+1$ et i. Ensuite pour tout $i \geq m+2$, soit $u = H_{i,m}/H_{m+1,m}$, on remplace alors la ligne L_i de H par $L_i - uL_{m+1}$ et la colonne C_{m+1} par $C_{m+1} + uC_i$ ce qui revient "à remplacer le vecteur e_{m+1} de la base par le vecteur $e_{m+1}+ue_i$" ou plus précisément à multiplier à gauche par $\left(\begin{array}{cc} 1 & 0 \\ -u & 1 \end{array} \right)$ et à droite par la matrice inverse $\left(\begin{array}{cc} 1 & 0 \\ u & 1 \end{array} \right)$ (en utilisant les lignes et colonnes $m+1$ et i au lieu de 1 et 2 pour ces matrices). Ceci a pour effet d'annuler le coefficient $H_{i,m}$ dans la nouvelle matrice.

On obtient ainsi en $O(n^3)$ opérations une matrice H' semblable à H de la forme :

$$\left(\begin{array}{cccccc} H'_{1,1} & H'_{1,2} & ... & H'_{1,n-2} & H'_{1,n-1} & H'_{1,n} \\ H'_{2,1} & H'_{2,2} & ... & H'_{2,n-2} & H'_{2,n-1} & H'_{2,n} \\ 0 & H'_{3,2} & ... & H'_{3,n-2} & H'_{3,n-1} & H'_{3,n} \\ 0 & 0 & ... & H'_{4,n-2} & H'_{4,n-1} & H'_{4,n} \\ \vdots & \vdots & ... & \vdots & \vdots & \vdots \\ 0 & 0 & ... & 0 & H'_{n,n-1} & H'_{n,n} \end{array} \right)$$

On calcule alors le polynôme caractéristique de H' par une récurrence qui s'obtient en développant le déterminant par rapport à la dernière colonne :

$$\begin{aligned} h_n(\lambda) = \det(\lambda I_n - H) &= (\lambda - H'_{n,n})h_{n-1}(\lambda) - (-H'_{n-1,n})(-H'_{n,n-1})h_{n-2}(\lambda) + \\ &\quad +(-H'_{n-2,n})(-H'_{n,n-1})(-H'_{n-1,n-2})h_{n-3}(\lambda) - ... \end{aligned}$$

où les h_i s'entendent en gardant les i premières lignes/colonnes de H'. On peut écrire cette formule pour $m \leq n$:

$$h_m(\lambda) = (\lambda - H'_{m,m})h_{m-1}(\lambda) - \sum_{i=1}^{m-1} H'_{m-i,m} \prod_{j=1}^{i-1} H'_{m-j+1,m-j} h_{i-1}(\lambda)$$

Pour effectuer cette récurrence de manière efficace, on conserve les $h_m(\lambda)$ dans un tableau de polynômes et on utilise une variable produit contenant successivement les $\prod H'_{m-j+1,m-j}$.

2.4 La méthode de Leverrier-Faddeev-Souriau

Cette méthode permet le calcul simultané des coefficients p_i $(i = 0..n)$ du polynôme caractéristique $P(\lambda) = \det(\lambda I - A)$ et des coefficients matriciels B_i $(i = 0..n-1)$ du polynôme en λ donnant la matrice adjointe (ou comatrice) $B(\lambda)$ de $\lambda I - A$:

$$(\lambda I - A)B(\lambda) = (\lambda I - A) \sum_{k \leq n-1} B_k \lambda^k = (\sum_{k \leq n} p_k \lambda^k) I = P(\lambda) I \qquad (1)$$

Remarquons que cette équation donne une démonstration assez simple de Cayley-Hamilton puisque le reste de la division euclidienne du polynôme $P(\lambda)I$ par $\lambda I - A$ est $P(A)$.

Pour déterminer simultanément les p_k et B_k, on a les relations de récurrence :

$$B_{n-1} = p_n I = I, \quad B_k - AB_{k+1} = p_{k+1} I \qquad (2)$$

Il nous manque une relation entre les p_k et B_k pour pouvoir faire le calcul par valeurs décroissantes de k, on va montrer le :

Théorème 1 *La dérivée du polynôme caractéristique $P'(\lambda)$, est égale à la trace de la matrice adjointe de $\lambda I - A$*

$$tr(B) = P'(\lambda)$$

Le théorème nous donne $\mathrm{tr}(B_k) = (k+1)p_{k+1}$. Si on prend la trace de (2), on a :

$$\mathrm{tr}(B_{n-1}) = np_n, \quad (k+1)p_{k+1} - \mathrm{tr}(AB_{k+1}) = np_{k+1}$$

donc on calcule p_{k+1} en fonction de B_{k+1} puis B_k :

$$p_{k+1} = \frac{\mathrm{tr}(AB_{k+1})}{k+1-n}, \quad B_k = AB_{k+1} + p_{k+1} I$$

Démonstration du théorème :
Soient $V_1(\lambda), ...V_n(\lambda)$ les vecteurs colonnes de $\lambda I - A$ et $b_{i,j}(\lambda)$ les coefficients de B, on a :

$$\begin{array}{rcl} P'(\lambda_0) & = & \det(V_1(\lambda), V_2(\lambda), ..., V_n(\lambda))'_{|\lambda=\lambda_0} \\ & = & \det(V_1'(\lambda_0), V_2(\lambda_0), ..., V_n(\lambda_0)) + \det(V_1(\lambda_0), V_2'(\lambda_0), ..., V_n(\lambda_0)) + \\ & & +... + \det(V_1(\lambda_0), V_2(\lambda_0), ..., V_n'(\lambda_0)) \end{array}$$

Il suffit alors de remarquer que $V_i'(\lambda_0)$ est le i-ième vecteur de la base canonique donc :

$$\det(V_1(\lambda_0), V_2(\lambda_0), ..., V_i'(\lambda_0), ..., V_n(\lambda_0)) = b_{i,i}(\lambda_0)$$

Finalement :

$$P'(\lambda_0) = \sum_{i=1}^{n} b_{i,i}(\lambda_0) = \text{trace } (B(\lambda_0))$$

Remarque :
En réindexant les coefficients de P et B de la manière suivante :

$$\begin{array}{rcl} P(\lambda) & = & \lambda^n + p_1\lambda^{n-1} + p_2\lambda^{n-2}... + p_n \\ B(\lambda) & = & \lambda^{n-1}I + \lambda^{n-2}B_1 + ... + B_{n-1} \end{array}$$

on a montré que :

$$\left\{ \begin{array}{ccc} A_1 = A, & p_1 = -\text{tr}(A), & B_1 = A_1 + p_1 I \\ A_2 = AB_1, & p_2 = -\frac{1}{2}\text{tr}(A_2), & B_2 = A_2 + p_2 I \\ \vdots & \vdots & \vdots \\ A_k = AB_{k-1}, & p_k = -\frac{1}{k}\text{tr}(A_k), & B_k = A_k + p_k I \end{array} \right.$$

On peut alors vérifier que $B_n = A_n + p_n I = 0$. D'où ce petit programme à utiliser avec xcas en mode mupad (`maple_mode(2);`), ou avec MuPAD, ou à adapter avec un autre système :

```
iequalj:=(j,k)->if j=k then return(1); else return(0); end_if;
faddeev:=proc(A) // renvoie la liste des matrices B et le polynome P
local Aj,AAj,Id,coef,n,pcara,lmat;
begin
 n:=ncols(A);
 Id:=matrix(n,n,iequalj);     // matrice identite
 Aj:=Id;
 lmat:=[];                    // B initialise a liste vide
 pcara:=[1];                  // coefficient de plus grand degre de P
 for j from 1 to n do
  lmat:=append(lmat,Aj);      // rajoute Aj a la liste de matrices
  AAj:=Aj*A;
  coef:=-trace(AAj)/j;        // mupad linalg::tr
  pcara:=append(pcara,coef);  // rajoute coef au polynome caracteristique
```

```
  Aj:=AAj+coef*Id;
 end_for;
 lmat,pcara;                    // resultat
end_proc;
```

2.5 Les vecteurs propres simples.

On suppose ici qu'on peut factoriser le polynôme caractéristique (ou calculer dans une extension algébrique d'un corps). Lorsqu'on a une valeur propre simple λ_0, en écrivant la relation $(A-\lambda_0 I)B(\lambda_0) = P(\lambda_0)I = 0$, on voit que les vecteurs colonnes de la matrice $B(\lambda_0)$ sont vecteurs propres. Remarquer que $B(\lambda_0) \neq 0$ sinon on pourrait factoriser $\lambda - \lambda_0$ dans $B(\lambda)$ et apres̀ simplifications on aurait :

$$(A - \lambda_0 I)\frac{B}{\lambda - \lambda_0}(\lambda_0) = \frac{P}{\lambda - \lambda_0}(\lambda_0)I$$

or le 2ème membre est inversible en λ_0 ce qui n'est pas le cas du premier. Pour avoir une base des vecteurs propres associés à λ_0, on calcule $B(\lambda_0)$ par la méthode de Horner appliquée au polynôme $B(\lambda)$ en $\lambda = \lambda_0$, et on réduit en colonnes la matrice obtenue.

2.6 La forme normale de Jordan

Pour les valeurs propres de multiplicité plus grande que 1, on souhaiterait généraliser la méthode ci-dessus pour obtenir une base de l'espace caractéristique, sous forme de cycles de Jordan. Soit λ_i, n_i les valeurs propres comptées avec leur multiplicité. On fait un développement de Taylor en λ_i :

$$\begin{aligned} -P(\lambda)I &= (A - \lambda I)\left(B(\lambda_i) + B'(\lambda_i)(\lambda - \lambda_i) + \ldots + \frac{B^{(n-1)}(\lambda_i)}{(n-1)!}(\lambda - \lambda_i)^{n-1}\right) \\ &= -(\lambda - \lambda_i)^{n_i}\prod_{j\neq i}(\lambda - \lambda_j)^{n_j} I \end{aligned}$$

Comme $A-\lambda I = A-\lambda_i I-(\lambda-\lambda_i)I$, on obtient pour les n_i premières puissances de $\lambda - \lambda_i$:

$$\begin{aligned} (A - \lambda_i I)B(\lambda_i) &= 0 && (3) \\ (A - \lambda_i I)B'(\lambda_i) &= B(\lambda_i) && (4) \\ &\ldots && (5) \\ (A - \lambda_i I)\frac{B^{(n_i-1)}(\lambda_i)}{(n_i-1)!} &= \frac{B^{(n_i-2)}(\lambda_i)}{(n_i-2)!} && (6) \\ (A - \lambda_i I)\frac{B^{(n_i)}(\lambda_i)}{n_i!} - \frac{B^{(n_i-1)}(\lambda_i)}{(n_i-1)!} &= -\prod_{j\neq i}(\lambda_i - \lambda_j)^{n_j} I && (7) \end{aligned}$$

Le calcul des matrices $B^{(n)}(\lambda_i)/n!$ pour $n < n_i$ se fait en appliquant n_i fois l'algorithme de Horner (avec reste).

Théorème 2 *L'espace caractéristique de λ_i est égal à l'image de $B^{(n_i-1)}(\lambda_i)/(n_i-1)!$.*

Preuve :
On montre d'abord que $\mathrm{Im} B^{(n_i-1)}(\lambda_i)/(n_i-1)!$ est inclus dans l'espace caractéristique correspondant à λ_i en appliquant l'équation (6) et les équations précédentes. Réciproquement on veut prouver que tout vecteur caractéristique v est dans l'image de $B^{(n_i-1)}(\lambda_i)/(n_i-1)!$. Prouvons le par récurrence sur le plus petit entier m tel que $(A-\lambda_i)^m v = 0$. Le cas $m=0$ est clair puisque $v=0$. Supposons le cas m vrai, prouvons le cas $m+1$. On applique l'équation (7) à v, il suffit alors de prouver que

$$w = (A-\lambda_i)\frac{B^{(n_i)}(\lambda_i)}{n_i!}v$$

appartient à l'image de $B^{(n_i-1)}(\lambda_i)/(n_i-1)!$. Comme $B^{(n_i)}(\lambda_i)$ commute avec A (car c'est un polynôme en A ou en appliquant le fait que $B(\lambda)$ inverse de $A-\lambda I$) :

$$(A-\lambda_i)^m w = \frac{B^{(n_i)}(\lambda_i)}{n_i!}(A-\lambda_i)^{m+1}v = 0$$

et on applique l'hypothèse de récurrence à w.

Pour calculer les cycles de Jordan, nous allons effectuer une réduction par le pivot de Gauß simultanément sur les colonnes des matrices $B^{(k)}(\lambda_i)/k!$ où $k < n_i$. La simultanéité a pour but de conserver les relations (3) à (6) pour les matrices réduites. Pour visualiser l'algorithme, on se représente les matrices les unes au-dessus des autres, colonnes alignées. On commence par réduire la matrice $B(\lambda_i)$ jusqu'à ce que l'on obtienne une matrice réduite **en recopiant** les opérations élémentaires de colonnes faites sur $B(\lambda_i)$ sur toutes les matrices $B^{(k)}(\lambda_i)/k!$. On va continuer avec la liste des matrices réduites issues de $B'(\lambda_i)$, ..., $B^{(n_i-1)}(\lambda_i)/(n_i-1)!$, mais en déplacant les colonnes non nulles de $B(\lambda_i)$ d'une matrice vers le bas (pour une colonne non nulle de la matrice réduite $B(\lambda)$ les colonnes correspondantes de $B^{(k)}(\lambda_i)$ réduite sont remplacées par les colonnes correspondantes de $B^{(k-1)}(\lambda_i)$ réduite pour k décroissant de n_i-1 vers 1). À chaque étape, on obtient une famille (éventuellement vide) de cycles de Jordan, ce sont les vecteurs colonnes correspondants aux colonnes non nulles de la matrice réduite du haut de la colonne. On élimine bien sûr les colonnes correspondant aux fins de cycles déjà trouvés.

Par exemple, si $B(\lambda_i) \neq 0$, son rang est 1 et on a une colonne non nulle, et un cycle de Jordan de longueur n_i fait des n_i vecteurs colonnes des matrices $B^{(k)}(\lambda_i)/k!$ réduites. Plus généralement, on obtiendra plus qu'un cycle de Jordan (et dans ce cas $B(\lambda_i)=0$).

2.6.1 Exemple 1

$$A = \begin{pmatrix} 3 & -1 & 1 \\ 2 & 0 & 1 \\ 1 & -1 & 2 \end{pmatrix}$$

$\lambda = 2$ est valeur propre de multiplicité 2, on obtient :

$$B(\lambda) = \lambda^2 I + \lambda \begin{pmatrix} -2 & -1 & 1 \\ 2 & -5 & 1 \\ 1 & -1 & -3 \end{pmatrix} + \begin{pmatrix} 1 & 1 & -1 \\ -3 & 5 & -1 \\ -2 & 2 & 2 \end{pmatrix}$$

on applique l'algorithme de Horner :

$$\begin{aligned} B(2) &= \begin{pmatrix} 1 & -1 & 1 \\ 1 & -1 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \\ B'(2) &= \begin{pmatrix} 2 & -1 & 1 \\ 2 & -1 & 1 \\ 1 & -1 & 1 \end{pmatrix} \end{aligned}$$

Comme $B(2) \neq 0$, on pourrait arrêter les calculs en utilisant une colonne non nulle et le cycle de Jordan associé $(2,2,1) \to (1,1,0) \to (0,0,0)$. Expliquons tout de même l'algorithme général sur cet exemple. La réduction de $B(2)$ s'obtient en effectuant les manipulations de colonnes $C_2 + C_1 \to C_2$ et $C_3 - C_1 \to C_3$. On effectue les mêmes opérations sur $B'(2)$ et on obtient :

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \\ \begin{pmatrix} 2 & 1 & -1 \\ 2 & 1 & -1 \\ 1 & 0 & 0 \end{pmatrix}$$

L'étape suivante consiste à déplacer vers le bas d'une matrice les colonnes non nulles de la matrice du haut, on obtient :

$$\begin{pmatrix} 1 & 1 & -1 \\ 1 & 1 & -1 \\ 0 & 0 & 0 \end{pmatrix}$$

qui se réduit en :

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

on chercherait alors dans les colonnes 2 et 3 de nouveaux cycles (puisque la colonne 1 a déja été utilisée pour fournir un cycle).

2.6.2 Exemple 2

$$A = \begin{pmatrix} 3 & 2 & -2 \\ -1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

$\lambda = 1$ est valeur propre de multiplicité 3. On trouve :

$$\begin{aligned} B(1) &= \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \\ B'(1) &= \begin{pmatrix} 2 & 2 & -2 \\ -1 & -1 & 1 \\ 1 & 1 & -1 \end{pmatrix}, \\ \frac{B''(1)}{2} &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{aligned}$$

Le processus de réduction commence avec $B'(1)$ en haut de la liste de matrices, on effectue les opérations élémentaires de colonne $C_2 - C_1 \to C_2$ et $C_3 + C_1 \to C_3$ et on obtient :

$$\begin{pmatrix} 2 & 0 & 0 \\ -1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix},$$
$$\begin{pmatrix} 1 & -1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

La première colonne donne le premier cycle de Jordan $(1, 0, 0) \to (2, -1, 1)$. On déplace les premières colonnes d'une matrice vers le bas :

$$\begin{pmatrix} 2 & -1 & 1 \\ -1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

qu'on réduit par les opérations $2C_2 + C_1 \to C_2$ et $2C_3 - C_1 \to C_3$ en :

$$\begin{pmatrix} 2 & 0 & 0 \\ -1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Puis on effectue $C_3 - C_2 \to C_3$ et la deuxième colonne nous donne le deuxième cycle de Jordan, réduit ici à un seul vecteur propre $(0, 1, 1)$.

2.7 Le polynôme minimal par Faddeev

On vérifie aisément que le degré du facteur $(\lambda - \lambda_i)$ dans le polynôme minimal de A est égal à $n_i - k$ où k est le plus grand entier tel que :

$$\forall j < k, \quad B^{(j)}(\lambda_i) = 0$$

2.8 Formes normales rationnelles

On se place ici dans une problématique différente : trouver une matrice semblable la plus simple possible sans avoir à introduire d'extension algébrique pour factoriser le polynôme caractéristique. Quitte à "compléter" plus tard la factorisation et la jordanisation à partir de la forme simplifiée. Il existe diverses formes associées à une matrice et plusieurs algorithmes permettant de les relier entre elles, forme de Smith, de Frobenius, forme normale de Jordan rationnelle.

On va présenter une méthode directe de calcul d'une forme normale contenant le maximum de zéros (dont la forme dite normale de Jordan rationnelle peut se déduire) en utilisant le même algorithme que pour la forme normale de Jordan. Soit $Q(\lambda) = q_0 + ... + q_d\lambda^d$ un facteur irréductible de degré d et de multiplicité q du polynôme caractéristique P. Il s'agit de construire un sous-espace de dimension dq formé de "cycles de Jordan rationnels". On part toujours de la relation $(\lambda I - A)\sum_{k\leq n-1} B_k\lambda^k = P(\lambda)I$. On observe que $Q(\lambda)I - Q(A)$ est divisible par $(\lambda I - A)$ donc il existe une matrice $M(\lambda)$ telle que :

$$(Q(\lambda)I - Q(A))(\sum_{k\leq n-1} B_k\lambda^k) = Q(\lambda)^q M(\lambda)$$

On observe aussi que Q a pour coefficient dominant 1 puisqu'il divise P, on peut donc effectuer des divisions euclidiennes de polynômes donc de polynômes à coefficients matriciels par Q sans avoir à diviser des coefficients. Ce qui nous permet de décomposer $B(\lambda) = \sum_{k\leq n-1} B_k\lambda^k$ en puissances croissantes de Q :

$$B(\lambda) = \sum_k C_k(\lambda)Q(\lambda)^k, \quad \deg(C_k) < q$$

On remplace et on écrit que les coefficients des puissances inférieures à q de Q sont nulles (la k-ième étant non nulle car $M(\lambda)$ n'est pas divisible par Q pour les mêmes raisons que pour la forme normale de Jordan). On a donc les relations :

$$Q(A)C_0 = 0, \quad C_k = Q(A)C_{k+1}$$

ce qui donne une colonne de matrice $C_{q-1} \to C_{q-2} ... \to C_0 \to 0$ qui sont images l'une de l'autre en appliquant $Q(A)$. On peut alors faire l'algorithme de réduction simultanée sur les colonnes des C_j. On observe ensuite que le nombre de cycles de Jordan de $Q(A)$ de longueur donnée est un multiple de d, en effet il suffit de multiplier un cycle par A, ..., A^{d-1} pour créer un autre cycle, de plus ces cycles forment des familles libres car on a supposé Q irréductible. On peut donc choisir pour un cycle de longueur k des bases de la forme $(v_{k-1}, Av_{k-1}..., A^{d-1}v_{k-1}) \to ... \to (v_0, Av_0..., A^{d-1}v_0) \to (0, ..., 0)$ où la flèche $\to$ désigne l'image par $Q(A)$. Si on écrit la matrice de A dans la base $v_0, Av_0..., A^{d-1}v_0, ..., v_{k-1}, Av_{k-1}..., A^{d-1}v_{k-1}$ on obtient un "quasi-bloc de Jordan rationnel" de taille kd multiple de d :

$$\begin{pmatrix}
0 & 0 & ... & -q_0 & 0 & 0 & ... & 1 & ... \\
1 & 0 & ... & -q_1 & 0 & 0 & ... & 0 & ... \\
0 & 1 & ... & -q_2 & 0 & 0 & ... & 0 & ... \\
\vdots & \vdots & ... & \vdots & \vdots & \vdots & ... & \vdots & ... \\
0 & 0 & ... & -q_{d-1} & 0 & 0 & ... & 0 & ... \\
 & & & & & & & & \\
0 & 0 & ... & 0 & 0 & 0 & ... & -q_0 & ... \\
0 & 0 & ... & 0 & 1 & 0 & ... & -q_1 & ... \\
\vdots & \vdots & ... & \vdots & \vdots & \vdots & ... & \vdots & ...
\end{pmatrix}$$

Exemple

Soit la matrice

$$A = \begin{pmatrix}
1 & -2 & 4 & -2 & 5 & -4 \\
0 & 1 & \frac{5}{2} & \frac{-7}{2} & 2 & \frac{-5}{2} \\
1 & \frac{-5}{2} & 2 & \frac{-1}{2} & \frac{5}{2} & -3 \\
0 & -1 & \frac{9}{2} & \frac{-7}{2} & 3 & \frac{-7}{2} \\
0 & 0 & 2 & -2 & 3 & -1 \\
1 & \frac{-3}{2} & \frac{-1}{2} & 1 & \frac{3}{2} & \frac{1}{2}
\end{pmatrix}$$

Son polynôme caractéristique est $(x-2)^2(x^2-2)^2$ et on va déterminer la partie bloc de Jordan rationnel correspondant au facteur irréductible sur les entiers $Q(x) = (x^2-2)$ de multiplicité $q = 2$. On calcule $B(x)$ et l'écriture de B comme somme de puissances de Q (ici avec `xcas` en mode `xcas`) :

```
A:=[[1,-2,4,-2,5,-4],[0,1,5/2,(-7)/2,2,(-5)/2],[1,(-5)/2,2,1/(-2),5/2,-3],
    [0,-1,9/2,(-7)/2,3,(-7)/2],[0,0,2,-2,3,-1],[1,(-3)/2,1/(-2),1,3/2,1/2]];
P:=det(A-x*idn(6));
B:=normal(P*inv(A-x*idn(6))); // preferer un appel a faddeev bien sur!
ecriture(B,Q,q):={
  local j,k,l,n,C,D,E;
  C:=B;
  D:=B;
  E:=NULL;
  n:=coldim(B);
  for (j:=0;j<q;j++){
    for (k:=0;k<n;k++){
      for (l:=0;l<n;l++){
        D[k,l]:=rem(C[k,l],Q,x);
        C[k,l]:=quo(C[k,l],Q,x);
      }
    }
    E:=E,D;
  }
  return E;
};
```

```
E:=ecriture(B,x^2-2,2);
QA:=A*A-2*idn(6);
```

On vérifie bien que `normal(QA*E(0))` et `normal(QA*E(1))-E(0))` sont nuls. On sait qu'on a un bloc de taille 2 de cycles de Jordan de longueur 2, donc il n'est pas nécessaire de faire des réductions ici, il suffit de prendre une colonne non nulle de $E(0)$, par exemple la première colonne en $x = 0$ et la colonne correspondante de $E(1)$ et leurs images par A, ici cela donne $(4, 24, 12, 32, 8, -4)$ correspondant à $(0, 4, -4, 8, 4, -4)$, on calcule les images par A, la matrice de l'endomorphisme restreint à ce sous-espace est alors le bloc de taille 4 :

$$\begin{pmatrix} 0 & 2 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Cette forme normale minimise le nombre de coefficients non nuls, mais présente un inconvénient, la partie nilpotente ne commute pas avec la partie bloc-diagonale, contrairement à la forme normale rationnelle de Jordan qui contient des blocs identités au-dessus de la diagonale de blocs. Pour créer la forme normale rationnelle de Jordan, on doit donc remplacer les blocs $\begin{pmatrix} ... & 0 & 1 \\ ... & 0 & 0 \\ ... & & \end{pmatrix}$ par des matrices identités. Supposons constitués les j premiers blocs de taille d numérotés de 0 à $j-1$ avec comme base de vecteurs $(v_{0,0}, ..., v_{0,d-1}, ..., v_{j-1,d-1})$. Il s'agit de trouver un vecteur $v_{j,0}$ pour commencer le bloc suivant. On définit alors $v_{j,l}$ en fonction de $v_{j,l-1}$ en appliquant la relation $Av_{j,l-1} = v_{j,l} + v_{j-1,l-1}$. Il faut donc chercher $v_{j,0}$ tel que

$$Av_{j,d-1} = -q_0 v_{j,0} - ... - q_{d-1} v_{j,d-1} + v_{j-1,d-1} \tag{8}$$

En utilisant les relations de récurrence précédentes, on voit que cela revient à fixer $Q(A)v_{j,0}$ en fonction des $v_{j',l}$ avec $j' < j$ (l quelconque). Ce qui est toujours possible en utilisant la colonne de matrices $C_{j'}$ qui s'obtiennent en fonction des $C_{j'+1}$ en appliquant $Q(A)$.

Plus précisément, calculons les $v_{j,l}$ en fonction de $v_{j,0}$ et des $v_{j',l'}$ ($j' < j$). On utilise les coefficients binomiaux $\binom{l}{m}$ calculés par la règle du triangle de Pascal et on montre par récurrence que :

$$v_{j,l} = A^l v_{j,0} - \sum_{m=1}^{\inf(l,j)} \binom{l}{m} v_{j-m,l-m} \tag{9}$$

On remplace dans (8) d'où :

$$A^d v_{j,0} - \sum_{m=1}^{\inf(d,j)} \binom{d}{m} v_{j-m,l-m} + \sum_{l=0}^{d} q_l (A^l v_{j,0} - \sum_{m=1}^{\inf(l,j)} \binom{l}{m} v_{j-m,l-m}) = 0$$

finalement :

$$Q(A)v_{j,0} = \sum_{l=1}^{d} q_l \sum_{m=1}^{\inf(l,j)} \binom{l}{m} v_{j-m,l-m} \tag{10}$$

Application à l'exemple :
Ici $v_{0,0} = (4, 24, 12, 32, 8, -4)$ et $v_{0,1} = Av_{j,0}$ dont une préimage par $Q(A)$ est $w_{1,0} = (0, 4, -4, 8, 4, -4)$ et $w_{1,1} = Aw_{1,0}$. On applique (10), comme $q_1 = 0$ et $q_2 = 1$ on doit avoir :

$$Q(A)v_{1,0} = \sum_{l=1}^{2} q_l \sum_{m=1}^{\inf(l,1)} \binom{l}{m} v_{1-m,l-m} = 2v_{0,1}$$

donc :

$$\begin{array}{rcccc} v_{1,0} & = & 2A(0,4,-4,8,4,-4) & = & (-8,-32,0,-48,-16,16) \\ v_{1,1} & = & Av_{1,0} - v_{0,0} & = & (4,40,-4,64,24,-20) \end{array}$$

On vérifie bien que $Av_{1,1} = 2v_{1,0} + v_{0,1}$.

2.9 Fonctions analytiques

Soit f une fonction analytique et M une matrice. Pour calculer $f(M)$, on calcule la forme normale de Jordan de $M = P(D+N)P^{-1}$ où $D = \mathrm{diag}(d_1, ..., d_m)$ est diagonale et N nilpotente d'ordre n. On calcule aussi le développement de Taylor formel de f en x à l'ordre $n-1$, on a alors :

$$f(N) = P\left(\sum_{j=0}^{n-1} \frac{\mathrm{diag}(f^{(j)}(d_1), ..., f^{(j)}(d_m))}{j!} N^j\right) P^{-1}$$

3 Quelques autres algorithmes utiles

Pour calculer le produit de matrices, on peut utiliser l'algorithme de Strassen, on présente ici la variante de Winograd. Soit à calculer :

$$\begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{pmatrix} = \begin{pmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{pmatrix}$$

On calcule :

$$s_1 = a_{2,1} + a_{2,2}, \quad s_2 = s_1 - a_{1,1}, \quad s_3 = a_{1,1} - a_{2,1}, \quad s_4 = a_{1,2} - s_2$$
$$t_1 = b_{1,2} - b_{1,1}, \quad t_2 = b_{2,2} - t_1, \quad t_3 = b_{2,2} - b_{1,2}, \quad t_4 = b_{2,1} - t_2$$

puis :

$$p_1 = a_{1,1}b_{1,1}, \quad p_2 = a_{1,2}b_{2,1}, \quad p_3 = s_1t_1, \quad p_4 = s_2t_2$$
$$p_5 = s_3t_3, \quad p_6 = s_4b_{2,2}, \quad p_7 = a_{2,2}t_4$$
$$u_1 = p_1 + p_2 \quad u_2 = p_1 + p_4, \quad u_3 = u_2 + p_5, \quad u_4 = u_3 + p_7$$
$$u_5 = u_3 + p_3, \quad u_6 = u_2 + p_3, \quad u_7 = u_6 + p_6$$

Alors $c_{1,1} = u_1, c_{1,2} = u_7, c_{2,1} = u_4, c_{2,2} = u_5$.
Cet algorithme utilise 7 multiplications et 15 additions ce qui économise 1 multiplication et permet en appliquant récursivement cet algorithme pour des matrices blocs de réduire la complexité d'un produit de grandes matrices normalement en $O(n^3)$ à $O(n^{\ln(7)})$ (la preuve est analogue à celle de la multiplication des polynômes par l'algorithme de Karatsuba).

La plupart des algorithmes d'algèbre linéaire "numérique" ont une utilité en calcul exact : par exemple la factorisation LU (avec les variations décrites dans la section réduction de Gauß), la factorisation QR (et donc la méthode de Gram-Schmidt, ici pour des raisons d'efficacité on orthogonalise d'abord la base de départ et on la normalise à la fin seulement), Cholesky,.... On peut aussi facilement programmer la recherche de la décomposition ${}^tPDP$ d'une matrice symétrique et en déduire la signature d'une forme quadratique. Citons enfin l'algorithme LLL (cf. Cohen) qui est utile dans de nombreux domaines (il permet de trouver des vecteurs assez courts dans un réseau, ce ne sont pas les plus courts, mais en contrepartie on les trouve très vite).

4 Quelques références

- Comme toujours on renvoie à l'excellent livre de Henri Cohen : A Course in Computational Algebraic Number Theory
- Gantmacher : Théorie des matrices
- Pour une implémentation des algorithmes de forme normale de Smith ou de Frobenius, cf. le source de MuPAD ou
  `http://www.mapleapps.com/maplelinks/share/normform.html`
- Ferrard, Lemberg : Mathématiques Concrètes, Illustrées par la TI 92 et la TI 89
  Présente aussi des algorithmes plus numériques, et le lien avec la diagonalisation numérique de matrices.
- Press et al. : Numerical recipies in Fortran/C/Pascal.
  Pour des algorithmes numériques (sur les matrices et autres).

A Bézout et les p-adiques.

Soit n et a/b une fraction irréductible d'entiers tels que b est premier avec n et $|a| < \sqrt{n}/2$ et $0 \leq b \leq \sqrt{n}/2$. Il s'agit de reconstruire a et b connaissant $x = a \times (b^{-1}) \pmod n$ avec $x \in [0, n[$.

Unicité
S'il existe une solution (a, b) vérifiant $|a| < \sqrt{n}/2$ et $0 \leq b \leq \sqrt{n}/2$, soit (a', b') une solution de $x = a \times (b^{-1}) \pmod n$ et vérifiant $|a'| < \sqrt{n}$ et $0 \leq b' \leq \sqrt{n}$, alors :

$$ab' = a'b \pmod n$$

Comme $|ab'| < n/2$, $|a'b| < n/2$, on en déduit que $ab' = a'b$. Donc $a/b = a'/b'$ donc $a = a'$ et $b = b'$ car a/b et a'/b' sont supposées irréductibles.

Reconstruction lorsqu'on sait qu'il y a une solution
On suit l'algorithme de calcul des coefficients de Bézout pour les entiers n et x. On pose :

$$\alpha_k n + \beta_k x = r_k$$

où les r_k sont les restes successifs de l'algorithme d'Euclide, avec la condition initiale :

$$\alpha_0 = 1, \beta_0 = 0, \alpha_1 = 0, \beta_1 = 1, r_0 = n, r_1 = x$$

et la relation de récurrence :

$$\beta_{k+2} = \beta_k - q_{k+2}\beta_{k+1}, \quad q_{k+2} = \frac{r_k - r_{k+2}}{r_{k+1}}$$

On a $\beta_k x = r_k \pmod n$ pour tout rang mais il faut vérifier les conditions de taille sur β_k et r_k pour trouver le couple (a, b). Montrons par récurrence que :

$$\beta_{k+1} r_k - r_{k+1}\beta_k = (-1)^k n \tag{11}$$

Au rang $k = 0$, on vérifie l'égalité, on l'admet au rang k, alors au rang $k+1$, on a :

$$\begin{aligned}
\beta_{k+2} r_{k+1} - r_{k+2}\beta_{k+1} &= \beta_k r_{k+1} - q_{k+2} r_{k+1}\beta_{k+1} - r_{k+2}\beta_{k+1} \\
&= \beta_k r_{k+1} - (r_k - r_{k+2})\beta_{k+1} - r_{k+2}\beta_{k+1} \\
&= \beta_k r_{k+1} - r_k\beta_{k+1} \\
&= -(-1)^k n
\end{aligned}$$

On vérifie aussi que le signe de β_k est positif si k est impair et négatif si k est pair, on déduit donc de (11) :

$$|\beta_{k+1}| r_k < n$$

(avec égalité si $r_{k+1} = 0$)

Considérons la taille des restes successifs, il existe un rang k tel que $r_k \geq \sqrt{n}$ et $r_{k+1} < \sqrt{n}$. On a alors $|\beta_{k+1}| < n/r_k \leq \sqrt{n}$.

Donc l'algorithme de Bézout permet de reconstruire l'unique couple solution s'il existe.

Exemple
On prend $n = 101$, $a = 2$, $b = 3$, $a/b = 68 \pmod{101}$. Puis on effectue Bézout pour 68 et 101 en affichant les étapes intermédiaires (par exemple avec `IEGCD` sur une HP49 ou exercice avec votre système de calcul formel) :

```
   = alpha*101+beta*68
101     1         0
 68     0         1  L1 - 1*L2
 33     1        -1  L2 - 2*L3
  2    -2         3  ...
```

On s'arrête à la première ligne telle que le coefficient de la 1ère colonne est inférieur à $\sqrt{101}$, on retrouve bien 2 et 3. Quand on programme l'algorithme de reconstruction, on ne calcule bien sûr pas la colonne des α, ce qui donne par exemple le programme xcas ou mupad suivant :

```
// Renvoie a/b tel que a/b=x mod n et |a|,|b|<sqrt(n)
padictofrac:=proc (n,x)
  local r0,beta0,r1,beta1,r2,q2,beta2;
begin
  r0:=n;
  beta0:=0;
  r1:=x;
  beta1:=1;
  sqrtn:=float(sqrt(n));
  while r1>sqrtn do
    r2:= irem(r0,r1);
    q2:=(r0-r2)/r1;
    beta2:=beta0-q2*beta1;
    beta0:=beta1; r0:=r1; beta1:=beta2; r1:=r2;
  end_while;
  return(r1/beta1);
end_proc;
```