COURS 7

Algèbre linéaire : de Gauss à Strassen

1. Introduction

Les problèmes algorithmiques en algèbre linéaire cachent des questions très subtiles. En général, les première questions que l'on se pose en rapport avec la manipulation des matrices sont celles du calcul du produit matrice-matrice, éventuellement du produit matrice-vecteur, du calcul de l'inverse, de la résolution de systèmes, etc ...

Les réponses à ces questions vont varier selon le type de matrices que l'on considère. Une classification possible est la suivante.

- Les matrices quelconques, sans structure, pas particulièrement creuses, sont appellées matrices *denses*. Nous verrons que l'algorithmique de ces matrice peut se ramener essentiellement au produit de matrices, dont la complexité est une question extrêmement délicate.
- Un grand nombre de probèmes linéaires se formulent en termes de matrices *creuses*, éventuellement définies sur de petits corps finis. Dans ce cas, l'algorithmique dense est inadaptée ; il est possible d'aller beaucoup plus loin en utilisant des outils mieux adaptés, à base de récurrences linéaires.
- Enfin, on a déjà rencontré (ou entendu parler de) familles de matrices structurées, telles que les matrices de Sylvester pour le résultant (et le PGCD), Vandermonde pour l'évaluation et l'interpolation, etc ... Les questions qui se posent dans ce cadre sont principalement celles du produit matrice-vecteur, ou de la résolution de système. Pour ces types de matrices clairement identifiés, on développe une algorithmique ad-hoc.

Schématiquement, l'algorithmique des matrices denses quelconques est la plus lente (de complexité entre $O(n^2)$ et $O(n^3)$, en taille n) ; la complexité du calcul avec les matrices creuses est de l'ordre de $O(n^2)$, et celle des matrices structurées que l'on a déjà rencontrées est en $O(n)$, à des logs près.

On va maintenant détailler ces résultats ; on ne reparlera désormais que des deux premiers cas, les algorithmes pour matrices structurées ne rentrant pas vraiment dans le cadre du cours présent.

Matrices denses. Dans le cas des matrices quelconques, les questions que l'on sera amené à se poser, ou simplement évoquer, dans ce cours sont celles de la complexité :

- du produit,
- du calcul d'inverse,
- du calcul du polynôme caractéristique ou minimal d'une matrice,
- de la mise sous une forme "canonique" (Smith, Hermite, Frobenius, LUP, ...)

– de la résolution de systèmes linéaires, ...

Comme pour de nombreuses questions déjà rencontrées, on dispose pour effectuer ces opérations d'algorithmes "naïfs" : pour les opérations ci-dessus, leur complexité est cubique en la taille. Le résultat principal de ce cours est que l'on peut faire mieux, et qu'il existe une constante ω (dépendante *a priori* du corps de base), qui contrôle la complexité quasiment toutes les opérations d'algèbre linéaire.

Pour formaliser ce point, on associe à chacun des problèmes ci-dessus son *exposant*. Dans le cas du produit, il est défini comme suit :

$$C_{\text{produit}}(n) = \text{coût minimal d'un algorithme pour le produit en taille } n$$

et

$$\omega_{\text{produit}} = \inf\{\tau \mid C_{\text{produit}}(n) \in O(n^\tau)\}.$$

L'algorithme naïf pour le produit est de complexité $O(n^3)$, donc $\omega_{\text{produit}} \leq 3$. On se doute bien que de l'autre côté, $\omega_{\text{produit}} \geq 2$: il faut au moins n^2 opérations.

Ensuite, on peut définir de la même manière les exposants de tous les autres problèmes, de la forme ω_{inverse}, $\omega_{\text{polynôme caractéristique}}$, $\omega_{\text{déterminant}}$, $\omega_{\text{décomposition LUP}}$, etc ... On a alors le résultat suivant.

THÉORÈME 1. *Les exposants* ω_{produit}, ω_{inverse}, $\omega_{\text{polynôme caractéristique}}$, $\omega_{\text{déterminant}}$, $\omega_{\text{décomposition LUP}}$, *sont tous égaux, et inférieurs à* 2.38 *; on note* ω *leur valeur commune. L'exposant correspondant à la résolution de systèmes linéaires est inférieur ou égal à* ω.

Ce théorème contient deux (familles de) résultats :

– des preuves d'*équivalence* entre problèmes,

– des bornes supérieures sur leur complexité, c'est-à-dire des algorithmes.

Dans le présent chapitre, il est impossible de démontrer ce théorème dans son intégralité. On se contentera de prouver (et encore, sous des hypothèses simplificatrices), que le produit et l'inverse ont le même exposant, et que celui-ci est inférieur à $\log_2 7 < 2.81$.

Matrices creuses. Les questions que l'on se pose en algorithmique creuse sont différentes. Ici, on considère une matrice de taille n, qui ne contient que s entrées non nulles, avec $s \ll n^2$ (qui correspondrait à une matrice pleine). Remarquer qu'alors, on sait effectuer le produit matrice-vecteur en $O(s)$ opérations.

Il n'est pas vraiment naturel de se poser la question du produit ou du calcul d'inverse pour les matrices creuses, le caractère creux n'étant pas vraiment stable par produit ou inverse (autrement dit, les exposants précédemment introduits ne sont pas les bons outils pour étudier les matrices creuses).

La question la plus fréquemment rencontrée dans ce cadre est celle de la *résolution de système*. Nous verrons une version simplifiée d'un algorithme probabiliste dû à Wiedemann, qui repose essentiellement sur des produits matrice-vecteur, et un calcul d'approximants de Padé.

THÉORÈME 2. *Soit* A *une matrice* $n \times n$ *inversible, contenant* s *entrées non nulles. On peut calculer une solution du système linéaire* $Ax = y$ *en* $O(ns)$ *opérations, par un algorithme probabiliste.*

Le cas où la matrice A n'est pas inversible demande davantage de travail, mais on obtient au final le même style de résultat.

Applications. Les questions que l'on a évoquées ici sont *très* fréquemment rencontrées en pratique, que ce soit dans la pratique du calcul formel ou du monde réel.

- De nombreux algorithmes de résolution de systèmes polynomiaux reposent sur de l'algèbre linéaire : les méthodes de calcul de base de Gröbner peuvent se ramener à de l'algèbre linéaire (principalement creuse) en très grande taille (milliers ou millions) ; d'autres approches demandent des produits et inverses de matrices de taille plus modérée (de l'ordre de la dizaine), mais remplies de polynômes de degrés potentiellement élevés (potentiellement des milliers).
- Les algorithmes de factorisation d'entiers les plus récents fonctionnent en deux phases. Tout d'abord on effectue une collecte d'information qui peut être effectuée de manière distribuée, et qui vise à fabriquer une (énorme) matrice creuse et définie sur $\mathbb{F}_2$; ensuite, on cherche un vecteur dans le noyau de cette matrice. Cette application a été un des moteurs de la recherche sur les matrices creuses, dans la communauté du calcul formel.

  D'autres algorithmes de cryptanalyse fonctionnent sur le même schéma ; le logarithmes discret dans les corps finis est un exemple typique.
- Les algorithmes factorisation des polynômes dans un corps fini reposent souvent sur des calculs d'algèbre linéaire : c'est manifeste pour certains d'entre eux (Berlekamp), et plus caché pour d'autres (l'algèbre linéaire intervenant pour accélérer des calculs de types "pas de bébé, pas de géant" ; nous y reviendrons dans un chapitre ultérieur).
- Mentionnons enfin qu'une large partie des ordinateurs du vrai monde passent leurs cycles à faire des calculs d'algèbre linéaire, que ce soit pour faire des calculs d'optimisation combinatoire (programmation linéaire par l'algorithme du simplexe), de la simulation numérique (méthode des éléments finis), ou de la recherche de pages web (Google repose sur de la recherche d'un vecteur propre d'une gigantesque matrice creuse).

2. Matrices denses

L'objectif de cette section est modeste : on va dans un premier temps étudier trois algorithmes de multiplication de matrices denses, en reportant à un cours ultérieur une étude approfondie de cette question. Dans un second temps, on montre comment réduire l'inversion à la multiplication, et réciproquement, de sorte que les deux problèmes ont essentiellement la même complexité.

2.1. Multiplication.

Algorithme naïf. Commençons par décrire l'algorithme "cubique" de multiplication. Étant données deux matrices A et B de taille n, leur produit s'écrit $C = AB$ avec

$$C_{i,k} = \sum_{j=1}^{n} A_{i,j} B_{j,k}.$$

Le coût nécessaire pour obtenir une entrée de C est donc de n multiplications et $n-1$ additions, de sorte que la complexité totale est en $O(n^3)$.

Raffinement : deux fois moins de divisions. Un algorithme dû à Winograd [2] permet essentiellement de diviser par deux le nombre de multiplications. Il repose sur l'identité suivante : pour tous $i, j, j'k$ on a

$$(a_{i,j} + b_{j',k})(a_{i,j'} + b_{j,k}) = a_{i,j}b_{j,k} + a_{i,j'}b_{j',k} + a_{i,j}a_{i,j'} + b_{j,k}b_{j',k}.$$

Ainsi, on récupère deux contributions au terme $C_{i,k}$ au prix d'une seule multiplication, à des termes correctifs près.

L'idée est maintenant d'associer j et j' d'une manière ou d'une autre (par exemple en posant $j' = n-j$; on suppose donc que n est pair), et de faire $i = 1, \ldots, n$ et $k = 1, \ldots, n$. Le point-clé est qu'il y seulement $O(n^2)$ termes correctifs $a_{i,j}a_{i,j'}$ et $b_{j,k}b_{j',k}$, car le premier ne dépend pas de k, et le second ne dépend pas de i. On peut donc tous les calculer pour $O(n^2)$ multiplications. Ensuite, la formule ci-dessus permet de n'effectuer que $n^3/2$ multiplications, au prix de $O(n^3)$ additions supplémentaires.

Au total, le coût de la multiplication de l'algorithme de Winograd est de $\frac{1}{2}n^3 + n^2$ multiplications et $\frac{3}{2}n^3 + 3n^2 - 2n$ additions (quand n est pair). Ainsi, on a essentiellement transformé $n^3/2$ multiplications en additions, ce qui est appréciable (moralement, les multiplications sont toujours plus difficiles que les additions).

Algorithme de Strassen. L'algorithme de Strassen fut le premier algorithme à descendre en-dessous de $O(n^3)$. Tout comme l'algorithme de Karatsuba pour les polynômes, celui-ci repose sur le gain d'une multiplication pour les matrices 2×2, qui devient un gain dans l'exposant quand on l'applique récursivement.

Soient donc A et B des matrices de taille 2. L'algorithme naïf demande d'effectuer 8 multiplications. L'algorithme de Strassen revient à

- calculer des combinaisons linéaires des $a_{i,j}$ et des $b_{i,j}$,
- effectuer **7** produits de ces combinaisons linéaires,
- recombiner les résultats pour obtenir le produit AB.

Explicitement, les formules à appliquer sont les suivantes. On commence par calculer

$$\left|\begin{array}{lcl} q_1 &=& (A_{1,1} - A_{1,2})B_{2,2} \\ q_2 &=& (A_{2,1} - A_{2,2})B_{1,1} \\ q_3 &=& A_{2,2}(B_{1,1} + B_{2,1}) \\ q_4 &=& A_{1,1}(B_{1,2} + B_{2,2}) \\ q_5 &=& (A_{1,1} + A_{2,2})(B_{2,2} - B_{1,1}) \\ q_6 &=& (A_{1,1} + A_{2,1})(B_{1,1} + B_{1,2}) \\ q_7 &=& (A_{1,2} + A_{2,2})(B_{2,1} + B_{2,2}) \end{array}\right.$$

On en déduit les entrées du produit AB de la manière suivante :

$$\left|\begin{array}{lcl} C_{1,1} &=& q_1 - q_3 - q_5 + q_7 \\ C_{1,2} &=& q_4 - q_1 \\ C_{2,1} &=& q_2 + q_3 \\ C_{2,2} &=& -q_2 - q_4 + q_5 + q_6 \end{array}\right.$$

Effectuons maintenant le pas récursif. Comme pour les polynômes, on ne se refuse pas le confort de supposer que les matrices ont une taille qui est une puissance de 2 ; l'analyse s'étendrait facilement dans le cas général.

Soient donc A et B des matrice de taille $2n$, n étant lui-même une puissance de 2. On procède à une découpe en blocs de A et B :

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \quad B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}.$$

Dans cette écriture, les $A_{i,j}$ et $B_{i,j}$ sont donc des matrices $n \times n$.

Le point-clé est le suivant : les formules données ci-dessus pour le cas 2×2 s'appliquent toujours si les $A_{i,j}$ et $B_{i,j}$ sont des matrices. Ainsi, elles permettent de ramener le produit en taille $2n$ à 7 produits en taille n, plus un certain nombre d'additions, disons Cn^2, où C est une constante (pour fabriquer les petites combinaisons linéaires avant de faire les produits, et récupérer ensuite les entrées de AB).

En en déduit le résultat suivant :

THÉORÈME 3. *Le produit AB peut se calculer en $O(n^{\log_2(7)}) \simeq O(n^{2.81})$ opérations de k.*

DÉMONSTRATION. Le coût $T(n)$ satisfait la récurrence

$$T(2n) \leq 7T(n) + Cn^2.$$

La conclusion s'en déduit facilement ; la démonstration est la même que celle de la complexité de Karatsuba. □

Il est difficile de donner une intuition pour motiver les formules de Strassen (à la différence de Karatsuba, qui repose de manière cachée sur l'évaluation en des valeurs entières). Voici ce que Strassen en dit :

First I had realized that an estimate tensor rank < 8 for two by two matrix multiplication would give an asymptotically faster algorithm. Then I worked over $\mathbb{Z}/2\mathbb{Z}$ (as far as I remember) to simplify matters.

Autrement dit, il y a dû y avoir recherche "à la main". Quant à la mention du tensor rank, elle est du ressort du prochain cours.

L'algorithme de Winograd mentionné plus haut effectue lui aussi le produit de matrices 2×2 en 7 opérations. Par contre, on ne peut pas l'utiliser de manière récursive comme celui de Strassen : la formule qui sous-tend Winograd repose sur le fait que $ab = ba$, si a et b sont des scalaires (des éléments du corps de base). Lors des appels récursifs, les objets à multiplier sont eux-mêmes des matrices, pour lesquels la loi de commutation n'est plus vérifiée. On dit donc que l'algorithme de Winograd est un algorithme *non-commutatif*.

Pour faire mieux que 2.81 *?* Strassen ayant prouvé que le produit de matrices était sous-cubique, la question naturelle est devenue de savoir si on pouvait descendre jusqu'à du $O(n^2)$. Des progrès ont été faits tout au long des années 1970 et 1980, les meilleurs algorithmes actuels étant en $O(n^{2.38})$.

Pour mesurer la qualité de ces algorithmes, on introduit naturellement la notion d'*exposant de la multiplication*, c'est-à-dire de la borne inférieure ω_{produit} de l'ensemble des réels τ tels que le produit de deux matrices de taille n peut se faire en $O(n^\tau)$ opérations.

On a évidemment $\omega_{\text{produit}} \geq 2$; l'algorithme de Strassen montre que $\omega_{\text{produit}} \leq \log_2(7)$, et les meilleures bornes supérieures actuelles sont $\omega_{\text{produit}} \leq 2.38$. On n'a pas de borne inférieure non-triviale pour ω_{produit} (c'est-à-dire autre que 2).

L'étude des algorithmes et des idées qui permettent de faire mieux que $\log_2(7)$ sera effectuée au prochain cours, si tout va bien.

En pratique. Les algorithmes rapides pour l'algèbre linéaire ont longtemps eu mauvaise presse ; concrètement, l'algorithme de Strassen, et, de manière plus marginale, un algorithme dû à Pan et repris par Kaporin [1], d'exposant 2.77, sont les seuls algorithmes "rapides" (avec un exposant inférieur à 3) qui ont été implantés avec succès.

Dans une bonne implantation, disons sur un corps fini "raisonnable", l'algorithme de Winograd est immédiatement rentable. L'algorithme de Strassen devient ensuite meilleur pour des tailles de l'ordre de quelques dizaines (64 est une borne raisonnable).

L'immense majorié des autres algorithmes connus reposent sur des techniques très complexes, qui se traduisent par la présence d'énormes constantes (et de facteurs logarithmiques) dans leurs estimation de complexité. En conséquence, dans leur versions actuelles, il ne peuvent pas être rentables pour des tailles inférieures à des millions ou des milliards ...

2.2. Autres opérations. Ce n'est pas tout de savoir multiplier les matrices ; il est tout aussi légitime de vouloir les inverser, calculer leur polynôme caractéristique, etc ...

L'objectif de cette sous-section est de montrer (ou de suggérer) que *toutes ces opérations sont équivalentes à la multiplication* en termes de complexité, et donc équivalentes entre elles. Ainsi, l'exposant ω_{produit} devient exposant pour quasiment tous les problèmes "classiques" d'algèbre linéaire. Le seul problème qui résiste est la résolution de systèmes : on sait qu'il n'est pas plus dur que la multiplication, mais peut-être est il plus facile ...

Le but de ce paragraphe est de donner quelques exemples de réductions entre ces problèmes : on montre (presque) l'équivalence entre l'inversion est la multiplication.

La multiplication n'est pas plus difficile que l'inversion. Cette réduction repose sur une formule ingénieuse. Soient A et B des matrices $n \times n$. On souhaite calculer $C = AB$. Pour cela, on pose

$$D = \begin{bmatrix} I_n & A & 0 \\ 0 & I_n & B \\ 0 & 0 & I_n \end{bmatrix}$$

Alors

$$D^{-1} = \begin{bmatrix} I_n & -A & AB \\ 0 & I_n & -B \\ 0 & 0 & I_n \end{bmatrix}$$

On ramène donc le produit en taille n à l'inversion en taille $3n$, ce qui prouve notre affirmation.

L'inversion n'est pas plus difficile que la multiplication. Soit A la matrice à inverser. On prśente ici un algorithme simple d'inversion ; celui-ci nécessite d'inverser des sous-matrices de A, des produits de telles sous-matrices, ... On suppose que *toutes ces matrices sont inversibles.* Le cas général est plus délicat, il passe par le calcul d'une décomposition LUP de A, ce qui est assez technique.

L'algorithme est récursif. Supposons que A est de taille $2n \times 2n$, et écrivons la sous une forme bloc

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}$$

En posant $S = A_{2,2} - A_{2,1}A_{1,1}^{-1}A_{1,2}$, on se convainc que A peut se réécrire

$$A = \begin{bmatrix} I_n & 0 \\ A_{2,1}A_{1,1}^{-1} & I_n \end{bmatrix} \begin{bmatrix} A_{1,1} & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} I_n & A_{1,1}^{-1} \\ 0 & I_n \end{bmatrix}.$$

On en déduit que l'inverse de A s'écrit

$$A = \begin{bmatrix} I_n & -A_{1,1}^{-1} \\ 0 & I_n \end{bmatrix} \begin{bmatrix} A_{1,1}^{-1} & 0 \\ 0 & S^{-1} \end{bmatrix} \begin{bmatrix} I_n & 0 \\ -A_{2,1}A_{1,1}^{-1} & I_n \end{bmatrix}.$$

On est donc amenés à effectuer deux inversions, ainsi qu'un certain nombre de produits, ce qui mène au résultat suivant.

THÉORÈME 4. *Soit ω_{produit} un exposant pour la multiplication de matrices. Si toutes les sous-matrices rencontrées au cours de l'algorithme sont inversibles, le coût de l'inversion de A est $O(n^{\omega_{\text{produit}}})$.*

DÉMONSTRATION. L'algorithme ci-dessus montre que la complexité $I(n)$ de l'inversion satisfait la récurrence

$$I(2n) \leq 2I(n) + Cn^{\omega_{\text{produit}}},$$

C étant une constante. On en déduit le résultat (attention, il faut utiliser le fait que $\omega_{\text{produit}} \geq 2$ pour ne pas perdre bêtement un facteur log dans la résolution de la récurrence). □

3. Matrices creuses

Dans cette section, on aborde des questions radicalement différentes : on ne cherche plus à effectuer les produit, inversion, ... de matrices quelconques, mais à manipuler des matrices *creuses.*

Attention, on ne va pas définir de ce qu'est une matrice creuse. Ce que sous-entend ce terme, c'est qu'on va donner un algorithme dont la complexité va s'exprimer en fonction du nombre d'entrées non nulles de la matrice.

Dans tout ce qui suit, on considère donc une matrice A de taille $n \times n$, et on fait les hypothèses suivantes :

- A est inversible,
- A contient s entrées non nulles.

On va montrer un algorithme dû à Wiedemann qui permet de calculer l'unique solution du système $Ax = y$, avec une complexité en $O(ns)$; l'algorithme original est plus général que ce que l'on va présenter, puiqu'il permet de traiter les matrices rectangulaires, ou carrées mais non inversibles.

Remarquons que si s est de l'ordre de n^2 (matrice plutôt pleine, donc), on retombe dans une complexité de l'ordre de $O(n^3)$. Le cas intéressant est celui où s

est de l'ordre de n, auquel cas l'algorithme est quadratique en n : il faut avoir en tête que l'exposant de l'algèbre linéaire creuse est 2, dans les bons cas.

3.1. Polynôme minimal et résolution de sysèmes. L'algorithme de Wiedemann passe par le calcul du *polynôme minimal* de A. Rappelons sa définition.

Si k est le corps de base, il est possible d'associer à tout polynôme en une variable $P \in k[T]$ la matrice $P(A)$. On sait que d'après le théorème de Cayley-Hamilton, $\chi(A) = 0$, où χ est le polynôme caractéristique de A. Le polynôme minimal de A est le polynôme unitaire de plus petit degré ayant cette propriété (on montre facilement que cette propriété le rend unique).

Soit P le polynôme minimal de A, supposé connu. Puisqu'on a supposé A inversible, le terme constant de P n'est pas nul (car il divise le terme constant du polynôme caractéristique, qui n'est lui-même pas nul).

L'égalité $P(A) = 0$ se réécrit sous la forme

$$A^{-1} = \frac{-1}{p_0}\left(A^{m-1} + p_{m-1}A^{m-2} + \cdots + p_1 I_n\right)$$

Pour résoudre le système $Ax = y$, on va calculer $x = A^{-1}y$, c'est-à-dire

$$\frac{-1}{p_0}\left(A^{m-1}y + p_{m-1}A^{m-2}y + \cdots + p_1 y\right).$$

Ainsi, il suffit de calculer les itérés $y, Ay, \ldots, A^{m-1}y$, puis d'additionner les termes $p_1 y, p_2 Ay, \ldots, A^{m-1}y$, pour retrouver x.

- Chacun des $A^i y$ s'obtient à partir de $A^{i-1}y$ en $O(s)$ opérations.
- Chacun des produits par p_i, et chaque addition, se fait en $O(n)$ opérations. Remarquer que $n \leq s$ (hypothèse d'inversibilité de A).

Au total, on a donc $O(ns)$ opérations à faire pour résoudre le système si on connaît le polynôme minimal de A.

3.2. Calcul du polynôme minimal. Écrivons le polynôme minimal de A sous la forme

$$P = T^m + p_{m-1}T^{m-1} + \cdots + p_0.$$

Alors la suite des puissances A satisfait la récurrence linéaire

$$A^{k+m} + p_{m-1}A^{k+m-1} + \cdots + p_0 A^k = 0,$$

et ne satisfait pas de récurrence d'ordre plus petit.

Pour tous vecteurs u et v, la suite (de nombres) $N_i := u^t A^i v$ vérifie donc

$$N_{k+m} + p_{m-1}N_{k+m-1} + \cdots + p_0 N_k = 0.$$

Si u et v sont mal choisis (nuls, par exemple), la suite N_i satisfait une récurrence d'ordre plus petit que m. La proposition suivante montre qu'en choisissant u et v au hasard, on tombe vraisemblablement sur une suite N_i qui ne satisfait pas de récurrence d'ordre plus petit.

PROPOSITION 1. *Il existe un polynôme non nul D dans $k[U_1, \ldots, U_n, V_1, \ldots, V_n]q$, de degré au plus $2n$, tel que si $D(u, v)$ est non nul, la suite N_i associée à u et v ne satisfait pas de récurrence d'ordre plus petit que m.*

L'idée est de choisir u et v au hasard. Si on n'a pas de chance, $D(u, v)$ est nul ; sinon, la suite des N_i associée à u et v permet de retrouver P par un calcul d'approximants de Padé (attention, on ne connaît pas explicitement le polynôme D, mais son existence assure que la plupart des choix de u et v sont "chanceux").

Algorithme de Wiedemann.:

Entrée :: la matrice A.

Sortie :: son polynôme minimal.

1. Choisir des vecteurs u et v aléatoires.
2. Pour $0 \leq i \leq 2n$, calculer les vecteurs $A^i v$, puis les scalaires $N_i = u^t A^i v$.
3. Retrouver la récurrence satisfaite par les N_i.

La complexité de l'étape 2 est $O(n)$ produits matrice-vecteur ou vecteur-vecteur, chacun prenant au pire $O(s)$ opérations ; au total on obtient donc $O(ns)$ opérations pour cette phase. L'algorithme d'approximants de Padé est négligeable, puisqu'il ne demande que $O(\mathsf{M}(n)\log(n)) \leq sn$ opérations.

Bibliographie

[1] Kaporin (I). – The aggregation and cancellation techniques as a practical tool for faster matrix multiplication. *Theoretical Computer Science*, vol. 315, n° 2-3, 2004, pp. 469–510.

[2] Winograd (S.). – A new algorithm for inner-product. *IEEE Transactions on Computers*, vol. 17, 1968, pp. 693–694.