Résolution de systèmes linéaires

Dorian Lesbre

Encadrant: Jérémy Berthomieu

5 juin 2019

Table des matières

1 Introduction **2**

2 Calcul itératif de série tronquée **2**
2.1 Méthode de Newton pour l'inversion matricielle 2
2.2 Résolution de systèmes linéaires 4
2.3 Prise en compte de la taille des éléments 5

3 Résolution particulière dans l'anneau des entiers **6**
3.1 Résolution modulo p dans $\mathbb{Z}$ 6
3.2 Décodage vers la forme rationnelle dans $\mathbb{Z}$ 7

4 Cas des polynômes, relèvement P-adique **10**
4.1 Définition et propriétés de l'expansion P-adique 10
4.2 Relèvement P-adique 11
4.3 Solution en série 13

5 Conclusion **14**

Références

[1] John D. Dixon. Exact solutions of linear equations using p-adic expansions. *Numer. Math*, (40.1) :137–141, février 1982. ISSN : 0029-599X, DOI : 10.1007/BF01459082, URL : `http://dx.doi.org/10.1007/BF01459082`.

[2] Robert T. Moenck et John H. Carter. *Symbolic and Algebraic Computation*, chapter Approximate algorithms to derive exact solutions to systems of linear equations, pages 65–73. Springer Berlin Heidelberg, 1979. ISBN : 978-3-540-35128-3.

[3] Arne Storjohann. High-order lifting. *Proceedings of the 2002 international Symposium on Symbolic and Algebraic computation*, (ISSAC 2002 Lille, France) :246–254, ACM, 2002. ISBN : 1-58113-484-3, DOI : 10.1145/780506.780538, URL : `http://doi.amc.org/10.1145/780506.780538`.

1 Introduction

Dans le cadre de l'algèbre effective, un des problème est de faire des calculs d'algèbre par informatique de manière efficace. Nous nous intéressons à des algorithmes de résolution de systèmes linéaires en calculant à partir d'une expansion adique. Pour cela, nous nous plaçons dans un anneau intègre et commutatif $(\mathcal{A}, +, \times)$, et notons notre système linéaire de taille $n \in \mathbb{N}^*$ sous forme matricielle :

$$Ax = b$$

Avec $A \in \mathcal{M}_{n\times n}(\mathcal{A})$ une matrice inversible de taille $n \times n$, b et $x \in \mathcal{A}^n$ des vecteurs de taille n.

Ce problème est très classique est a été beaucoup étudié sous de nombreux aspects. Il existe ainsi plusieurs méthodes pour résoudre ce genre de systèmes. Par exemple il est possible de procéder à un pivot de Gauss, que ce soit par approximation flottante ou en calculant sur des valeurs de tailles arbitraire tel des entiers multiples. L'intérêt d'utiliser un développement partiel en série est alors de garder l'exactitude du calcul, tout en limitant la tailles des valeurs manipulées. En effet, le pivot de Gauss par exemple, peut conduire à des calculs sur des valeurs dont la taille croit exponentiellement par rapport à celle du système initial, ce qui n'est pas désirable.

Notre objectif est donc de présenter des algorithmes pour résoudre ce système modulo un idéal, en essayant de minimiser le temps de calcul, tout en prenant en compte la taille des valeurs manipulées. Nous présenterons dans un premier temps des méthodes de résolution dans le cas général d'un anneau $\mathcal{A}$. La méthode obtenu ainsi donne une résolution modulo un idéal jusqu'à l'ordre k en $O(kn^2)$ opérations, en limitant la taille des valeurs uniquement lors de produit.

Pour affiner cette méthode, nous nous plaçons ensuite dans les cas particuliers de l'anneau des entiers et de celui des polynômes. Cela permet de mieux contrôler les contraintes de taille (nombre de bits ou degré) de valeurs, ainsi que la complexité des opérations d'anneau sur ces valeurs.

2 Calcul itératif de série tronquée

Nous travaillons ici dans un anneau intègre et commutatif $(\mathcal{A}, +, \times)$. Une première méthode de résolution consiste à chercher une solution proche sous forme de série entière tronquée modulo un idéal. Soit $p \in \mathcal{A}$, on notera $\langle p \rangle = \{ap, a \in \mathcal{A}\}$ l'idéal engendré par le singleton $\{p\}$.

Le calcul de la solution rationnelle exacte à partir de la série peut se faire pourvu que l'ordre est été poussé assez loin. En effet, le dénominateur est $\det A$, il suffit donc de multiplier la série par l'expansion de $\det A$ modulo $\langle p \rangle$ avec suffisamment de précision pour obtenir le numérateur.

2.1 Méthode de Newton pour l'inversion matricielle

Dans cette section, nous étudions comment obtenir des approximations successives de plus en plus précises de l'inverse de $A \in \mathcal{M}_{n\times n}(\mathcal{A})$ modulo des puissances de p à partir d'une première approximation. Plus précisément, nous allons montrer qu'il est possible de calculer une chaine de k matrices $(B_1, B_2, B_4, \ldots, B_{2^k})$ vérifiant :

$$\forall i \in [\![1,\ k]\!],\ A \times B_{2^i} \equiv I \mod \langle p^{2^i} \rangle$$

à partir d'une première approximation B_1 inverse de A modulo $\langle p \rangle$. Cela se calcule au moyen de l'algorithme suivant.

Algorithme 1 – Inversion matricielle modulo $\langle p^{2^k} \rangle$ par la méthode de Newton

Données : les matrices A et B=B_1, l'élément p, le nombre d'itérations k
Résultat : la matrice B_{2^k}
Algorithme :

```
Pour i = 1 à k − 1 faire
    E ← I − A × B mod p^(2^i)
    B ← B × (I + E)
renvoyer B
```

Proposition 1 : (Correction et complexité)
L'algorithme ci-dessus est correct, il calcule effectivement B_{2^k} tel que

$$A \times B_{2^i} \equiv I \mod \langle p^{2^i} \rangle$$

L'algorithme a une complexité en $O(kn^3)$ opérations d'anneau, l'exposant pouvant être amélioré selon la méthode de multiplication matricielle utilisée, et effectue $O(kn^2)$ calculs de modulo dans l'anneau.

Démonstration : Montrons par récurrence sur $i \in [\![1,\ k]\!]$ le propriété suivante :

$$H(i) = \text{"Au début de l'itération } i,\ B = B_{2^i}\text{"}$$

$H(1)$ est vraie par hypothèse sur les données. Soit $i \in [\![2,\ k]\!]$, supposons $H(i-1)$ vraie. L'algorithme calcule $E \equiv AB_{2^{i-1}} - I \mod \langle p^{2^i} \rangle$, comme $AB_{2^{i-1}} \equiv I \mod \langle p^{2^{i-1}} \rangle$, nous avons

$$E = p^{2^{i-1}} \tilde{E}$$

où $\tilde{E} \in \mathcal{M}_{n\times n}(\mathcal{A})$. Cela donne donc :

$$\begin{aligned} A \times B(I+E) &= A \times B_{2^{i-1}}(I + p^{2^{i-1}}\tilde{E}) \\ &\equiv (I - p^{2^{i-1}}\tilde{E})(I + p^{2^{i-1}}\tilde{E})) \mod \langle p^{2^i} \rangle \\ &\equiv I - \left(p^{2^{i-1}}\tilde{E}\right)^2 \mod \langle p^{2^i} \rangle \\ &\equiv I - p^{2^i}\tilde{E}^2 \mod \langle p^{2^i} \rangle \\ &\equiv I \mod \langle p^{2^i} \rangle \end{aligned}$$

Comme il y à $k-1$ itérations, la valeur renvoyée correspond à celle au début de la k-ième itération, d'où la correction. □

Exemple d'utilisation : cet algorithme peut notamment servir dans le cas où $\mathcal{A}$ est l'anneau des polynômes $\mathbb{K}[x]$ sur un corps $\mathbb{K}$. Les composants des B_{2^i} sont alors les les termes d'une série entière tronquée, que l'on peut utiliser pour retrouver la fraction rationnelle $A^{-1} \in \mathcal{M}_{n\times n}(\mathbb{K}(x))$, par produit avec le déterminant de A.

2.2 Résolution de systèmes linéaires

Nous proposons maintenant un algorithme de résolution de $Ax = b$ modulo un idéal $\langle p \rangle$, en supposant A inversible et modulo p. Il calcule (x_k) vérifiant $Ax_k \equiv b \mod \langle p^k \rangle$.

Algorithme 2 – Résolution de système linéaires

Données : $A \in \mathcal{M}_{n\times n}(\mathcal{A}), b \in \mathcal{A}^n, p \in \mathcal{A}$, et $k \in \mathbb{N}$ le nombre d'itérations
Résultat : x_k vérifiant $Ax_k \equiv b \mod \langle p^k \rangle$
Algorithme :

```
A' ← A mod p
b' ← b mod p
x  ← A'^-1 × b mod p
pM ← A' − A
Pour i allant de 1 à k − 1 faire
    x ← A'^-1 × pM × x + A'^-1 × b
renvoyer x
```

Remarque : Cette méthode de résolution repose sur une transformation de notre système linéaire $Ax = b$, en décomposant

$$A = A' - pM$$

pM étant bien défini car $A \equiv A' \mod \langle p \rangle$ donc $A - A'$ est un multiple de p. Le système devient alors $A'x = pMx + b$.

À partir d'un vecteur x_0 nous pouvons définir récursivement une suite (x_k) par la relation $A'x_k = pMx_{k-1} + b$, c'est ce que calcule l'algorithme dans la boucle.

Proposition 2 : (Propriété de la suite (x_k))
La suite $(x_k)_{k\geqslant 1}$ ainsi définie vérifie

$$\forall k \in \mathbb{N}^*, Ax_k = b \mod \langle p^k \rangle$$

Cela implique notamment la correction de l'algorithme.

Démonstration : Notons x la solution du système $Ax = b$, il vérifie $A'x = pMx + b$. En y soustrayant la définition de la suite (x_k) nous obtenons pour $k \geqslant 1$:

$$A'(x - x_{k+1}) = pM(x - x_k)$$

Il s'agit d'une suite géométrique de raison $A'^{-1}pM$, donc $x - x_{k+1} = (A'^{-1}pM)^k(x - x_1)$. Or $(A'^{-1}pM)^k = p^k(A'^{-1}M)^k \equiv 0 \mod \langle p^k \rangle$ car l'anneau est commutatif. De plus $x - x_1 \equiv 0 \mod \langle p \rangle$ car $A'x_1 \equiv Ax_1 \equiv b \mod \langle p \rangle$. Par produit, nous avons donc

$$x - x_{k+1} \equiv 0 \mod \langle p^{k+1} \rangle$$

x_{k+1} est donc bien solution modulo $\langle p^{k+1} \rangle$. □

Applications : dans le cas où $\mathcal{A} = \mathbb{Z}$, cet algorithme permet de développer des solutions en arithmétique p-adique, ou directement dans $\mathbb{Q}$ en multipliant par le déterminant de A. Cette méthode fonctionne également pour l'anneau des polynômes $\mathbb{K}[x]$,

Complexité : cet algorithme peut-être optimisé en ne calculant qu'une seule fois A'^{-1} (l'inverse modulo $\langle p \rangle$ suffit d'ailleurs), ainsi que les produits $A'^{-1} \times pM$ et $A'^{-1}b$. Sa complexité est alors en $O(kn^2 + n^3)$ opérations d'anneau.

2.3 Prise en compte de la taille des éléments

Le calcul effectué dans l'algorithme de la section précédente n'impose pas de bornes sur la taille des valeurs dans $\mathcal{A}$ manipulées, et peut donc effectuer des produits sur valeurs non bornées (par exemple des polynômes à haut degré ou des grands entiers), ce qui limite son efficacité. Nous développons donc une autre version cherchant à faire de calculs de taille bornée, au moins pour les produits.

Algorithme 3 – Version optimisée de la résolution de système

Données : $A \in \mathcal{M}_{n\times n}(\mathcal{A}), b \in \mathcal{A}^n, p \in \mathcal{A}$, et $k \in \mathbb{N}$ le nombre d'itérations
Résultat : x_k vérifiant $Ax_k \equiv b \mod \langle p^k \rangle$
Algorithme :

```
C ← inverse(A mod p, p)  // inverse modulo ⟨p⟩
x ← C × b
Pour i = 1 à k − 1 faire
    w ← b − A × x mod p^(i+1)
    y ← C × w / p^i mod p^2
    x ← x + p^i×y mod p^(i+1)
renvoyer x
```

Proposition 3 : (Correction et complexité)
Cet algorithme est correct. Il possède également une complexité en $O(kn^2)$ opérations d'anneau, et il ne fait des produits que sur des valeurs bornées par $\max\{\deg A, p^2\}$.

Démonstration : D'après la proposition 2, nous avons

$$A'(x_{k+1} - x_k) \equiv b - A'x_k \equiv b - Ax_k \mod \langle p^{k+1} \rangle$$

Cette même proposition implique que $x_{k+1} - x_k \equiv 0 \mod \langle p^k \rangle$ donc cette différence s'écrit sous la forme $x_{k+1} - x_k \equiv p^k y_k \mod \langle p^{k+1} \rangle$ pour un certain $y_k \in \mathcal{A}^n$. Ainsi l'équation ci-dessus se réécrit :

$$A'p^k y_k \equiv b - A'x_k \mod \langle p^{k+1} \rangle$$

Cela implique que p^k divise le membre de droite. Pour trouver y_k, et donc x_{k+1}, il suffit alors de résoudre le système :

$$A'y_k = \frac{b - A'x_k}{p^k} \mod \langle p \rangle$$

Ce qui est ce que l'algorithme calcule, d'où la correction.

Pour la complexité, indiçons par i la valeurs des variables au début de l'itération i. Le calcul limitant de la boucle est le produit Ax_i. Or à l'itération précédente, x_i était défini par $x_{i-1} + p^{i-1}y_{i-1}$. Des deux termes de cette somme, Ax_{i-1} est déjà connu (calculé à l'itération précédente) et Ay_{i-1} est de taille bornée (car y_{i-1} est défini modulo $\langle p^2 \rangle$). Ainsi les seules opérations sur des valeurs de taille non-borné sont des sommes, et non des produits matriciels. □

3 Résolution particulière dans l'anneau des entiers

3.1 Résolution modulo p dans $\mathbb{Z}$

Prenons dans cette partie $\mathcal{A} = \mathbb{Z}$ l'anneau des entiers. Nous cherchons toujours à avoir un calcul borné en terme de tailles des entiers (nombre de bits requis pour les représenter).

Notons $\lambda_i = \|C_i(A)\|_2$ la norme euclidienne de la i-ième colonne de A, ainsi que $\lambda_0 = \|b\|_2$. Posons enfin

$$\delta = \prod_{i \in J_n} \lambda_i$$

le produit sur les n plus grand (λ_i), c'est-à-dire que J_n est l'ensemble $[\![0,\, n]\!]$ privé d'un indice i tel que $\lambda_i = \min_{j \in [\![1,\, n]\!]} \lambda_j$

Proposition 4 : (Bornes de la solution)
Les numérateurs et dénominateurs d'une solution sont bornés en valeur absolue par δ.

Démonstration : d'après la règle de Cramer, les éléments d'une solution x s'écrivent

$$x_j = \frac{\det A_j}{\det A}$$

où A_j est la matrice formée par A en remplaçant la j-ième colonne par b. Les deux déterminant étant entiers, nous avons là une écriture rationnelle, nécessairement plus grande en valeur absolue que celle irréductible. D'après l'inégalité d'Hadamard, pour toute matrice $M \in \mathcal{M}_n(\mathbb{R})$, $|\det M| \leqslant \prod_{i=1}^{n} \|C_i(M)\|_2$. Ce qui implique le résultat par définition de δ. □

Notation : Fixons $\beta = \max\{\|A\|_\infty, \|b\|_\infty\}$ une borne des coefficients de A et b en valeur absolue. Nous compterons dans la suite les complexité en fonction du nombre d'opérations entre entiers de taille β.

Toute colonne de A ou b est majorée par celle ne contenant que des β, dont la norme euclidienne est $\sqrt{n\beta^2} = \beta n^{\frac{1}{2}}$. D'où la majoration du produit de n colonnes :

$$\delta \leqslant \beta^n n^{\frac{n}{2}}$$

L'algorithme suivant permet de trouver une solution modulo un nombre p, généralement premier, inférieur à β, et non diviseur de $\det A$.

Algorithme 4 – Résolution dans l'anneau des entiers

Données : $A \in \mathcal{M}_{n\times n}(\mathbb{Z})$ inversible, $b \in \mathbb{Z}^n$, $p \in \mathcal{A}$ et p premier, k la précision
Résultat : $x \in \mathbb{Z}^n$ solution de $Ax \equiv b \mod p^k$
Algorithme :

```
C ← inverse(A mod p, p)   // inverse modulo p
x ← tableau(k, n)         // tableau de m vecteurs de Z^n
x[0] ← C × b mod p
Pour i = 1 à k − 1 faire
    b ← p^{-1} × (b − A × x[i−1])
    x[i] ← C×b mod p
renvoyer x[0] + x[1]×p + ... + x[m−1]×p^{m−1}
```

Proposition 5 : (Correction et complexité)
Cet algorithme est correct, et s'exécute en $O(kn^2 \log n + n^3)$ opérations.

Démonstration : Notons x_i la valeur de la i-ème case du tableau après affectation, et b_i la valeur de b en fin de la boucle d'indice i. Nous obtenons ainsi deux suites de vecteurs (x_i) et (b_i) vérifiant :

$$\begin{cases} b_0 = b \text{ et } \forall i \in [\![1,\ m-1]\!]\ b_i = p^{-1}(b_{i-1} - Ax_{i-1}) \\ \forall i \in [\![0,\ m-1]\!]\ , x_i \equiv Cb_i \mod p \end{cases}$$

De plus, (b_i) est à valeurs dans $\mathbb{Z}^n$ car $b_i - Ax_i \equiv A(Cb_i - x_i) \equiv A(Cb_i - Cb_i) \equiv 0 \mod p$.

Commençons par la correction. Nous avons :

$$Ax = \sum_{i=0}^{k-1} p^i Ax_i = \sum_{i=0}^{k-1} p^i (b_i - pb_{i+1}) = b_0 - p^k b_k \qquad \text{(définition de } b_{i+1})$$

car $\forall i > 0, b_i = p^{-1}(b_{i-1} - Ax_{i-1})$ donc $Ax_{i-1} = b_{i-1} - pb_i$. Cela implique bien que $Ax \equiv b_0 = b \mod p^k$.

Passons à la complexité. L'inversion matricielle peut se faire par des méthodes classiques en $O(n^3)$, voir moins avec des algorithmes plus malins. Le reste des initialisations est en deçà de la complexité annoncée.

Dans la boucle, les x_i sont bornées par $p-1$. Cela implique que $\|Ax_i\|_\infty \leqslant n\beta(p-1)$. Si $\|b_i\|_\infty \leqslant n\beta$ alors $\|b_{i+1}\|_\infty \leqslant \frac{n\beta + n\beta(p-1)}{p} = n\beta$. La récurrence étant initialisée, les (b_i) sont bornée par $n\beta$. Il s'agit donc d'entiers sur $O(\log n + \log \beta)$ bits. Les produit étant entre matrice et vecteurs, il se font en $O(n^2)$ opérations sur des entier de taille $O(\log n)$. Le corps de la boucle est en $O(n^2 \log n)$, ce qui donne la complexité totale annoncée de $O(kn^2 \log n + n^3)$ □

3.2 Décodage vers la forme rationnelle dans $\mathbb{Z}$

Il reste à obtenir la forme rationnelle de la solution à partir de ce calcul. Il serait bien sur possible de prendre m tel que $p^m \geqslant 2\delta$, assurant ainsi que x est la solution grâce aux bornes de la proposition 4. Mais la complexité serait énorme (δ n'est majoré que par $\beta^n n^{\frac{n}{2}}$). Alternativement, il serait possible de faire les calculs modulo plusieurs p et conclure par le théorème du reste chinois. Toutefois nous aurions besoin d'effectuer l'algorithme environ $O(\ln \delta)$ fois (avec de $p < \beta$), ce qui reste peu souhaitable.

Une autre méthode permet de choisir m en $O(n \ln n)$, et de récupérer chaque composant de la solution en $O(m^2)$ opérations à partir de sa composante correspondante de la solution modulo p.

Résultats classiques sur les fractions continues : nous utiliserons sans démonstrations les faits suivants. Étant donné un réel y, on peut définir deux suites (y_n) et (a_n) récursivement par $y_0 = y$, et tant que (y_n) n'est pas entier,

$$a_n = \lfloor y_n \rfloor \text{ et } y_{n+1} = \frac{1}{y_n - a_n}$$

(a_n) est une suite d'entiers positif (sauf éventuellement le premier) appelée fraction continue associé à y. Notons $f_n = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \ldots + \frac{1}{a_n}}}$. Définissons également (u_n) et (v_n) des

suites d'entiers par

$$\begin{array}{lll} u_{-2}=0 & u_{-1}=1 & u_n=a_nu_{n-1}+u_{n-2} \\ v_{-2}=1 & v_{-1}=0 & v_n=a_nv_{n-1}+v_{n-2} \end{array}$$

Nous avons alors les résultats suivants :

1. si $y \in Q$ alors (a_n) et (y_n) sont finies.
2. (f_n) est une suite rationnelle qui converge vers y.
3. $(f_n)=\left(\dfrac{u_n}{v_n}\right)$, les suites (u_n) et (v_n) sont appelées réduites de la fraction continue.
4. (u_n) et (v_n) sont croissantes, et leur termes premiers entre eux.
5. si $y=\frac{p}{q}$ alors, (v_np-u_nq) est alternée est décroit en valeur absolue.

Reformulation du problème : nous voulons calculer une fraction $\frac{f}{g}$ avec $|f|$ et $|g|<\delta$ d'après la proposition 4, en sachant que $gx_r = f \mod p^m$ avec x_r la r-ème composante de la solution modulo p^m.

Proposition 6 :
Supposons $x_r > 1$ et $|f|, |g| < \lambda p^{\frac{m}{2}}$ avec $\lambda = 0.618\ldots$ racine de $\lambda^2+\lambda=1$.

Soient $\frac{w_i}{v_i}$ les réduites de la fraction continue $\frac{x_r}{p^m}$, notons $u_i = v_ix_r - w_ip^m$. Si k est le plus petit entier vérifiant $|u_k| < \sqrt{p^m}$ alors $\dfrac{f}{g}=\dfrac{u_k}{v_k}$.

Démonstration : par propriété des fonctions continues, les suites (w_i) et (v_i) sont croissantes tandis que (u_i) est de signe alterné et de valeur absolue décroissante. De plus, $gx_r = f + tp^m$ pour un certain entier t, car $gx_r \equiv f \mod p^m$, donc

$$\left|\frac{x_r}{p^m}-\frac{t}{g}\right| = \left|\frac{x_rg-tp^m}{p^mg}\right| = \left|\frac{fg}{p^mg^2}\right| < \frac{1}{2g^2}$$

Cela implique que $\frac{t}{g}$ est l'une des réduites de $\frac{x_r}{p^m}$. (théorème 19, Khinhin *Continued fractions*, 3ème édition, Chicago Press, 1961) Notons j son indice.

w_j et v_j étant premiers entre eux, $|u_j| \leqslant |f| \leqslant \lambda\sqrt{p^m}$, et donc par minimalité de k cela implique $j \geqslant k$. De plus $u_j = v_jx_r - w_jp^m$ et $u_k = v_kx_r - w_kp^m$ ce qui donne $u_jv_k - u_kv_j \equiv 0[p^m]$.

Enfin, comme $j \geqslant k$ nous avons

$$|u_jv_k - u_kv_j| \leqslant (|u_j| + |u_k|)\,|v_j| < (\lambda+1)\lambda p^m = p^m$$

Nous en déduisons $u_jv_k = u_kv_j$ puis $j=k$, donc $\dfrac{f}{g}=\dfrac{u_k}{v_k}$ ce qui achève la preuve. □

Algorithme : la valeur se calcule alors avec une version modifiée de l'algorithme d'Euclide. En prenant m tel que $x_r \leqslant p^m$. Nous allons calculer simultanément trois suites $(|u_i|)$, (v_i) et (q_i) définies par

$$\begin{array}{lll} |u_{-1}| = p^m & |u_0| = x_r & |u_{i+1}| = |u_{i-1}| - q_i\,|u_i| \\ v_{-1} = 0 & v_0 = 1 & v_{i+1} = v_{i-1} + q_iv_i \\ q_i = \left\lfloor \left| \frac{u_{i-1}}{u_i} \right| \right\rfloor & & \end{array}$$

Par la proposition 6, il suffit de calculer ces suites jusqu'au plus grand k vérifiant $|u_k| < \sqrt{p^m}$. Comme nous calculons la valeur absolue de (u_i), il faut rajouter un $(-1)^i$ pour avoir la bonne valeur

Algorithme 5 – Décodage vers la forme rationnelle

Données : p, m, x_r la r-ième coordonnée de la solution modulo p.
Résultat : s_r la r-ème coordonnée de la solution rationnelle.
Algorithme :

```
u1 ← p^m
u2 ← x_r
v1 ← 0
v2 ← 1
i  ← −1
Tant que u2 < p^(m/2) faire
    q ← partieEntiere(u1 / u2)
    u1, u2 ← u2, u1 − q × u2
    v1, v2 ← v2, v1 + q × v2
    i ← i+1
renvoyer (−1)^i × fraction(u1, v1)
```

Proposition 7 : (Correction et complexité)
Cet algorithme est correct et de complexité $O(m^2)$ opérations sur des entiers de taille au plus β.

Démonstration : La correction découle de la proposition 6.

Pour la complexité, nous avons $v_i \geqslant \prod_{j=0}^{i-1} q_j$ et $v_i \geqslant \sqrt{2^i}$ par récurrence. En effet, ces inégalités sont vraies au rangs 0 et 1. Si elles sont vraies au rangs i et $i-1$, alors

— les v_i sont positifs donc $v_{i+1} = v_{i-1} + q_i v_i \geqslant 0 + \prod_{j=0}^{i-1} q_j$

— $v_{i+1} = v_{i-1} + q_i v_i \geqslant 2^{\frac{i-1}{2}} + 2^{\frac{i}{2}} = 2^{\frac{i+1}{2}} \underbrace{\left(\frac{1}{2} + \frac{1}{\sqrt{2}}\right)}_{\geqslant 1}$

De plus $(-1)^k \frac{u_k}{v_k} = \frac{f}{g}$ ce qui implique $|v_k| \leqslant |g| < \sqrt{p^m}$. Donc en passant au logarithme dans les deux minorants de v_i montrés ci-dessus, $\sum_{j=0}^{k-1} \log q_i$ et k sont en $O(\log p^m) = O(m)$. Les entiers $|u_i|$ et v_i sont bornés par $\sqrt{p^m}$, donc s'expriment en $O(\log p^m) = O(m)$ bits. L'algorithme s'exécutant en $O(k)$ opérations d'anneau, il est en $O(m^2)$ opérations élémentaires sur des entiers de taille au plus β. □

Complexité globale : la résolution est correcte pourvue que $\delta < \lambda p^{\frac{m}{2}}$ d'après la proposition 6. Prenons donc

$$m = \left\lceil 2 \frac{\ln \delta}{\ln \lambda p} \right\rceil$$

Or $\delta \leqslant \beta^n \sqrt{n^n}$ ce qui implique $m = O(n \log n)$. Cette deuxième étape est alors en $O(n^2 \ln^2 n)$ pour le calcul d'une coordonnée, soit un $O(n^3 \ln^2 n)$ pour le calcul de toutes les coordonnées.

La complexité globale des deux parties (résolution modulaire puis décodage rationnel), est donc en $O(n^3 \ln^2 n)$ opérations sur des nombre de taille au plus β.

4 Cas des polynômes, relèvement P-adique

4.1 Définition et propriétés de l'expansion P-adique

Dans cette partie, nous nous intéressons à $\mathcal{A} = \mathbb{K}[x]$ l'anneau de polynômes. Fixons un polynôme P non-constant et premier avec le déterminant de A. Nous ferons des calculs à partir des expansion P-adique des fraction rationnelles de $\mathbb{K}(x)$. En effet, pour toute fraction $f \in \mathbb{K}(x)$ dont le dénominateur est premier avec P, il existe une unique écriture

$$f = \sum_{i=0}^{\infty} C_i P^i$$

avec (C_i) une famille de polynôme de degré strictement inférieur à celui de P.

Nous utiliserons surtout cette expansion sur des matrices, l'expansion P-adique d'une matrice F à même forme que ci-dessus, avec cette fois-ci (C_i) des matrice de même taille que F, dont tous les coefficients sont à degré strictement inférieur à P.

Notre objectif est trouver des algorithmes rapides pour calculer une partie du développement P-adique de $A^{-1}b$, dans le but de résoudre le système $Ax = b$. Nous supposerons donc dans toute la suite que P est premier avec le déterminant de A (dénominateur de cette fraction rationnelle).

Si $\frac{\deg b}{\deg A}$ et $m(1 + \frac{\deg B}{\deg A})$ sont en $O(n)$, ce système peut être résolu en $O(n^\theta \deg A)$ opérations élémentaires du corps $\mathbb{K}$, avec $\theta \leqslant 3$ l'exposant de multiplication matricielle.

Modèle de complexité : Nous considérerons élémentaires les opérations du corps $\mathbb{K}$ $(+, \times, -, /)$. La multiplication de polynômes de degré au plus d est alors en $O(d^{1+\varepsilon})$ et les multiplication de matrice $n \times n$ est en $O(n^\theta)$.

En plus de ces opérations d'anneau, nous introduisons trois fonctions, intimement liées à la notion d'expansion P-adique. Pour une fraction rationnelle Q d'expansion P-adique $Q = \sum_{i=0}^{\infty} q_i P^i$, posons :

- Gauche$(Q, k) = \sum_{i=0}^{\infty} q_{i+k} P^i$ le décalage de k vers la gauche.
- Tronc$(Q, k) = \sum_{i=0}^{k-1} q_i P^i$ la troncature à l'ordre k
- si Q est un polynôme premier avec P, alors Inverse(Q, k) est l'unique $R \in \mathbb{K}[x]$ tel que $R =$ Tronc(R, k) et Tronc$(RQ, k) =$ Tronc$(QR, k) = 1$

Ces définitions s'étendent linéairement aux matrices de polynômes.

Leur coût dépend de la représentation choisie. Une façon simple et relativement efficace est de les représenter sous forme de tableau. Avec cette représentation, l'addition et la soustraction s'effectuent en $O(1 + \min(\deg Q, \deg R)$. Le produit nécessite un tempe en $O((1 + \deg Q)(1 + \deg R)^\varepsilon)$. Les opérations Gauche, Tronc peuvent se calculer en temps constant par décalage de pointeurs, et l'inverse s'effectue en $O((k \deg P)^{1+\varepsilon})$

Proposition 8 : (Propriétés élémentaires)
Ces trois opérations ont les propriétés suivantes :

1. Gauche est linéaire en a.
2. si Q est un polynôme vérifiant $\deg Q < \deg P^k$, $\text{Gauche}(Q, k) = 0$
3. $\text{Gauche}(\text{Tronc}(q, k), \ell) = \text{Tronc}(\text{Gauche}(a, \ell), k - \ell)$
4. si $\deg R < \deg(P^k)$ alors $\text{Gauche}(Q + R, k) = \text{Gauche}(Q, k)$

Démonstration : ces propriétés découlent simplement des définitions. □

4.2 Relèvement P-adique

Principe : soit b un vecteur colonne, supposons $\deg A$ et $\deg b \leqslant \deg P$, nous cherchons l'expansion de $A^{-1}b$ jusqu'à l'ordre P^k. Pour cela, commençons par un premier sous-problème, chercher l'expansion de $A^{-1}b$ jusqu'à l'ordre $P^{\frac{k}{2}}$.

$$A^{-1}b \equiv c_0 + c_1 P + \ldots + c_{\frac{k}{2}-1} P^{\frac{k}{2}-1} \mod P^{\frac{k}{2}}$$

En remplaçant le modulo par un résidu $A^{-1} r_{\frac{k}{2}} P^{\frac{k}{2}}$, nous obtenons :

$$A^{-1}b = c_0 + c_1 P + \ldots + c_{\frac{k}{2}-1} P^{\frac{k}{2}-1} + A^{-1} r_{\frac{k}{2}} P^{\frac{k}{2}}$$

Ce qui permet d'obtenir $r_{\frac{k}{2}} = \dfrac{b - A(c_0 + c_1 P + \ldots + c_{\frac{k}{2}-1} P^{\frac{k}{2}-1})}{P^{\frac{k}{2}}}$, et implique que $r_{\frac{k}{2}} \in \mathcal{M}_{n,1}(\mathbb{K}[x])$, et vérifie $\deg r_{\frac{k}{2}} < \deg P$. Ainsi le deuxième sous-problème, à savoir calculer l'expansion de $A^{-1} r_{\frac{k}{2}}$ jusqu'à $P^{\frac{k}{2}}$ a la même forme que le premier. Il faut néanmoins résoudre le premier pour connaitre $r_{\frac{k}{2}}$ avant de pouvoir s'attaquer au second.

Proposition 9 : (Inverse matricielle)
Soit $A \in \mathcal{M}_n(\mathbb{K}[x])$ inversible vérifiant $\det A \perp P$. Soit $B \in \mathcal{M}_{n \times m}(\mathbb{K}[x])$ et $(k, \ell) \in \mathbb{N}^*$. Définissons trois matrices à partir de d'expansion P-adique de A^{-1} et $A^{-1}B$:

— $C = \text{Tronc}(A^{-1}, \ell)$

$$A^{-1} = \underbrace{* + *P + \ldots + *P^{\ell-1}}_{C} + * P^{\ell} + \ldots$$

— $D = \text{Tronc}(A^{-1}B, k)$
— $E = \text{Tronc}(\text{Gauche}(A^{-1}B, k), \ell)$

$$A^{-1}B = \underbrace{* + *P + \ldots + *P^{k-1}}_{D} + \underbrace{*P^k + \ldots + *P^{k+\ell-1}}_{EP^k} + * P^{k+\ell} + \ldots$$

Alors

$$E = \text{Tronc}(C \cdot \text{Gauche}(B - AD, k), \ell)$$

Démonstration :

$$\begin{aligned}
AD &= A(A^{-1}B - EP^k - *P^{k+\ell}) \\
B - AD &= AEP^k - *P^{k+\ell} \\
\text{Gauche}(B - AD, k) &= AE \\
C \cdot \text{Gauche}(B - AD, k) &= CAE = (A^{-1} - *P^{\ell})AE = E - *P^{\ell} \\
\text{Tronc}(C \cdot \text{Gauche}(B - AD, k), \ell) &= E
\end{aligned}$$

□

Calcul des composants de haut ordre : Notons $Z_i = \text{Inverse}(A, 2^i)$. Nous allons chercher à récupérer les composants de haut ordre de A^{-1}, $E_i = \text{Gauche}(Z_i, 2^i - 2)$. Notons l'expansion P-adique de $A^{-1} = \sum_{i=0}^{\infty} c_i X^i$ alors :

$$
\begin{aligned}
Z_1 &= \underbrace{c_0 + c_1 X}_{E_1} \\
Z_2 &= c_0 + c_1 X + \underbrace{c_2 X^2 + c_3 X^3}_{E_2 P^2} \\
Z_3 &= c_0 + c_1 X + c_2 X^2 + c_3 X^3 + c_4 X^4 + c_5 X^5 + \underbrace{c_6 X^6 + c_7 X^7}_{E_3 P^6}
\end{aligned}
$$

En partant de Z_0 nous pouvons calculer les E_k par la méthode suivante, si l'on suppose $\deg A \leqslant \deg P$.

Algorithme 6 – Calcul des composants de haut ordre

Données : $A \in \mathcal{M}_n(\mathbb{K}[x])$ et $k \in \mathbb{N}$
Résultat : $(E_1, \ldots, E_k)$
Algorithme : ComposantsHautOrdre[P](A, k)

```
L ← Inverse(A, 1) // Z_0
H ← Tronc(L × Gauche(I − A×L, 1), 1)
E_1 ← L + P×H
Pour i allant de 2 à k faire
    L ← Tronc(Gauche(E_{i−1} Gauche(−A×L, 1), 1), 1)
    H ← Tronc(Gauche(E_{i−1} Gauche(−A×H, 1), 1), 1)
    E_i ← L + P×H
renvoyer (E_1, ..., E_k)
```

Proposition 10 : (Correction et complexité)
Cet algorithme renvoie les E_i comme défini ci-dessus, et s'exécute en $O(kn^\theta(\deg P)^{1+\varepsilon})$

Démonstration : les valeurs manipulées étant relativement petites (inférieures en degré à $2\deg P$ par troncature), les produits polynomiaux sont en $O\big((\deg P)^{1+\varepsilon}\big)$. Le corps de la boucle est alors en $O(n^\theta)$, d'où la complexité annoncée.

Soit L_i, H_i les valeurs de L et H après la boucle d'indice i. Montrons par récurrence sur $j \in [\![1,\ k]\!]$ la propriété suivante :

$$
\begin{aligned}
L_j &= c_{2j-2} \\
H_j &= c_{2j-1} \\
E_j &= c_{2j-2} + c_{2j-1} P
\end{aligned}
$$

Pour l'initialisation, nous avons $L_1 = Z_0 = c_1$. Ensuite $H_1 = c_2$ découle de la proposition 9 avec $k = \ell = 1$ et $B = I_n$, enfin E_1 est vrai par définition.

Passons à l'hérédité. Il est clair que les deux premières équations impliquent la troisième, qui vaut bien E_i par définition de ce dernier (pas de conflit entre la notation des variables de l'algorithme et celles introduites avant). Les deux premières relations se montrent de façon similaires, il suffira donc de faire la preuve de la deuxième. Soit $i \geqslant 1$, supposons le résultat au rang i.

La boucle calcule $H_{i+1} = \text{Tronc}\Big(\underbrace{\text{Gauche}\big(E_i \overbrace{\text{Gauche}(-AH_i, 1)}^{R}\big), 1\big)}_{S}, 1\Big)$. De plus nous avons par hypothèse d'induction les encadrements de degré :

$$\deg(Z_i - P^{2^i-1}H_i) < \deg P^{2^i-1}$$
$$\deg(Z_i - P^{2^i-2}E_i) < \deg P^{2^i-2}$$

Posons $Q = \dfrac{(I - AZ_i)}{P^{2^{i+1}-1}}$ et montrons que $R = Q$. Soit $T = -AH_i - PQ$. D'après la deuxième inégalité de degré, comme $H_i = \text{Gauche}(E_i, 1)$ par hypothèse de récurrence, nous avons $\deg T < \deg A \leqslant \deg P$, de plus $-AH_i = PQ + T$ et $\deg T < \deg P$ donc d'après la proposition 8, point 4,

$$R = \text{Gauche}(-AH_i, 1) = \text{Gauche}(PQ, 1) = Q$$

Pour conclure, posons $U = Z_iR$ et $V = E_iR - U$. Comme $R = Q$ donne $\deg R < \deg P$, et d'après la première inégalité de degré, nous obtenons $\deg V < \deg(P)$. Ainsi en réutilisant la proposition 8, point 4, nous obtenons

$$S = \text{Gauche}(E_iR, 1) = \text{Gauche}(U + V, 1) = \text{Gauche}(U, 1)$$

En utilisant la proposition 8, point 3, nous pouvons conclure. □

4.3 Solution en série

Nous avons maintenant les outils pour trouver une l'expansion P-adique de notre système $Ax = b$, que nous calculons avec l'algorithme suivant :

Données : $A \in \mathcal{M}_n(\mathbb{K}[x]), b \in \mathbb{K}[x]^n$ et $k \in \mathbb{N}$
Résultat : $x \in \mathbb{K}[x]^n$ début de la série solution $x = \text{Tronc}(\text{Inverse}(A, 2^k)b, 2^k)$
Algorithme : `SerieSolution[P](A, b, k)`

```
(E_1, ..., E_k) ← ComposantsHautOrdre[P](A, k)
B ← [b_0 | b_1 | ... | b_{2^k-1}]  // expansion P-adique de b
Pour i allant de k − 1 à 1 faire
    B ← premières (2^k − 2^i) colonnes de B
    B ← Gauche(−A × Tronc(Gauche(E_i B, 1), 1), 1)
    B ← [0_{n×m2^i} | B]   // on rajoute m2^i colonnes pour avoir n × 2^k)
    B ← B + B
B ← Tronc(E_1 B, 2)
renvoyer B[0] + ... + B[2^k−2] × P^{2^k−2} + Tronc(B[2^k−1], 1) P^{2^k−1}
// avec B[i] la i-ième colonne de B
```

Proposition 11 : (Correction et complexité)
Cet algorithme est correct et s'exécute en $O\left(k2^k n^{\theta-1}(\deg P)^{1+\varepsilon}\right)$ opérations de corps.

Démonstration : effectuons les deux preuves en simultané. L'algorithme commence par calculer les composants de haut ordre en $O(kn^{\theta-1}2^k(\deg P)^{1+\varepsilon})$ d'après la proposition 10.

Pour la boucle, notons $f(m,i)$ le coût de calcul de $\mathrm{Tronc}(Z_kQ, 2^i)$ pour un certain $Q \in \mathcal{M}_{n\times m}(\mathbb{K}[x])$ de degré inférieur à $\deg P$. Pour $i > 1$, nous admettrons le résultat suivant, issu de l'étude du relèvement P-adique quadratique :

$$\mathrm{Tronc}(Z_iB, 2^i) = \mathrm{Tronc}(Z_{i-1}B, 2^{i-1}) + \mathrm{Tronc}(Z_{i-1}H, 2^{i-1})P^{2^{i-1}}$$

avec $H = \mathrm{Gauche}(B - A\mathrm{Tronc}(Z_{i-1}B, 2^{i-1}), 2^{i-1})$. La quantité de gauche est celle que nous cherchons à calculer, car elle correspond aux coefficients de l'expansion P-adique de notre solution. Le degré de H est strictement inférieur à $\deg A$ inférieur à celui de P, ainsi en reprenant la formule ci-dessus, nous avons

$$f(i,m) \leqslant f(i-1,m) + \text{ coût du calcul de } H$$

De plus, comme Z_i et $P^{2^i-2}E_i$ ont même coefficients aux hauts ordres, ils ont même translation gauche de $2^i - 2$, ce qui permet de réécrire

$$H = \mathrm{Gauche}(-A\mathrm{Tronc}(\mathrm{Gauche}(E_{i-1}B, 1), 1), 1)$$

C'est cette formule qu'utilise l'algorithme. Si $i = 1$, alors $\mathrm{Tronc}(Z_1B, 2)$ se calcule directement, car $Z_1 = E_1$ est connu. Puis à chaque itération de la boucle nous doublons le nombre de coefficients connus par la formule admise.

Le nombre de colonnes non-nulles étant doublé à chaque itération, la dernière itération domine toutes les précédentes en complexité, d'où le coût en $O(kn^{\theta-1}2^k(\deg P)^{1+\varepsilon})$

Enfin, l'algorithme termine en additionnant les coefficients calculés avec les bonnes puissances. Ce qui se fait en temps dominé par les étapes précédentes. □

5 Conclusion

Nous avons présenté trois différentes méthodes pour calculer des expansions en séries de solution modulo un idéal. La première méthode se place dans le cas général d'un anneau $\mathcal{A}$ intègre et commutatif. Elle calcule les k premier termes d'une série modulo un idéal p avec un complexité relativement bonne en $O(n^3 + kn^2)$ opérations d'anneau. Selon l'anneau prix en compte, les calculs peuvent en réalité coûter plus cher, nous avons donc étudié un raffinement de cet algorithme général cherchant à faire des produits sur des valeurs bornées, il obtient une complexité en $O(kn^2)$ opérations d'anneau.

Pour la suite nous avons étudié le cas de l'anneau des entiers, et nous nous sommes intéressés à un algorithme en $O(kn^2 \log n + n^3)$ permettant d'obtenir les k premier coefficient de l'expansion en arithmétique p-adique. Cette complexité compte cette fois également les additions sur de nombre de taille bornés, et n'est donc pas nécessairement plus lente que la précédente. De plus, l'algorithme permet de recouvrir les solutions exactes en temps $O(n^3 \log^2 n)$, ce qui est relativement plus efficace que les méthodes proposées avec la solution générale utilisant un calcul de déterminant.

Finalement nous avons abordé le cas d'une expansion P-adique dans l'anneau des polynômes. Grâce à une troisième méthode, nous parvenons à calculer une série solution en $O\left(k2^k n^{\theta-1}(\deg P)^{1+\varepsilon}\right)$ opérations de corps (et non de polynômes), pour une série cette fois de 2^k termes. Il est donc comparable en terme de complexité, pourvu que P ne soit pas de trop grand degré, aux algorithmes précédents. La théorie utilisée ici peut de plus permettre des algorithmes similaires pour récupérer les composants d'ordre élevé de ce développement. Cela permet d'obtenir des méthodes intéressantes, en terme de complexité, permettant de certifier l'intégrité de $A^{-1}b$ et de calculer le déterminant de A.