Résolution de systèmes linéaires creux : l'algorithme de Wiedemann

Préparation à l'agrégation - option Calcul formel

Antoine Chambert-Loir

1. Introduction

L'algorithme du pivot de Gauss, ou celui de Gauss-Jordan, sont des méthodes bien connues permettant de résoudre des systèmes d'équations linéaires. Pour peu qu'on les utilise dans des contextes où les calculs arithmétiques sont exacts, ils fournissent en outre la solution exacte. Cependant, lorsque le système à résoudre n'a que peu de coefficients non nuls (on parle de *système creux*), ces algorithmes n'en tirent pas partie et, au contraire, « remplissent » petit à petit la matrice du système, ruinant ainsi l'éventuel gain de complexité, en temps et en espace mémoire/disque, que l'on aurait pu espérer.

Par exemple, dans sa phase finale, la factorisation d'un entier de 130 chiffres (challenge RSA-130) requit la résolution d'un système ayant 3 516 502 lignes et 3 504 823 colonnes, dont, en moyene, seulement un quarantaine de coefficients par lignes n'étaient pas nuls.

L'*algorithme de Wiedemann* dont il est question dans ce texte permet de résoudre un système $Ax = b$ en ne modifiant pas la matrice A et en ne manipulant que des vecteurs obtenus par application itérée de A. Ils sont similaires à l'algorithme de Lanczos ou à celui du gradient conjugué en analyse numérique mais fournissent une solution exacte dans des contextes tels que les corps finis.

2. La suite de Krylov

Soit un système d'équations linéaires $Ax = b$ à coefficients dans un corps K, A étant une matrice $n \times n$ que l'on suppose inversible. La *suite de Krylov* est la suite des vecteurs de l'espace vectoriel $V = K^n$,

$$b, Ab, A^2 b, A^3 b, \dots$$

Il existe une relation de dépendance linéaire entre les $n+1$ premiers vecteurs ; de plus, si

$$c_0 b + c_1 Ab + \cdots + c_n A^n b = 0$$

est une relation non triviale, avec $c_0 \neq 0$, alors

$$x = -\frac{c_1}{c_0} b - \cdots - \frac{c_n}{c_0} A^{n-1} b$$

est solution de $Ax = b$.

Récrivant la relation de dépendance sous la forme $p(A)b = 0$, avec $p = c_0 + c_1 T + \ldots + c_n T^n$, il convient d'observer le lemme suivant.

Lemme 2.1. — *L'ensemble des polynômes $p \in K[T]$ tels que $P(A)b = 0$ est un idéal de $K[T]$, engendré par un polynôme unitaire f de degré $\leqslant n$.*

Il s'agit de déterminer efficacement ce polynôme f. En effet, la solution de $Ax = b$ est alors donnée par $x = f^*(A)b$, où $f^*(T) = \frac{f(0)-f(T)}{f(0)T}$.

Remarquons que si $f = c_0 + c_1 T + \cdots + c_d T^d$, alors pour tout entier $j \geqslant 0$,

$$c_0 A^j b + c_1 A^{j+1} b + \cdots + c_d A^{j+d} b = A^j p(A) b = 0,$$

autrement dit la suite de vecteurs $(A^j b)$ est solution d'une équation de récurrence linéaire de degré $d \leqslant n$. Il en est de même des suites $(\varphi(A^j b))$, pour tout forme linéaire φ sur l'espace vectoriel V.

3. L'algorithme de Berlekamp–Massey

Soit $(s_0, s_1, \ldots, s_{2n-1})$ une suite d'éléments de K. L'algorithme de Berlekamp–Massey a précisément pour fonction de détecter une relation de récurrence linéaire entre les s_i, c'est-à-dire des éléments $c_0, \ldots, c_n$ de K tels que

$$c_0 s_k + c_1 s_{k+1} + \cdots + c_n s_{k+n} = 0 \qquad \text{pour tout } k < n.$$

Notons $\tilde{S} \in K[T]$ le polynôme défini par $\tilde{S}(T) = s_{2n-1} + s_{2n-2} T + \cdots + s_0 T^{2n-1}$; les relations précédentes signifient que, notant $c(T) = c_0 + c_1 T + \cdots + c_n T^n$, les coefficients du polynôme $c(T)\tilde{S}(T)$ de degrés n à $2n-1$ sont tous nuls. Autrement dit, il existe des polynômes $a(T)$ et $b(T)$ tels que

$$b(T) = a(T) T^{2n} + c(T)\tilde{S}(T), \qquad \deg(a), \deg(b) < n.$$

Rappelons aussi que cette équation intervient en théorie des codes correcteurs d'erreurs, notamment pour le décodage des codes BCH et de Goppa.

3.1. Via l'algorithme d'Euclide. — Appliquons l'algorithme d'Euclide étendu aux polynômes T^{2n} et $\tilde{S}(T)$; on obtient des suites de polynômes $u_i(T)$, $v_i(T)$, $r_i(T)$, tels que $r_0(T) = T^{2n}$, $r_1(T) = \tilde{S}(T)$, et, pour tout i, $r_i(T) = u_i(T) T^{2n} + v_i(T)\tilde{S}(T)$. En s'arrêtant au premier indice i tel que $\deg(r_i) < n$, on a aussi $\deg(u_i) < n$ et $\deg(v_i) < n$, d'où la relation voulue en posant $c = v_i$. (La démonstration que cet algorithme fournit le résultat escompté a été faite dans le texte sur les codes de Goppa.)

Du point de vue de l'algorithme initial, il n'est pas nécessaire de calculer $u_0, \ldots, u_i$, seul importe celui de $r_0, r_1, \ldots$ et de $v_0, v_1, \ldots$.

3.2. Une autre méthode, plus efficace. — Cette méthode ne fournit que le polynôme a (mais b est facile à retrouver après) et semble sensiblement plus rapide que la précédente. Elle consiste en l'algorithme suivant

Initialisation : $m = 1$, $r = 0$, $a = 1$, $c = T$.
Tant que $m \leqslant 2n$:
  Soit e le coefficient de T^{m-1} dans la série aS.
  Si $e \neq 0$:
    On pose $a^* = a - ec$;
    Si $2r < m$, on pose $c = a/e$ et $r = m - r$;
    On remplace a par a^* ;
  Fin si.
  On remplace c par Tc et n par $n+1$.

4. Un peu de hasard

L'algorithme de Berlekamp–Massey ne concerne que les suites de nombres, pas celles de vecteurs. Pour l'appliquer à notre situation, à savoir détecter une relation de récurrence linéaire entre une suite de vecteurs, nous allons appliquer aux termes de cette suite une application linéaire.

Rappelons les notations : A est une matrice inversible $n \times n$ à coefficients dans un corps K, b est un vecteur de $V = K^n$ et l'on souhaite calculer le polynôme unitaire f, générateur de l'idéal des polynômes p tels que $p(A)b = 0$.

Soit φ une forme linéaire sur V. L'ensemble des relations de récurrence linéaires satisfaites par la suite $(\varphi(A^n b))$ s'identifie à l'ensemble des polynômes $p \in K[T]$ tels que $\varphi(A^k p(A)b) = 0$ pour tout entier $k \geqslant 0$. Cet ensemble est encore un idéal de $K[T]$, donc est engendré par un polynôme unitaire f_φ. Il est clair que f_φ divise f.

Désignons par F l'ensemble (fini) des diviseurs unitaires stricts de f ; pour $g \in F$, le vecteur $g(A)b$ n'est pas nul et les formes linéaires φ telles que $\varphi(g(A)b) = 0$ forment un hyperplan H_g de V^*. Les mauvaises formes, c'est-à-dire celles pour lesquelles $f_\varphi \neq f$, sont donc celles qui appartiennent à l'un des H_g.

Si le cardinal du corps K est infini, ou s'il est assez grand, une forme linéaire φ choisie au hasard ne devrait pas appartenir à H_g pour aucun g. Cela fournit une première solution pour déterminer f, à savoir tirer φ au hasard, calculer f_φ par l'algorithme de Berlekamp–Massey et recommencer si $f_\varphi(A)b \neq 0$.

Exercice 4.1. — a) Estimer la probabilité que φ soit mauvaise en fonction du degré de f (ou de n) et du cardinal du corps K.

b) Soit S une partie de K ; on choisit des formes linéaires φ au hasard, uniformément parmi l'ensemble S^n des vecteurs lignes à coefficients dans S. La probabilité qu'une telle forme soit mauvaise est au plus $n/\operatorname{Card}(S)$.

Plutôt que de recommencer le calcul à zéro, on peut tirer parti du fait que f est le ppcm des f_φ, lorsque φ parcourt une famille assez grande de formes linéaires, telles qu'une base de V^*. Si $(\varphi_1, \varphi_2, \ldots)$ est une suite de formes linéaires, on aura intérêt à poser $f_1 = f_{\varphi_1}$, puis $f_2 = \operatorname{ppcm}(f_1, f_{\varphi_2})$, etc., et à continuer tant que $f_i(A)b \neq 0$. On a deux options : soit prendre a priori une base de V^*, soit tirer chaque forme au hasard.

Une dernière méthode tire peut-être mieux profit des calculs antérieurs et évite les calculs de $ppcm$. On pose $b_1 = b$, on choisit une forme linéaire φ_1 (disons au hasard) et on détermine le polynôme $f_1 = f_{\varphi_1}$ tel que $\varphi_1(f_1(A)b_1) = 0$ par l'algorithme de Berlekamp–Massey. On pose alors $b_2 = f_1(A)b_1$. Si $b_2 = 0$, on a fini. Sinon, on recommence avec une forme linéaire φ_2 : on détermine le polynôme f_2 tel que $\varphi_2(f_2(A)b_2) = 0$, on pose $b_3 = f_2(A)b_2 = (f_1f_2)(A)b_1$, et ainsi de suite. À un certain moment, $b_{k+1} = 0$ et $f = f_1 \cdots f_k$.

4.2. Complexité. — La complexité de cet algorithme est délicate à analyser en détail, notamment par son aspect probabiliste. Disons-en tout de même quelque mots.

Du point de vue de l'espace mémoire qu'il nécessite tout d'abord, et mis à part le stockage de la matrice A, on n'a besoin que de $O(n)$ éléments de K à la fois. En effet, dans la partie principale de l'algorithme, le vecteur $A^j b$ peut être oublié après avoir calculé $A^{j+1}b$ et $\varphi(A^j b)$. De même, l'algorithme de Berlekamp–Massey ne demande que $O(n)$ éléments de K.

Du point de vue du nombre d'opérations, l'algorithme de Berlekamp–Massey requiert $O(n^2)$ opérations dans K. Le calcul des vecteurs $Ab, \dots, A^n b$ demande *a priori* $O(n^3)$ opérations, mais si la matrice A est creuse et stockée en conséquence (par exemple, par l'ensemble des triplets (i, j, a_{ij}) tels que $a_{ij} \neq 0$) ce nombre peut être significativement réduit.

5. Quelques suggestions

Outre la démonstration de quelques points mathématiques du texte (mais en évitant la justification du second algorithme de Berlekamp–Massey), vous pourriez expérimenter l'algorithme de Wiedemann et comparer les différentes stratégies pour obtenir le polynôme f.