Rapport de stage au LIRMM du 6 juin au 26 juillet 2019

Résolution efficace de systèmes d'équations linéaires à coefficients entiers ou polynomiaux: étude et implémentation

Dorian Lesbre

Encadrants: Romain Lebreton et Pascal Giorgi

14 août 2019

Table des matières

Références

[1] John D. Dixon. Exact solutions of linear equations using p-adic expansions. *Numer. Math*, (40.1) :137–141, février 1982.

[2] A. Bostan F. Chyzak M. Giusti R. Lebreton G. Lecerf B. Salvy et E. Schost. *Algorithmes efficaces en calcul formel*. 2018.

[3] E. Kaltofen et G. Yuhasz. On the matrix berlekamp-massey algorithm. *ACM Transactions on Algorithms*, 9(4), septembre 2013.

[4] J. Galten et J. Gerhard. *Modern Computer Algebra*. Cambridge, 2013. Third Edition.

[5] 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.

[6] Arne Storjohann. High-order lifting. *Proceedings of the 2002 international Symposium on Symbolic and Algebraic computation*, (ISSAC 2002 Lille, France) :246–254, ACM, 2002.

1 Introduction

L'objectif de ce stage était de s'intéresser à des algorithmes efficaces de résolution modulaire de systèmes matriciels polynomiaux. Plus précisément, nous cherchions à optimiser la complexité en comptant comme élémentaire les opérations dans le corps de base des polynômes. Ces résolutions modulaires permettent à la fois de limiter cette complexité en travaillant sur des valeurs bornées, mais elle donnent également les résolutions exactes par reconstruction rationnelle pourvu que la résolution modulaire soit suffisamment précise.

Mon travail a consisté à prendre connaissance des complexités connues du calcul matriciel et polynomial. Puis de s'en servir pour comprendre, prouver et exhiber les complexités des différents algorithmes de résolutions ([1], [5] et [6]), notamment ceux introduits par Storjohann. Couplée à cette partie théorique venait l'implémentation en SageMath de ces algorithmes afin de les vérifier et comparer leurs temps d'exécutions expérimentaux. Les figures de ce rapport proviennent toute de ces tests. Enfin, vers la fin du stage, nous avons essayé de modifier le code source SageMath (en cython) pour y rajouter des produits optimisés sur les matrices polynomiales de la bibliothèque Linbox, afin d'avoir de meilleurs résultats sur les tests de vitesse, mais sans avoir le temps de finir.

2 Complexités en algèbre matricielle et polynomiale

Dans cette partie, nous présentons les différents algorithmes connus pour les calculs en algèbre matricielle et polynômiale, dans le but de définir le coût des différentes opérations élémentaires que nous utiliserons dans la suite. Nous ne démontrerons pas ici ces algorithmes et complexités, et nous contenterons de les présenter. Ils sont couverts en détail dans [2] et [4].

2.1 Calculs dans un anneau de polynômes

Soit $\mathbb{K}[X]$ un anneau de polynômes dans un corps $\mathbb{K}$. Les polynômes sont classiquement représentés par des tableaux, ce qui donne un accès à chaque coefficient en temps constant. La somme de deux polynômes de degré borné par d est alors en $O(d)$ (somme terme à terme). Cette complexité est clairement optimale (temps nécessaire à lire les entrées).

Produits de polynômes : plusieurs algorithmes existent avec diverses complexités et contraintes,

- l'algorithme naïf, partant de la formule du produit de Cauchy, est en $O(d^2)$
- Karatsuba publie en 1962 un produit en $O(d^{\log_2 3}) \approx O(d^{1.585})$ qui combine les produits récursifs de moitié des polynôme.
- Similairement, Toom-3 publié en 1963 obtient une complexité en $O(d^{\log_3 5}) \approx O(d^{1.46})$ en faisant des produits de tiers de polynôme.
- Une version généralisée, Toom-k (1966), coupe les polynômes en k parties et a une complexité en $O(c(k)d^{\log_k(2k-1)})$, ce qui tend vers 1. Cependant les constantes $c(k)$ croient trop rapidement pour rendre cet algorithme avantageux.
- en 1965, la FFT permet un produit en $O(d \log d)$ dans tout corps ayant des racines $2d$-ièmes de l'unité.
- en 1971, Schönhage et Strassen développent une méthode en $O(d \log d \log(\log d))$ sur un corps quelconque.
- en 2019, Harvey-Hoeven obtient une complexité de $O(d \log d)$ sur les corps premiers $\mathbb{F}_p$.

En pratique, les algorithmes utilisés sont la FFT si possible, Karatsuba, Toom-3 ou Schönage et Strassen dans les cas favorables, et sinon le naïf.

Dans toute la suite, nous noterons $M(d)$ la complexité du produit entre deux polynômes de degré inférieur à d. Selon l'algorithme utilisé, elle peut valoir l'une des valeurs présentées ci-dessus

Autres opérations : la division euclidienne peut-être réalisé avec la même complexité que le produit. À partir de cela, l'algorithme d'Euclide permet un calcul de pgcd et d'inverse modulaire en $O(M(d)\log d)$.

De plus, l'évaluation multipoints (avec d points) et l'interpolation peuvent se faire également en $O(M(d)\log d)$ par divisions euclidiennes successives (algorithme présenté dans la section 5.4 de [2]).

Produit médian : Pour deux polynômes P, Q de degrés respectifs $2d$ et d, le produit médian $\mathrm{mp}(P, Q)$ est le polynôme des termes de degrés d à $2d$ du produit $P \times Q$. Ce calcul peut se faire avec une complexité équivalente à celle du produit de deux polynômes de taille d, et non de taille $2d$.

2.2 Calculs dans un anneau matriciel

Soit $\mathcal{M}_n(\mathcal{A})$ un anneau de matrices sur l'anneau $\mathcal{A}$. Nous comptons ici comme élémentaires les opérations de l'anneau $\mathcal{A}$, et nous intéressons aux complexités des opérations matricielles en fonction de n. La somme naïve en $O(n^2)$ est ici encore optimale.

Produit matriciel : tout comme pour les polynômes, plusieurs algorithmes permettent d'effectuer ce produit,

— l'algorithme naïf est en $O(n^3)$

— en 1969, Strassen établit un algorithme diviser pour régner en $O(n^{\log_2 7}) \approx O(n^{2.81})$

— en 2014, Le Gall montre qu'il existe un algorithme en $O(n^\omega)$ avec $\omega \in [2, 2.37[$.

En pratique, les algorithmes utilisés sont le naïf et Strassen. Le naïf possède une faible constante du fait de d'optimisation bas niveau (bonne utilisation des mémoires caches).

Dans la suite, nous noterons $MM(n) = n^\omega$ la complexité du produit matriciel.

La sous-linéarité du produit permet de gagner en complexité en faisant des produits complets avec des matrices carrées plutôt que plusieurs produits entre matrice et vecteur.

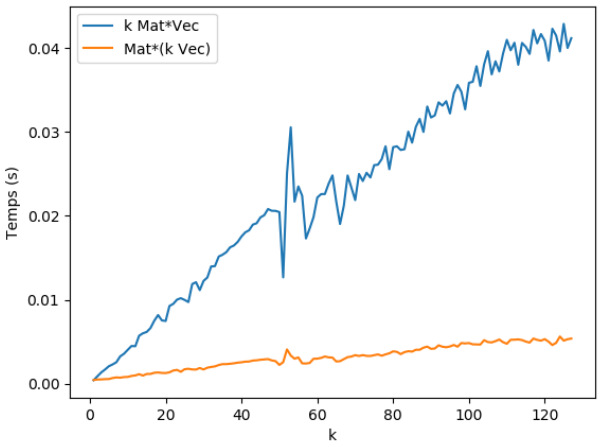

FIGURE 1 – Temps d'exécution de k produit matrices $(n \times n)$ vecteurs $(n \times 1)$ et d'un produit matrice $(n \times n)$ matrice $(n \times k)$ pour $n = 128$ dans le corps premier $\mathbb{F}_{524309}$

Réductions : nous pouvons réaliser, avec la même complexité que la multiplication, de nombreuses autres opérations sur les matrices. En effet, il est possible de faire une décomposition LU généralisée avec cette complexité.

À partir de cette décomposition, le calcul de déterminant est simple, donc en $O(MM(n))$ également. L'inverse de matrice triangulaire étant simple, l'inversion est aussi en même complexité que le produit. Il s'ensuit que la résolution de systèmes linéaires est donc naturellement en $O(MM(n))$. De même, cette décomposition permet le calcul du rang en $O(MM(n))$.

Enfin, les polynômes minimaux et caractéristiques peuvent être obtenu en $O(MM(n)\log n)$.

2.3 Algorithmique des matrices polynomiales

Intéressons nous maintenant à l'anneau des matrices polynomiales $\mathcal{M}_n(\mathbb{K}[X])$ sur un corps $\mathbb{K}$. Pour les complexités, les opérations de $\mathbb{K}$ seront considérées élémentaires. Une fois de plus la somme de matrices de taille $n \times m$ et de degré d est optimale en $O(nmd)$.

Produit : pour effectuer le produit, nous pouvons appliquer directement les algorithmes matriciels aux polynômes. Toutefois, ces algorithmes ne sont pas garantis de ne travailler que sur des polynômes de degré inférieur à d pour les calculs intermédiaires. Pour y remédier, nous appliquons ces calculs modulo X^{2d} (faire des modulo par un monôme est une opération élémentaire), ce qui suffit pour obtenir le résultat. Ainsi cette première méthode a une complexité en $O(MM(n)M(d))$.

Une autre méthode procède par évaluation interpolation. Nous évaluons les deux matrices en $2d+1$ points, faisons les $2d+1$ produits des matrices évaluées, puis interpolons les matrices obtenues pour avoir les polynômes solutions. Cette méthode est en $O(MM(n)d + n^2M(d))$.

Notons que cette méthode a l'avantage, lorsque l'on multiplie de nombreuse fois par une même matrice, de précalculer l'évaluation de celle-ci et ne faire que l'évaluation de l'autre matrice et l'interpolation. Cela permet de gagner asymptotiquement dans le cas de produits matrices vecteurs, car une fois les n^2 évaluations des polynômes de la matrices faites, il ne reste que n évaluations, d produits matrice-vecteur (en $O(n^2)$, et n interpolations, d'où une complexité en $O(n^2d + nM(d))$.

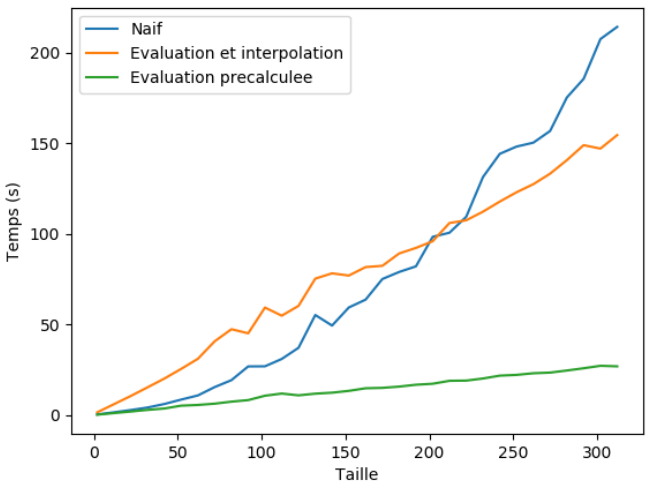

FIGURE 2 – Temps d'exécution de différents produits de matrices polynomiales sur $\mathbb{F}_{524309}$ de degré $d = 128$ en fonction de la taille des matrices

Résolution de système : pour résoudre un système de matrices polynomiales, nous pouvons reprendre l'idée, déjà utilisée pour le produit, de faire les calculs intermédiaires modulo une borne de la solution. Toutefois, le déterminant et les mineurs sont de degré borné par nd, donc le complexité de cette résolution naïve est en $O(MM(n)M(nd))$.

Des algorithmes par approximations successives permettent d'obtenir des meilleures complexités. En se basant sur un calcul d'inverse imprécis puis raffinant itérativement la solution, comme avec les algorithme développés par Dixon, puis Mœnck et Carter dans [1] et [5], nous obtenons des complexités de $O(n^3M(d))$.

Enfin, avec le high-order lifting de Storjohann dans [6], nous obtenons une complexité de résolution en $O(n^\omega M(d)\log n)$.

3 Résolution modulaire de systèmes linéaires

Cette partie couvre le travail théorique qui a été fait durant le stage dans le but de développer une autre intuition, et une autre démonstration, du high-order lifting introduit par Storjohann dans [6]. Nous partons de la notion de suites récurrentes de matrices et du problème du calcul des termes suivants, pour ensuite montrer que ce problème est une reformulation de la résolution de système.

Dans cette section, nous fixons $n \in \mathbb{N}^*$ une taille de matrice et $d \in \mathbb{N}^*$ un degré.

3.1 Suites récurrentes de matrices

Définition : Pour un $m \in \mathbb{N}^*$ fixé et $(C_i)_{i\in[\![0,\,d]\!]} \in \mathcal{M}_n(\mathbb{K})^{d+1}$ $d+1$ matrices carrées non toutes nulles, une suite récurrente de matrices $(U_i) \in \mathcal{M}_{n\times m}(\mathbb{K})$ pour $(C_i)_{i\in[\![0,\,d]\!]}$ est une suite vérifiant :

$$\forall \ell \in \mathbb{N}, \sum_{i=0}^{d} C_i U_{i+\ell} = 0_{n\times m}$$

Une définition plus formelle est présentée dans [3].

Remarque : les matrices étant non-commutatives, nous parlons ici de suite récurrente gauche, qui sont celles qui apparaissent naturellement dans le cadre de la résolution de système. Les suites droites se définissent de manière analogue.

Séries formelles : nous noterons $U(X)$ la série formelle à coefficients matriciels $\sum_{k=0}^{+\infty} U_k X^k$. Fixons également $C(X)$ le polynôme, dit générateur gauche, $\sum_{k=0}^{d} C_{d-k} X^k$. Remarquons l'inversion de l'ordre des coefficients dans $C(X)$ qui sert à alléger les notations dans la suite.

Théorème 1 : (rationalité de $U(X)$)
Il existe $N(X) \in \mathbb{K}[X]$ de degré inférieur à $d-1$ telle que

$$C(X)U(X) = N(X)$$

Si $C(X)$ est inversible, alors $U(X)$ est donc une fraction rationnelle matricielle, ce qui garantit l'unicité (et donc la bonne définition) de la suite (U_i).

Démonstration : Il suffit de développer le produit,

$$C(X) \times U(X) = \underbrace{\sum_{i<d} \left(\sum_{k+\ell=i} C_{d-k} U_\ell \right) X^i}_{N(X)} + \sum_{i \geqslant d} \underbrace{\left(\sum_{j=0}^{d} C_j U_{i-d+j} \right)}_{=0 \text{ par défintion de la suite}} X^i \quad \square$$

3.2 Calculs de termes suivants

Problème 1 : (Calcul des termes suivants)
Étant donné le polynôme générateur $C(X)$, et les premiers termes $(U_0, \ldots, U_{d-1})$, nous voulons calculer les termes suivants de la suite (U_i).

Algorithme 1 : (Résolution naïve)
En considérant le résultat du théorème 1, nous pouvons résoudre le problème 1 en calculant $C(X)^{-1}$ et $N(X)$:

Algorithme 1 : RésolutionNaive

Données : d, $C(X)$ et $U = \sum_{j=0}^{d-1} U_j X^j$
Résultat : $(U_d, \ldots U_{2d-1})$
début
  $D \leftarrow \text{inverse}(C, X^{2d})$
  $N \leftarrow C \times U \mod X^d$
  $U \leftarrow D \times N \mod X^{2d}$
  renvoyer $U[d : 2d]$
fin

Notation X^d-adique : Dans la suite, nous utiliserons la notation $\widehat{U_i}$ pour représenter les coefficients X^d-adiques de U :

$$\forall i \in \mathbb{N}, \widehat{U_i} = \sum_{k=0}^{d-1} U_{k+id} X^k \quad \text{et} \quad U(X) = \sum_{k=0}^{+\infty} \widehat{U_k} X^{kd}$$

À la différence du X-adique, en X^d adique le produit de deux termes déborde sur le terme suivant. Nous noterons ainsi $\overline{\widehat{U_i}\widehat{C_0}}$ le reste du produit de ces deux termes par X^d et $\underline{\widehat{U_i}\widehat{C_0}}$ leur quotient :

$$\widehat{U_i}\widehat{C_0} = \overline{\widehat{U_i}\widehat{C_0}} + X^d \times \underline{\widehat{U_i}\widehat{C_0}}$$

C et N étant de degré inférieur à d, ils sont constants en X^d-adique, on les notera donc sans chapeau ni indice. Nous noterons également $D = \sum \widehat{D_i} X^{id}$ l'inverse de C.

Remarque : la méthode précédente a le désavantage de calculer l'inverse à l'ordre $2d$, et ensuite recalculer le d premiers termes dans le produit $D \times N$. Cependant, il est possible de calculer le numérateur N correspondant à la suite décalée de d : $(U_{i+d})_{i \geqslant 0}$. En effet, la suite ainsi décalée satisfait la même relation de récurrence.

Proposition 2 : (Numérateur décalé)
Pour $i \in \mathbb{N}$, notons $N_i = \overline{C\widehat{U}_i}$ le numérateur correspondant à la suite décalée de id. Cette suite de numérateurs vérifie :

$$\forall i \in \mathbb{N}, N_{i+1} \equiv \frac{N_i - C\widehat{U}_i}{X^d} \equiv -\underline{C\widehat{U}_i} \mod X^d$$

Démonstration : en X^d-adique le théorème 1 modulo X^{2d} appliqué à la suite décalée de id se réécrit :

$$\begin{aligned} C\big(\widehat{U}_i + X^d\widehat{U}_{i+1}\big) &\equiv N_i && \mod X^{2d} \\ C\widehat{U}_{i+1} &\equiv \frac{N_i - C\widehat{U}_i}{X^d} && \mod X^d \end{aligned}$$

Enfin, comme $N_i = \overline{C\widehat{U}_i}$, le calcul de $N_i - C\widehat{U}_i$ revient à annuler les termes bas du produit. □

Algorithme 2 : (Mœnck et Carter)
Nous utilisons le résultat de la proposition 2 pour gagner un facteur constant en complexité par rapport à l'algorithme 1. Il suffit en effet ici de calculer l'inverse modulo d.

Algorithme 2 : RésolutionMoenckEtCarter

Données : d, C et $\widehat{U}_0$
Résultat : $\widehat{U}_1$
début
  $D \leftarrow \text{inverse}(C, X^d)$
  $N \leftarrow (C \times U)[d : 2d]$
  $U \leftarrow D \times N \mod X^d$
  renvoyer $U[d : 2d]$
fin

3.3 High-order lifting

Théorème 3 : (Relation fondamentale)
Pour toute suite matricielle (U_i) récurrente de polynôme générateur C, nous avons en notant $D = \sum \widehat{D}_i X^{id}$ l'inverse de C :

$$\forall j \in \mathbb{N}, \forall \ell \in \mathbb{N},\ \underline{\widehat{D}_{j-1}N_\ell} + \overline{\widehat{D}_j N_\ell} = \widehat{U}_{j+\ell}$$

avec la convention que $\widehat{D}_{-1} = 0$

Remarque : il s'agit là de termes d'ordre $[d, 2d-1]$ du produit $(X^d\widehat{D}_{j-1} + \widehat{D}_j)N_0$, c'est à dire le produit médian que l'on notera $\text{mp}\left(\widehat{D}_{j-1} + X^d\widehat{D}_j, N_0\right) = \underline{\widehat{D}_{j-1}N_0} + \overline{\widehat{D}_j N_0}$

Démonstration : D'après le théorème 1 nous avons $U = DN$ donc en développant

$$\begin{aligned} \sum_{j=0}^{+\infty} \widehat{U}_j X^{dj} &= \sum_{j=0}^{+\infty} \widehat{D}_j N X^{dj} \\ &= \sum_{j=0}^{+\infty} \left(\overline{\widehat{D}_j N} + X^d \underline{\widehat{D}_j N}\right) X^{dj} \end{aligned}$$

Il suffit alors d'identifier les termes X^d-adique pour trouver le résultat annoncé. □

Algorithme 3 : (Avancée de k)
Le théorème 3 nous fournit une relation pour calculer calculer $\widehat{U}_{k+\ell}$ à partir de $\widehat{U}_\ell$ si l'on connait $\widehat{D}_k$ et $\widehat{D}_{k-1}$, nous avons donc un moyen de faire de pas de $+k$ dans la suite.

Algorithme 3 : PasArbitraire

Données : d, C, $\widehat{D}_k, \widehat{D}_{k-1}$ et $\widehat{U}_\ell$
Résultat : $\widehat{U}_{k+\ell}$
début
  $N \leftarrow C \times \widehat{U}_\ell \mod X^d$
  renvoyer $\mathrm{mp}(\widehat{D}_k + X^d \widehat{D}_{k-1},\, N)$
fin

Problème 2 : (Résolution de système en X^d-adique)
Étant donnée une matrice $C \in \mathcal{M}_n(\mathbb{K}[X])$ de degré inférieur à d, une matrice $N \in \mathcal{M}_{n,m}(\mathbb{K}[X])$ de degré inférieur à d, et une précision k, calculer une solution U tel que

$$CU \equiv N \mod X^{dk}$$

Lien avec les suites récurrentes : Nous retrouvons ici une relation similaire à celle du théorème 1. De fait les problèmes 1 et 2 sont deux points de vues sur cette relation, la différence étant que le premier part de C et $\widehat{U}_0$ tandis que le second de C et N.

Comme $N = \overline{C\widehat{U}_0}$ et $\widehat{U}_0 = \overline{\widehat{D}_0 N}$ passer d'un problème à l'autre peut se faire immédiatement dans un sens et après calcul d'inverse dans l'autre.

3.4 Composantes de haut-ordre et résolution

Calcul des composantes de haut ordre : appliquons notre résolution de système à $N_0 = I$ la matrice identité. La solution cherchée est donc $U = D$ l'inverse de C. Le résultat du théorème 3 donne alors :

$$\forall j \in \mathbb{N}, \mathrm{mp}\left(\widehat{D}_{j-1} + X^d \widehat{D}_j, I\right) = \widehat{D}_j$$

Cela nous donne une relation sur les termes de l'inverse de C, toutefois $\widehat{D}_j$ apparait des deux côtés de cette relation. Pour y remédier, nous employons le résultat de la proposition 2 qui permet, en changeant le numérateur, d'obtenir un décalage dans les indices

$$\forall k \in \mathbb{N}, \forall j \in \mathbb{N}, \mathrm{mp}(X^d \widehat{D}_{j-1} + \widehat{D}_j, N_k) = \widehat{D}_{j+k} \tag{1}$$

Les $N_k = \overline{C\widehat{D}_k}$ sont calculables si le $\overline{D}_k$ correspondant est connu, nous obtenons donc une formule qui à partir de $\widehat{D}_j$, $\widehat{D}_{j-1}$, et $\widehat{D}_k$ calcule $\widehat{D}_{j+k}$.

Afin de limiter les données, nous choisissons $k = j$ et $k = j-1$ pour calculer respectivement $\widehat{D}_{2j}$ et $\widehat{D}_{2j-1}$.

Algorithme 4 : (Calcul des composantes de haut ordre)
Grâce aux remarques ci-dessus, nous pouvons calculer les composantes de D d'ordre 2^k et $2^k - 1$, et calculant les deux premiers par inversion et par la méthode employée dans l'algorithme 3.

Algorithme 4 : ComposantesHautOrdre

Données : d, C, et k la précision
Résultat : $(\widehat{D}_{2^j})$ et $(\widehat{D}_{2^j-1})$ pour $j \in [\![0,\, k]\!]$
début
  $\widehat{D}_0 \leftarrow \text{inverse}(C, X^d)$
  $N_1 \leftarrow -(C \times \widehat{D}_0)[d : 2d]$
  $\widehat{D}_1 \leftarrow \widehat{D}_0 \times N_1 \mod X^d$
  pour $j = 1$ **à** k **faire**
    $\widehat{D}_{2^j-1} \leftarrow \text{mp}\big(\widehat{D}_{2^{j-1}} + X^d \widehat{D}_{2^{j-1}-1},\, \widehat{D}_{2^{j-1}-1}\big)$
    $\widehat{D}_{2^j} \leftarrow \text{mp}\big(\widehat{D}_{2^{j-1}} + X^d \widehat{D}_{2^{j-1}-1},\, \widehat{D}_{2^{j-1}}\big)$
  fin
  renvoyer $(\widehat{D}_0,\, \widehat{D}_1),\, \ldots,\, (\widehat{D}_{2^k-1}, \widehat{D}_{2^k})$
fin

Algorithme 5 : (Résolution de système avec second membre borné)
Nous avons à présent tous les outils pour résoudre le problème 2. Il suffit d'appliquer l'algorithme 4 pour calculer les composantes de haut ordre, puis réappliquer le résultat du théorème 3 aux système afin de trouver toutes les composantes.

L'algorithme double les termes connus à chaque itération, et utilise les composantes de haut ordre suivantes pour augmenter le pas. On connait ainsi $\widehat{U}_0$ à l'itération 1, $\widehat{U}_0$ et $\widehat{U}_1$ à l'itération 2, $\widehat{U}_0, \widehat{U}_1, \widehat{U}_2$ et $\widehat{U}_3$ à l'itération 3 et ainsi de suite. La correction résulte de l'application du théorème 3.

Algorithme 5 : ResolutionPetitSecondMembre

Données : d, C, N le second membre et k la précision
Résultat : $U(X)$ tel que $CU \equiv N \mod X^{d2^k}$
début
  $(\widehat{D}_0,\, \widehat{D}_1),\, \ldots,\, (\widehat{D}_{2^k-1}, \widehat{D}_{2^k}) \leftarrow \text{ComposantesHautOrdre}(d, C, k-1)$
  $U \leftarrow D_0 \times N \mod X^d$
  pour $j = 1$ **à** k **faire**
    $N \leftarrow C \times U \mod X^d$ $T \leftarrow \text{mp}\big(\widehat{D}_{2^{j-1}} + X^d \widehat{D}_{2^{j-1}-1},\, N\big)$
    $U \leftarrow [U|T]$ `// concaténation matricielle`
  fin
  renvoyer $U[0] + U[1]X + ... + U[2^k - 1]X^{2^k-1}$
fin

3.5 Second membre non borné

Les résultats précédents s'appuient sur le théorème 3, qui repose sur l'hypothèse d'un second borné en degré par d ($\deg N < \deg d$). Nous allons maintenant voir comment nous pouvons procéder en retirant cette hypothèse.

Travail sur les numérateurs : dans cette section, nous travaillerons plus avec les numérateurs des suites décalées qu'avec les termes de la suite proprement dit. Nous savons que nous pouvons calculer le premier terme d'une suite à partir de son numérateur par la relation de la proposition 2 ($\widehat{U}_0 = \widehat{D}_0 N$). Ainsi, il est facile de retrouver les résultats à partir de tous les numérateurs décalés.

Problème 3 : (Résolution de système à second membre non borné)
Nous cherchons à trouver une solution U telle que

$$CU \equiv N \mod X^{d2^k}$$

pour un C constant en X^d-adique, un k fixé et un N qui s'écrit en X^d-adique : $\sum_{j=0}^{2^k-1} \widehat{N}_j X^{jd}$.
Comme nous résolvons modulo X^{d2^k}, nous pouvons ignorer les termes suivants du numérateur.

Proposition 4 : (Décomposition de la solution)
Soit, pour $j \in [\![0,\, 2^k-1]\!]$, $V^{(j)}$ la solution de

$$CV^{(j)} = \widehat{N}_j$$

Alors, la somme $U = \sum_{j=0}^{2^k-1} V^{(j)} X^{jd}$ est solution du système initial.

Démonstration : ce résultat est clair par linéarité. □

Calcul de numérateurs : nous noterons $\widehat{N}_{j,\ell}$ le numérateur correspondant à la suite décalée $(\widehat{V}^{(j)}_{p+\ell})_{p\geqslant 0}$. D'après les propositions 4 et 2, nous avons pour $p \in [\![0,\, 2^k-1]\!]$

$$\widehat{U}_p = \sum_{j=0}^{p} \widehat{V}^{(j)}_{p-j} = \sum_{j=0}^{p} \overline{D_0 N_{j,p-j}} = \overline{D_0 \sum_{j=0}^{p} N_{j,p-j}}$$

Nous cherchons donc à calculer les sommes partielles $\sum_{j=0}^{2^k-1} N_{j,p-j}$. Nous utilisons un calcul des composantes de haut ordre légèrement différent de celui de l'algorithme 4, pour d'avancer de 2^j les numérateurs et non les termes de la suite.

Proposition 5 : (Pas de 2^k sur les numérateurs)
Ce calcul des termes de haut-ordre facilite la suite, car d'après le théorème 3 et 2, nous avons :

$$\forall j \in \mathbb{N},\ \underline{C\left(\underline{D_{2^j-2} N_\ell} + \overline{D_{2^j-1} N_\ell}\right)} = \underline{C U_{2^j-1+\ell}} = N_{2^j+\ell}$$

d'où une façon simple de décaler le numérateur à partir de cette relation en prenant les parties supérieures du produit par C de la relation fondamentale. Notons la linéarité de cette relation, essentielle pour la suite.

Algorithme 6 : (Composantes de haut ordre 2)
Pour pouvoir utiliser le résultat de la proposition 5, il faut calculer les composantes d'ordre 2^j-2 et 2^j-1 et non 2^j-1 et 2^j comme dans l'algorithme 4. D'où cette version modifiée, qui permet faire $+2^j$ sur les numérateurs.

Algorithme 6 : ComposantesHautOrdre2

Données : d, C, et k la précision
Résultat : $(\widehat{D}_{2^j}-2)$ et $(\widehat{D}_{2^j-1})$ pour $j \in [\![1,\ k]\!]$
début
  $\widehat{D}_0 \leftarrow \text{inverse}(C, X^d)$
  $N_1 \leftarrow -(C \times \widehat{D}_0)[d : 2d]$
  $\widehat{D}_1 \leftarrow \widehat{D}_0 \times N_1 \mod X^d$
  pour $j = 2$ **à** k **faire**
    `// calcul du numérateur par la formule de la proposition 5`
    $N_{2^j-2} \leftarrow \Big(C \times \text{mp}\big(\widehat{D}_{2^{j-1}-1} + X^d\widehat{D}_{2^{j-1}-2},\ \widehat{D}_{2^{j-1}-2}\big)\Big)[d : 2d]$
    $\widehat{D}_{2^j-2} \leftarrow \widehat{D}_0 N_{2^j-2} \mod X^d$
    $N_{2^j-1} \leftarrow \Big(C \times \text{mp}\big(\widehat{D}_{2^{j-1}-1} + X^d\widehat{D}_{2^{j-1}-2},\ \widehat{D}_{2^{j-1}-1}\big)\Big)[d : 2d]$
    $\widehat{D}_{2^j-1} \leftarrow \widehat{D}_0 N_{2^j-1} \mod X^d$
  fin
  renvoyer $\widehat{D}_0,\ \widehat{D}_1,\ \ldots,\ \widehat{D}_{2^k-2}, \widehat{D}_{2^k-1}$
fin

Exemple visuel : prenons le cas $k = 3$, nous avons alors un numérateur en $2^3 = 8$ termes. Nous pouvons représenter les numérateurs à calculer sous forme d'un tableau.

$$\begin{array}{cccccccc}
\widehat{N}_{0,0} & \widehat{N}_{0,1} & \widehat{N}_{0,2} & \widehat{N}_{0,3} & \widehat{N}_{0,4} & \widehat{N}_{0,5} & \widehat{N}_{0,6} & \widehat{N}_{0,7} \\
 & \widehat{N}_{1,0} & \widehat{N}_{1,1} & \widehat{N}_{1,2} & \widehat{N}_{1,3} & \widehat{N}_{1,4} & \widehat{N}_{1,5} & \widehat{N}_{1,6} \\
 & & \widehat{N}_{2,0} & \widehat{N}_{2,1} & \widehat{N}_{2,2} & \widehat{N}_{2,3} & \widehat{N}_{2,4} & \widehat{N}_{2,5} \\
 & & & \widehat{N}_{3,0} & \widehat{N}_{3,1} & \widehat{N}_{3,2} & \widehat{N}_{3,3} & \widehat{N}_{3,4} \\
 & & & & \widehat{N}_{4,0} & \widehat{N}_{4,1} & \widehat{N}_{4,2} & \widehat{N}_{4,3} \\
 & & & & & \widehat{N}_{5,0} & \widehat{N}_{5,1} & \widehat{N}_{5,2} \\
 & & & & & & \widehat{N}_{6,0} & \widehat{N}_{6,1} \\
 & & & & & & & \widehat{N}_{7,0}
\end{array}$$

Dans cette représentation, nous cherchons les numérateurs (sommes de colonnes), et nous savons effectuer un "décalage à gauche", calculer $\widehat{N}_{j,i+2^\ell}$ à partir de $\widehat{N}_{j,i}$ de manière linéaire (nous pouvons donc l'appliquer à plusieurs lignes en même temps). Afin de calculer les termes grisé du tableau efficacement, nous cherchons à faire peu de produits matrices matrices au lieu de plusieurs produits matrices vecteurs. Pour calculer les suivants, nous procédons comme suit :

1. prendre les $2^k - 2^0$ premiers numérateurs, leur appliquer la relation $+2^0$, puis les sommer au $2^k - 2^0$ derniers, les numérateurs bleus sont maintenant connus :

$$\begin{array}{cccccccc}
\widehat{N}_{0,0} & \widehat{N}_{0,1} & \widehat{N}_{0,2} & \widehat{N}_{0,3} & \widehat{N}_{0,4} & \widehat{N}_{0,5} & \widehat{N}_{0,6} & \widehat{N}_{0,7} \\
 & \widehat{N}_{1,0} & \widehat{N}_{1,1} & \widehat{N}_{1,2} & \widehat{N}_{1,3} & \widehat{N}_{1,4} & \widehat{N}_{1,5} & \widehat{N}_{1,6} \\
 & & \widehat{N}_{2,0} & \widehat{N}_{2,1} & \widehat{N}_{2,2} & \widehat{N}_{2,3} & \widehat{N}_{2,4} & \widehat{N}_{2,5} \\
 & & & \widehat{N}_{3,0} & \widehat{N}_{3,1} & \widehat{N}_{3,2} & \widehat{N}_{3,3} & \widehat{N}_{3,4} \\
 & & & & \widehat{N}_{4,0} & \widehat{N}_{4,1} & \widehat{N}_{4,2} & \widehat{N}_{4,3} \\
 & & & & & \widehat{N}_{5,0} & \widehat{N}_{5,1} & \widehat{N}_{5,2} \\
 & & & & & & \widehat{N}_{6,0} & \widehat{N}_{6,1} \\
 & & & & & & & \widehat{N}_{7,0}
\end{array}$$

2. prendre les $2^k - 2^1$ suivants, les décaler de $+2^1$ et les sommer au $2^k - 2^1$ derniers, ce qui donne les termes verts.

$$\begin{array}{cccccccc}
\widehat{N}_{0,0} & \widehat{N}_{0,1} & \widehat{N}_{0,2} & \widehat{N}_{0,3} & \widehat{N}_{0,4} & \widehat{N}_{0,5} & \widehat{N}_{0,6} & \widehat{N}_{0,7} \\
 & \widehat{N}_{1,0} & \widehat{N}_{1,1} & \widehat{N}_{1,2} & \widehat{N}_{1,3} & \widehat{N}_{1,4} & \widehat{N}_{1,5} & \widehat{N}_{1,6} \\
 & & \widehat{N}_{2,0} & \widehat{N}_{2,1} & \widehat{N}_{2,2} & \widehat{N}_{2,3} & \widehat{N}_{2,4} & \widehat{N}_{2,5} \\
 & & & \widehat{N}_{3,0} & \widehat{N}_{3,1} & \widehat{N}_{3,2} & \widehat{N}_{3,3} & \widehat{N}_{3,4} \\
 & & & & \widehat{N}_{4,0} & \widehat{N}_{4,1} & \widehat{N}_{4,2} & \widehat{N}_{4,3} \\
 & & & & & \widehat{N}_{5,0} & \widehat{N}_{5,1} & \widehat{N}_{5,2} \\
 & & & & & & \widehat{N}_{6,0} & \widehat{N}_{6,1} \\
 & & & & & & & \widehat{N}_{7,0}
\end{array}$$

3. prendre les $2^k - 2^2$ suivants, les décaler de $+2^2$ et les sommer au $2^k - 2^2$ derniers, ce qui donne les termes rouges.

$$\begin{array}{cccccccc}
\widehat{N}_{0,0} & \widehat{N}_{0,1} & \widehat{N}_{0,2} & \widehat{N}_{0,3} & \widehat{N}_{0,4} & \widehat{N}_{0,5} & \widehat{N}_{0,6} & \widehat{N}_{0,7} \\
 & \widehat{N}_{1,0} & \widehat{N}_{1,1} & \widehat{N}_{1,2} & \widehat{N}_{1,3} & \widehat{N}_{1,4} & \widehat{N}_{1,5} & \widehat{N}_{1,6} \\
 & & \widehat{N}_{2,0} & \widehat{N}_{2,1} & \widehat{N}_{2,2} & \widehat{N}_{2,3} & \widehat{N}_{2,4} & \widehat{N}_{2,5} \\
 & & & \widehat{N}_{3,0} & \widehat{N}_{3,1} & \widehat{N}_{3,2} & \widehat{N}_{3,3} & \widehat{N}_{3,4} \\
 & & & & \widehat{N}_{4,0} & \widehat{N}_{4,1} & \widehat{N}_{4,2} & \widehat{N}_{4,3} \\
 & & & & & \widehat{N}_{5,0} & \widehat{N}_{5,1} & \widehat{N}_{5,2} \\
 & & & & & & \widehat{N}_{6,0} & \widehat{N}_{6,1} \\
 & & & & & & & \widehat{N}_{7,0}
\end{array}$$

Pour gagner en complexité, nous appliquons les $+2^j$ à une matrice regroupant plusieurs numérateurs (en colonne), ce qui permet de gagner en complexité par rapport à des produits matrice-vecteur par sous linéarité du produit matriciel.

Algorithme 7 : (Résolution de système modulaire, second membre non borné)
Il suffit de généraliser la boucle présentée à un k arbitraire :

Algorithme 7 : Résolution2

Données : d, C, $(\widehat{N}_0, \ldots, \widehat{N}_{2^k-1})$ le second membre et k la précision
Résultat : $U(X)$ tel que $CU \equiv N \mod X^{d2^k}$
début
  $\widehat{D}_0$, $\widehat{D}_1$, …, $\widehat{D}_{2^k-2}$, $\widehat{D}_{2^k-1} \leftarrow$ ComposantesHautOrdre2$(d, C, k-1)$
  $N \leftarrow [\widehat{N}_0 | \ldots | \widehat{N}_{2^k-1}]$
  $\widehat{D}_1 \leftarrow \widehat{D}_0 \times N_1 \mod X^d$
  pour $j = 0$ **à** $k-1$ **faire**
    `// calcul du numérateur par la formule de la proposition 5`
    $T \leftarrow N[0 : 2^k - 2^j]$
    $T \leftarrow C \times \mathrm{mp}\big(\widehat{D}_{2^{j-1}-1} + X^d \widehat{D}_{2^{j-1}-2},\ \widehat{D}_{2^{j-1}-2}\big)$
    $N[2^j : 2^k] \leftarrow N[2^j : 2^k] + T$
  fin
  $U \leftarrow D_0 N \mod X^d$
  renvoyer $U[0] + U[1]X + \ldots + U[2^k - 1]X^{2^k-1}$
fin

Remarque : l'algorithme fonctionne également si l'on renverse l'ordre de la boucle (j décroit de $k-1$ à 0). C'est d'ailleurs cette version décroissante que Storjohann présente dans [6]. La version présentée dans l'algorithme 7 est celle que nous avons élaborée dans le cadre des suites récurrentes. Elle est plus simple a prouver et donne un tableau d'exemple lisible de gauche à droite (pour des opérations équivalentes). Sur notre exemple $k = 3$, l'algorithme décroissant donne les coefficient dans l'ordre suivant (noir - bleu - vert - rouge) :

$$\begin{array}{cccccccc}
\widehat{N}_{0,0} & \widehat{N}_{0,1} & \widehat{N}_{0,2} & \widehat{N}_{0,3} & \widehat{N}_{0,4} & \widehat{N}_{0,5} & \widehat{N}_{0,6} & \widehat{N}_{0,7} \\
 & \widehat{N}_{1,0} & \widehat{N}_{1,1} & \widehat{N}_{1,2} & \widehat{N}_{1,3} & \widehat{N}_{1,4} & \widehat{N}_{1,5} & \widehat{N}_{1,6} \\
 & & \widehat{N}_{2,0} & \widehat{N}_{2,1} & \widehat{N}_{2,2} & \widehat{N}_{2,3} & \widehat{N}_{2,4} & \widehat{N}_{2,5} \\
 & & & \widehat{N}_{3,0} & \widehat{N}_{3,1} & \widehat{N}_{3,2} & \widehat{N}_{3,3} & \widehat{N}_{3,4} \\
 & & & & \widehat{N}_{4,0} & \widehat{N}_{4,1} & \widehat{N}_{4,2} & \widehat{N}_{4,3} \\
 & & & & & \widehat{N}_{5,0} & \widehat{N}_{5,1} & \widehat{N}_{5,2} \\
 & & & & & & \widehat{N}_{6,0} & \widehat{N}_{6,1} \\
 & & & & & & & \widehat{N}_{7,0}
\end{array}$$

Proposition 6 : (Correction)
Cet algorithme est correct.

Démonstration : la preuve se fait par récurrence sur l'indice de la boucle j. Il faut montrer que

$$H(j) : \text{"au début de l'itération } j, \forall s \in [\![0,\ 2^k - 1]\!]\ \mathtt{N}[s] = \sum_{p=0}^{\min(2^j-1,s)} \widehat{N}_{s-p,p}\text{"}$$

avec $\mathtt{N}$ la valeur de la variable N de l'algorithme. Nous Noterons $\mathtt{N}_j$ la valeur de $\mathtt{N}$ au début de l'itération j.

L'initialisation est claire lors de la création de $\mathtt{N}$.

Pour l'hérédité fixons $j \in [\![1,\ k-1]\!]$ et supposons $H(j-1)$ vraie. Alors d'après la proposition 5 et par linéarité, la valeur de $\mathtt{T}$ (qui vaut initialement $\mathtt{N}_{j-1}$ tronqué) après le produit vérifie

$$\forall s \in [\![0,\ 2^k - 2^j]\!]\ , \mathtt{T}[s] = \sum_{p=0}^{\min(2^j-1,s)} \widehat{N}_{s-p,p+2^j} \qquad (\mathtt{N}_{j-1}[s] \text{ décalé de } 2^j)$$

Enfin la dernière ligne ne change que les $2^k - 2^j$ dernières colonnes de $\mathtt{N}$. Les 2^j premières sont déjà correcte par hypothèse de récurrence, s y étant plut petit que $2^j - 1$. Pour $s > 2^j$ nous avons

$$\begin{aligned}
\mathtt{N}_j[s] &= N_{j-1}[s] + T[s - 2^j] \\
&= \sum_{p=0}^{2^j-1} \widehat{N}_{s-p,p} + \sum_{p=0}^{\min(2^j-1,s-2^j)} \widehat{N}_{s-2^j-p,p+2^j} \\
&= \sum_{p=0}^{\min(2^{j+1}-1,s)} \widehat{N}_{s-p,p}
\end{aligned}$$

Nous retrouvons alors l'expression de $\mathtt{N}_{j+1}$, ce qui achève la récurrence. □

4 Résultat de l'implémentation des méthodes de résolution

Expérimentalement (voir figures 3 et 4), notre implémentation ne montre pas directement l'avantage du high-order lifting. Ceci s'explique par le fait que ce gain de complexité provient de l'utilisation de multiplication matricielle efficace, or SageMath n'utilise pas d'algorithmes efficaces pour le produit de matrice polynômiales. Vers la fin du stage nous avons essayer de brancher des produits matriciels fournit par la bibliothèque LinBox dans SageMath en modifiant le code cython afin d'avoir des résultats plus cohérents. Nous n'avons toutefois pas eu le temps de finir.

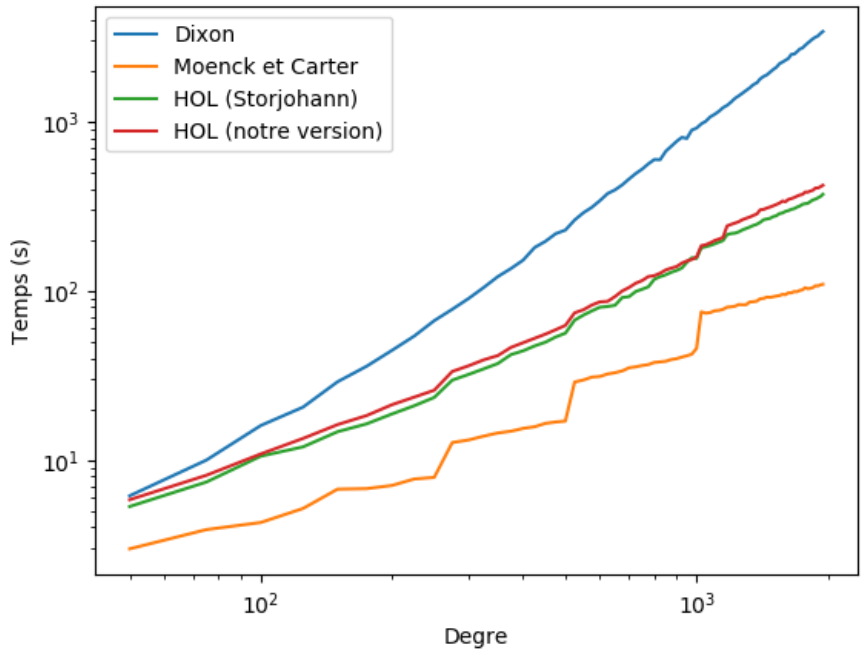

FIGURE 3 – Temps d'exécution des différentes méthodes de résolution modulaire vues pour des systèmes de taille $n = 32$ jusqu'à l'ordre $32d$ en fonction du degré d (échelle logarithmique) sur $\mathbb{F}_{524309}$

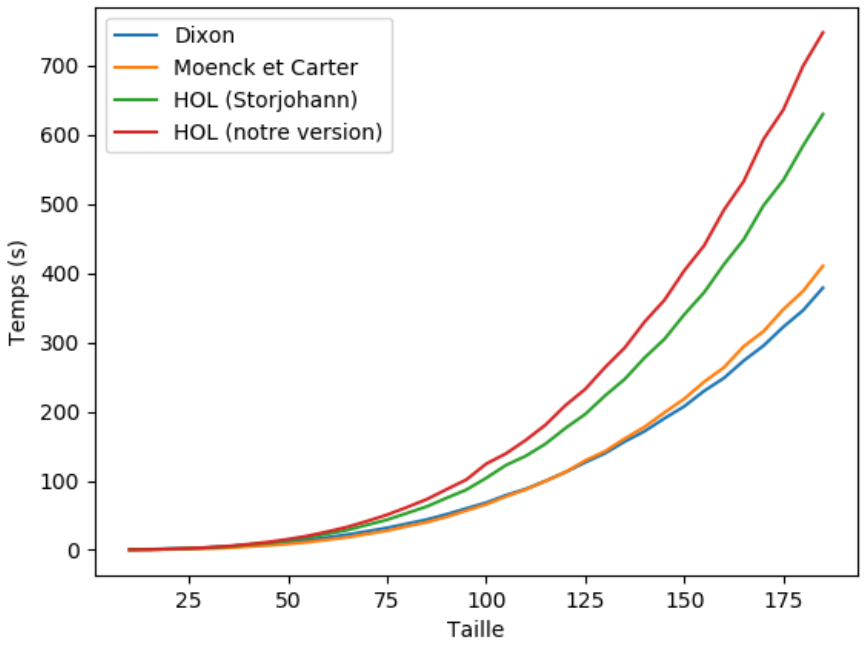

FIGURE 4 – Temps d'exécution des différentes méthodes de résolution modulaire vues pour des systèmes de matrices polynomiales de degré $d = 32$ jusqu'à l'ordre $1024 = 32d$ en fonction de leur taille n sur $\mathbb{F}_{524309}$

5 Conclusion

Ce stage a permis de former une compréhension poussée des différentes méthodes de résolutions vue durant le préstage (dans [1], [5] et [6]), ainsi que de proposer une nouvelle perspective du problème qui est présentée en partie trois, puis de les formaliser pour obtenir une démonstration alternative.

Côté expérimental, j'ai pu coder et vérifier expérimentalement tous les algorithmes de résolution modulaire ainsi qu'un méthode de produit matriciel polynomial par évaluation interpolation en SageMath. Le but était de comparer leur temps d'exécution pour démontrer les avantages du high-order lifting. Toutefois l'absence de multiplication matricielle polynomiale rapide nous a empêché de retrouver les résultats des calculs théoriques de complexité.

Enfin mon travail a permis de mettre à jour et corriger un défaut de vitesse des méthodes de dérivation polynomiales en SageMath, et de proposer un début d'interface permettant d'y rajouter des multiplications de matrices polynomiales rapides.

Remerciements

Je tiens à remercier mes deux encadrants, Romain Lebreton et Pascal Giorgi, pour m'avoir accueilli et suivi tout au long du stage. Je remercie également toute l'équipe ECO pour son accueil durant ces deux mois à Montpellier. Enfin je souhaite remercier Jérémy Berthomieu qui m'a encadré lors de mon mémoire pré-stage à Paris.