Sorbonne Université

Des polynômes pour les robots

Amel CHETTI et Inas KACI encadrées par Mohab SAFEY EL DIN

10 septembre 2019

Contents

Rapport de Projet: Des polynômes pour des robots

Amel Chetti et Inas Kaci

September 10, 2019

1 Introduction

Dans le cadre de l'UE 2I013, nous avons travaillé sur le sujet: Des polynômes pour des robots. Un robot est vu mathématiquement comme une application de $\mathbb{R}^n$ dans $\mathbb{R}^n$. L'espace de configuration qui encode l'ensemble des positions et mouvements possibles de celui-ci est un sous-ensemble de l'ensemble d'arrivée de cette application. Des algorithmes de calcul formel ont été produits pour permettre d'étudier les singularités de cette application. Ce qui revient finalement à étudier des systèmes d'équations polynomiales à deux variables de degrés et coefficients relativement grands. Ce qui nous amène à la nécessité d'utiliser des outils mathématiques et des paradigmes du calcul haute performance afin de réduire les temps de calcul.

Pour notre projet, nous nous sommes intéressées au calcul du résultant de deux polynômes à deux variables. De ce fait, dans un corps K algébriquement clos, si P et Q sont deux polynômes homogènes de degrés n, m respectivement sans facteur irréductible en commun i.e $PGCD(P, Q) = 1$ alors les deux courbes projectives C_p et C_q n'ont qu'un nombre fini de points en commun $p_1, ..., p_k$. Le cas le plus simple est l'intersection d'une courbe affine plane C_1 d'équation $F(X, Y) = 0$, où $F \in K[X, Y]$ a un degré d_1, avec une courbe affine plane C_2 dont l'équation est de la forme $Y = Q(X)$ où $Q \in K[X]$ a un degré d_2. On trouve les coordonnées des points d'intersection en substituant $Q(X)$ à Y dans l'équation de C_1 et en résolvant $F(X, Q(X)) = 0$. Le rôle du résultant est d'éliminer la variable Y même quand l'équation de C_2 n'est pas de la forme $Y = Q(X)$.

Notre démarche consiste en premier lieu en une familiarisation avec les bibliothèques C: FLINT qui gère la théorie des nombres et prend en charge l'arithmétique avec des nombres, des polynômes, des séries de puissances, des matrices sur plusieurs anneaux de base; GMP qui est pour l'arithmétique en précision arbitraire, fonctionne sur des nombres signés, des nombres rationnels et des nombres à virgule flottante. Cette bibliothèque fait de la mémoire disponible sur la machine d'exécution des programmes la seule limite pratique à la précision; OMP qui est est une interface de programmation pour le calcul parallèle sur architecture à mémoire partagée. Cela dans le but de pouvoir manipuler des polynômes avec des coefficients étant des entiers multiprécision ou des entiers modulo n (opérations arithmétiques sur les polynômes, écriture et lecture de polynômes dans un fichier, génération aléatoire de polynômes, comparaison de performance des différentes fonctions de multiplication de polynômes présentes dans FLINT) .

Puis en deuxième lieu, à l'étude des différents algorithmes de calcul du résultant de deux polynômes. C'est à dire faire le calcul du déterminant de la matrice de Sylvester associée : **L'algorithme de Gauss** qui consiste essentiellement à faire des opérations d'addition et soustraction de lignes ou colonnes d'une matrice afin de la trianguliariser puis faire un simple produit des éléments de la diagonale. **L'algorithme Big prime** qui consiste à faire une réduction de la matrice modulo un entier n qu'on va déterminer, d'appliquer l'algorithme de Gauss sur la matrice obtenue puis de reconstruire la solution finale à partir de la solution trouvée dans l'anneau Z/nZ. **L'algorithme small primes** qui consiste à calculer des images de notre matrice chacune modulo un nombre n_i différent, puis leur appliquer l'algorithme de Gauss pour calculer le résultant dans chaque anneau Z/n_iZ. Ces calculs peuvent être effectués en parallèle. Le résultat final est ensuite retrouvé grâce au théorème des restes chinois Après une étude de complexité, il s'est avéré que le deuxième algorithme est plus rapide que le premier. L'avantage de ce dernier est qu'il nous évite un problème de croissance des calculs intermédiaires rencontré avec le premier. Le troisième algorithme reste le plus performant en temps de calcul.

En troisième lieu, On implémente notre stratégie multimodulaire avec le deuxième algorithme. Étant donné que nous manipulons des polynômes à deux variables, on détermine un nombre nb fini de points sur lesquels on évalue les deux polynômes pour se retrouver à chaque fois avec des polynômes univariés. On calcule le résultant pour chaque évaluation puis on construit le résultant des deux polynômes bivariés par interpolation. On procède à une partie expérimentale pour montrer l'exactitude de notre étude théorique avec une comparaison entre les temps de calculs de notre stratégie multimodulaire pour le calcul du résultant et la fonction résultant de FLINT.

2 Librairies

2.1 La bibliothèque arithmétique de précision multiple de GNU (GMP: GNU Multiple Precision Arithmetic Library)

GMP est une bibliothèque libre pour l'arithmétique de précision arbitraire, fonctionnant sur des entiers signés, des nombres rationnels et des nombres à virgule flottante. Il n'y a pas de limite pratique à la précision, à l'exception de celles impliquées par la mémoire disponible dans la machine sur laquelle GMP s'exécute (la taille de l'opérande est limitée à 2^{32} -1 bits sur les machines 32 bits et à 2^{37} bits sur les machines 64 bits). GMP dispose d'un riche ensemble de fonctions, dont l'interface est régulière. La bibliothèque GMP est une bibliothèque qui va nous permettre de gérer de très très grands nombres, et d'effectuer tout un tas de calculs rapidement car un "double" ou un "float" ne peut pas suffire pour gérer ces grands nombres.
Le type "double", le plus grand des types primitifs, a pour extréma -1.7∗10 ^{308} et $1.7*10^{308}$. Prenons l'exemple 386^{4279}: le résultat dépasse largement le type "double". En plus des opérations arithmétiques de haut niveau sur les nombres flottants, elle contient une classe de fonctions de bas niveau qui traitent les entiers naturels codés dans des tableaux de mots machines (limb dans la terminologie de Gmp). Sur les processeurs MIPS un mot machine est composé de 64bits, un bloc peut être de longueur 32.

Pour des raisons d'efficacité, la majorité de ces fonctions sont écrites en assembleur (par exemple sur les plate-formes x86).

2.2 Flint

FLINT est une bibliothèque C pour la théorie des nombres, elle prend en charge l'arithmétique avec des nombres, des polynômes, des séries entières et des matrices sur de nombreux anneaux de base, notamment:

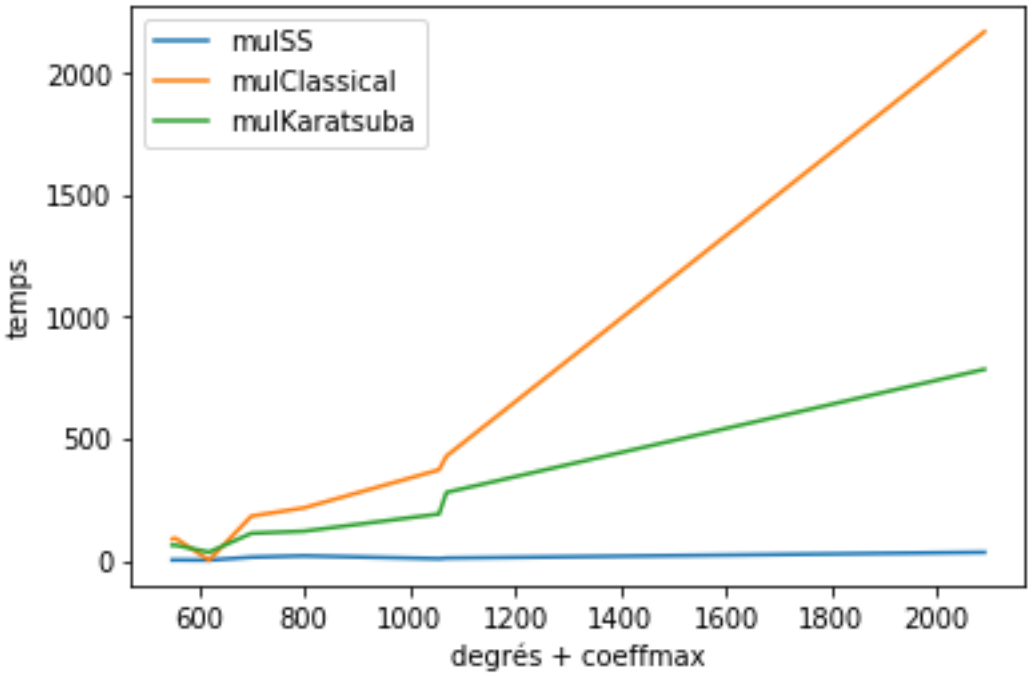
1. Entiers multiprécisions et rationnels.
2. Entiers modulo n.

FLINT dépend des bibliothèques MPIR / GMP et MPFR, est utilisé comme package par défaut pour l'arithmétique polynomiale sur $\mathbb{Z}, \mathbb{Q}$ et $\mathbb{Z}/n\mathbb{Z}$ pour les petits n.

2.2.1 Fonctions de FLINT

FLINT est une collection de modules, avec une interface similaire à celle de la bibliothèque GMP. Par exemple, le module FLINT $fmpz\_poly$ définit le type $fmpz\_poly\_t$ (représentant un polynôme avec des coefficients entiers multi-précision) et fournit des méthodes telles que $fmpz_poly$_add, $fmpz$_poly$_gcd, etc.$

Un exemple de test effectué sur des fonctions de multiplication de FLINT afin d'évaluer leurs performances.



On constate que la fonction MulSS est plus performante que MulKaratsuba et cette dernière est plus performante que MulClassical.

2.2.2 Objets de FLINT

- fmpz(Entiers Multi-Précision): pointe directement sur une structure combinée.

  1. Représentation du pointeur balisé, avec des nombres entiers allant jusqu'à 30 ou 62 bits, utilisant un seul mot et ne nécessitant pas d'allocation de tas.
  2. Met en cache des entiers plus grands pour une allocation et une désallocation rapide.
  3. Très efficace pour les petits entiers et performances similaires au type mpz pour les grands nombres.
  4. un fmpz est implémenté en tant que slong. Quand son deuxième bit le plus significatif est 0, le fmpz représente un entier long ordinaire. Lorsque le deuxième bit le plus significatif est 1, la valeur représente un pointeur (le pointeur est décalé à droite de 2 bits et le deuxième msb est défini à 1 . Cela repose sur le fait que malloc alloue toujours de la mémoire sur une limite de 4 ou 8 octets).

- fmpz_t : Un tableau de longueur 1 de fmpz. Ceci est utilisé pour permettre le passage par référence.

  La Structure fmpz_t:
  typedef slong fmpz;
  typedef fmpz fmpz_t[1];

- fmpz_poly_t: Le type de données fmpz_poly_t représente des éléments de Z [x].
  FLINT utilise une représentation dense de polynômes. Cela signifie que le coefficient de chaque degré du polynôme est stocké dans un vecteur. Cela contraste avec une approche fragmentée qui ne stocke que les coefficients non nuls d'un polynôme dans un dictionnaire. Flint utilise également sa propre classe d'entiers comme type de coefficient. Les entiers de Flint sont fondamentalement similaires aux entiers GMP et respectent les mêmes conventions de nommage. (par exemple,

fmpz_t est analogue à mpz_t). Ils ont des optimisations internes pour les rendre plus rapides à utiliser dans la bibliothèque FLINT elle -même. En effet, chaque coefficient d'un fmpz_poly_t est un entier du type fmpz_t. Ce modèle présente deux avantages.

1. le type fmpz_t est géré par la mémoire, de sorte que l'utilisateur peut manipuler les coefficients individuels d'un polynôme sans avoir à gérer de fastidieuse gestion de la mémoire.
2. un coefficient d'un fmpz_poly_t peut être modifié sans changer la taille d'aucun des autres coefficients.

- mp_limb_t: signifie la partie d'un nombre multi−précision qui tient dans un seul mot machine. "Limb" est composé de 32 ou 64 bits. Le type de données C pour un "limb"est mp_limb_t.

3 Objets mathématiques

3.1 Objets mathématiques

3.1.1 définitions

Polynôme à une variable: Expression algébrique constituée par une somme algébrique de monômes qui sont des expressions algébriques ne comportant qu'un seul terme (séparés par les signes + ou –).
exemple: $3x^2 + x^6 + 5$, x, 1 ...

Implémentation:
FLINT utilise une représentation dense pour chaque polynôme, c'est à dire que c'est un tableau tab de coefficients tab[i] des monômes de degré i. La bibliothèque FLINT nous fournit le type de données fmpz_poly_t pour représenter les éléments de $Z[X]$ dont les coefficients sont des entiers de type fmpz_t.

Polynôme à plusieurs variables: (à coefficients dans un anneaux commutatif unitaire A) est un élément d'une A-algèbre associative qui généralise l'algèbre A[X] des polynômes en une variable X.

Implémentation:

Afin de générer un polynôme à deux variables. On implémente un tableau de polynômes à une variable X. on a donc un polynôme de variable Y dont les coefficients sont des polynômes de variable X. On a donc bien nos deux variables.

Anneau:
Un triplet $(A, +, \times)$ formé d'un ensemble A muni de deux lois de composition internes sur A. une addition + et une multiplication $\times$ tel que les conditions suivantes soient vérifiées.

- (A, +) est un groupe commutatif.
- la loi $\times$ est distributive sur l'addition.
- la multiplication est associative et possède un élément neutre.
- Si de plus la multiplication est commutative alors l'anneau est commutatif.

Exemples:
- L'anneau Z muni des deux lois de composition usuelles (multiplication et addition). Il est commutatif.
- L'anneau $F[X, A]$ l'ensemble des applications de X à valeurs dans A
- L'anneau $A[X]$ des polynômes à coefficients dans l'anneau A.

Soit A un anneau et B une partie de A.

Sous-anneau: On dit que B est un idéal de A si:
- B est sous-groupe additif de A
- Quels que soient x et y dans B, xy est dans B.
- 1 est un élément neutre multiplicatif appartenant à B

Exemples:
- Z est un sous-anneau de R
- L'ensemble des fonctions continues de R dans R est un sous-anneau de $F(R, R)$.

Idéal: Supposons A commutatif. On dit que B est un idéal de A si:
- B est un sous-groupe additif de A.
- Quelque soit x dans A et y dans B, xy est dans B.

Exemples:
- Les idéaux de Z sont les nZ ou n parcours Z ou N
- L'ensemble Z n'est pas un idéal de Q et R

Anneau quotient: Considérons un anneau commutatif A et un idéal I. L'ensemble A/I muni de l'addition et la multiplication définies par les formules:

$$(x + I) + (y + I) = (x + y) + I \tag{1}$$

$$(x + I)(y + I) = xy + I$$

est un anneau commutatif. On l'appelle l'anneau quotient de A par I.

Exemples:
L'anneau quotient Z/nZ:
Soit n un entier naturel non nul, nZ étant un idéal de Z. l'ensemble Z/nZ est donc muni d'une structure d'anneau commutatif, pour laquelle l'addition et la multiplication sont données par les égalités:

$$\overline{a} + \overline{b} = \overline{a + b} \tag{2}$$

$$\overline{a} * \overline{b} = \overline{ab} \tag{3}$$

L'élément neutre additif est $\overline{0}$ et l'élément neutre multiplicatif est $\overline{1}$. Il est appelé l'anneau des entiers modulo n.

Propriétés:
- Inverse modulaire: Un élément a de Z/nZ est inversible si et seulement si a est premier avec n. C'est à dire que $pgcd(a, n) = 1$. Pour le trouver on peut utiliser l'algorithme d'Euclide étendu qui repose sur le théorème de Bézout.

$$\exists u, v \in \mathbb{Z}, u * a + b * n = 1 \tag{4}$$

Exemples:

- 2 n'admet pas d'inverse dans $Z/6Z$ car $pgcd(2,6) = 2$
- $pgcd(49,2) = 1$ donc 2 admet un inverse dans $Z/49Z$, on a:

$$49 - 2 * 28 = 1 \tag{5}$$

donc l'inverse est -28 modulo 49 donc 21
Autrement dit, c'est l'ensemble des entiers a tels que n divise $a - 1$.

Polynômes dans Z/nZ:

C'est un polynôme à coefficients dans Z/nZ.
implémentation:
Pour implémenter nos polynômes Z/nZ, On utilise dans la librairie FLINT nmod_poly_t qui est un type de données des éléments de $Z/nZ[X]$. Chaque coefficient est de type mp−limb−t qui représente un entier réduit modulo n. C'est une liste de longueur 1 de nmod polystruct. Ce qui permet de passer des paramètres de type nmod_poly_t par adresse.

Déterminant: Soit E un K-espace vectoriel de dimension n, B une base de E et $(v_1, v_2, ..., v_n)$ une famille de n vecteurs de E. Pour chaque $i, j = 1, ..., n$ on note v_{ij} le $(j - i)$ème coefficient de v_i décomposé dans la base B. On appelle déterminant de $(v_1, ..., v_n)$ dans la base B, le scalaire:

$$det\colon E \to K$$
$$(v_1, ..., v_n) \mapsto \sum_{\sigma \in Sn} \varepsilon(\sigma) \prod_{i=0}^{n-1} a_{\sigma(i),i}$$

avec $\sigma \in Sn$ l'ensemble des permutations de n éléments et $\varepsilon(\sigma)$ la signature(parité de décomposition en transpositions: échange de deux éléments 1 si paire et -1 si impaire).
- det est une application n-linéaire antisymétrique (alternée) .

- n-lineaire: $det(v_1, . + av_i + bu_i .., v_n) = a \times det(v_1, ..v_i ..., v_n) + b \times det(v_1 ...u_i ...v_n)$.
- $alternée$: $det(v_1, ...v_i ...v_j ...v_n) + det(v_1, ...v_j ...v_i ..., v_n) = 0$
- $det(AB) = det(A) \times det(B)$
- $det(\lambda A) = \lambda^n det(A)$
- $det(A) = det(A^t)$
- pour n=2 $det(A) = a_{1,1}a_{2,2} - a_{2,1}a_{1,2}$
  Exemple: pour A la matrice suivante: $\begin{pmatrix} 1 & 4 \\ 2 & 1 \end{pmatrix}$ $det(A) = 1 \times 1 - 2 \times 4 = -7$

Calcul du déterminant grâce à la notion de cofacteur:
théoreme: si A une matrice de taille $n \times n$. On définit le mineur $A_{i,j}$ de A comme la matrice de dimension $(n - 1) \times (n - 1)$ issue de la matrice A privée de sa i^{ieme} ligne et j^{ieme} colonne. Soit $k \in \{ 0,..., n \}$.

$$det(A) = \sum_{i=0}^{n-1} (-1)^{i+k} a_{i,k} det(A_{i,k}) = \sum_{j=0}^{n-1} (-1)^{i+k} a_{k,j} det(A_{k,j}) \tag{6}$$

$(-1)^{i+j} a_{i,j} det(A_{i,j})$ est appelé cofacteur de $a_{i,j}$ Ceci nous donne donc l'algorithme élémentaire pour le calcul du déterminant d'une matrice.

Exemple: considérons la matrice de taille 4×4 suivante:

$$\begin{pmatrix} 1 & 4 & 2 & 5 \\ 2 & 1 & 7 & 3 \\ 1 & 2 & 2 & 1 \\ 1 & 2 & 1 & 1 \end{pmatrix}$$

$$\det \begin{pmatrix} 1 & 4 & 2 & 5 \\ 2 & 1 & 7 & 3 \\ 1 & 2 & 2 & 1 \\ 1 & 2 & 1 & 1 \end{pmatrix} = 1 \det \begin{pmatrix} 1 & 7 & 3 \\ 2 & 2 & 1 \\ 2 & 1 & 1 \end{pmatrix} - 2 \det \begin{pmatrix} 4 & 2 & 5 \\ 2 & 2 & 1 \\ 2 & 1 & 1 \end{pmatrix} + 1 \det \begin{pmatrix} 4 & 2 & 5 \\ 1 & 7 & 3 \\ 2 & 1 & 1 \end{pmatrix} - 1 \det \begin{pmatrix} 4 & 2 & 5 \\ 1 & 7 & 3 \\ 2 & 2 & 1 \end{pmatrix}$$

$$= 1 * (1 \det \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix} - 2 \det \begin{pmatrix} 7 & 3 \\ 1 & 1 \end{pmatrix} + 2 \det \begin{pmatrix} 7 & 3 \\ 2 & 1 \end{pmatrix}) - 2 * (4 \det \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix} - 2 \det \begin{pmatrix} 2 & 5 \\ 1 & 1 \end{pmatrix} + 2 \det \begin{pmatrix} 2 & 5 \\ 2 & 1 \end{pmatrix}) + 1 * (4 \det \begin{pmatrix} 7 & 3 \\ 1 & 1 \end{pmatrix} - 1 \det \begin{pmatrix} 2 & 5 \\ 1 & 1 \end{pmatrix} + 2 \det \begin{pmatrix} 2 & 5 \\ 7 & 3 \end{pmatrix}) - 1 * (4 \det \begin{pmatrix} 7 & 3 \\ 2 & 1 \end{pmatrix} - 1 \det \begin{pmatrix} 2 & 5 \\ 2 & 1 \end{pmatrix} + 2 \det \begin{pmatrix} 2 & 5 \\ 7 & 3 \end{pmatrix})$$

$$= 1 \times (1 \times (2 \times 1 - 1 \times 1) - 2 \times (7 \times 1 - 1 \times 3) + 2 \times (7 \times 1 - 2 \times 3)) - 2 \times (4 \times (2 \times 1 - 1 \times 1) - 2 \times (2 \times 1 - 1 \times 5) + 2 \times (2 \times 1 - 2 \times 5)) + 1 \times (4 \times (7 \times 1 - 1 \times 3) - 1 \times (2 \times 1 - 1 \times 5) + 2 \times (2 \times 3 - 7 \times 5)) - 1 \times (4 \times (7 \times 1 - 2 \times 3) - 1 \times (2 \times 1 - 2 \times 5) + 2 \times (2 \times 3 - 7 \times 5)) = 14$$

Complexité :
Si on considère TDM(n) le nombre de produits nécessaires pour le calcul du déterminant d'une matrice de taille n $TDM(n) = n \times TDM(n-1) + n(7)$.
On en déduit que TDM(n)=O(n!) , la complexité est identique à celle via les permutations. Elle est grande, nous présenterons des méthodes de calcul plus optimisées dans la partie étude algorithmique.

3.1.2 Résultant

Dans la suite nous considérerons deux polynômes P et Q de degrés respectivement n et m:

$$P(x) = \sum_{i=0}^{n} a_i x^i; \tag{8}$$

$$Q(x) = \sum_{i=0}^{m} b_i x^i \tag{9}$$

Matrice de Sylvester:

La matrice de Sylvester associée à P et Q notée $S_{p,q}$ est une matrice carrée de taille $(m+n) \times (m+n)$ de la forme suivante:

$$\begin{pmatrix} a_0 & 0 & \dots & 0 & b_0 & 0 & \dots \\ a_1 & a_0 & \ddots & 0 & b_1 & b_0 & \ddots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \ddots & a_0 & b_m & \vdots & \vdots \\ a_n & \dots & \ddots & \ddots & \vdots & 0 & \vdots \\ 0 & a_n & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & a_n & 0 & \cdots & b_m \end{pmatrix}$$

La première colonne est formée des coefficients de P suivis de zéros, la seconde à partir de la première par permutation circulaire vers le bas. On répète ainsi de suite la même opération pour l'ensemble des n-2 colonnes suivantes. La colonne n+1 est formée des coefficients de Q suivi de 0. Les m-1 colonnes suivantes sont formées de la même façon par permutation circulaire.

Exemple: Pour $P(x) = 6x^3 + x + 1$ et $Q(x) = 2x^3 + x^2$ la matrice de Sylvester correspondante est:

A= $\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 6 & 0 & 1 & 2 & 1 & 0 \\ 0 & 6 & 0 & 0 & 2 & 1 \\ 0 & 0 & 6 & 0 & 0 & 2 \end{pmatrix}$

Définition du résultant:

Le résultant de P et Q noté res(P,Q) est le déterminant de la matrice de Sylvester.

Exemples: Grâce à la méthode de calcul du déterminant d'une matrice définie précédemment, pour $P(x) = 6x^3 + x + 1$ et $Q(x) = 2x^3 + x^2$

$$Res(P,Q) = det \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 6 & 0 & 1 & 2 & 1 & 0 \\ 0 & 6 & 0 & 0 & 2 & 1 \\ 0 & 0 & 6 & 0 & 0 & 2 \end{pmatrix} = -2$$

Propriétés:

- Si P et Q sont des polynômes dans un corps K
  alors ils ont un diviseur commun si: $res(P,Q) = 0$.

  si en plus P est scindé de racines $a_1, a_2, ..., a_n$, alors on a la formule suivante:

$$Res(P,Q) = a_p^q Q(a_1), ..., Q(a_p) \tag{10}$$

- $\forall \nu, \mu \in R(\nu P, \mu Q) = \nu^n \mu^m R(P,Q)$
- $R(P,Q) = (-1)^{mn} R(Q,P)$
- $deg(PGCD(P,Q)) = m + n - rang(S_{p,q})$

3.1.3 Théorème des restes chinois

Soient m_1, m_2, ... , m_n des entiers supérieurs à 2 et deux à deux premiers entre eux, alors pour tout entier a_1, a_2, ... , a_n, on a le système d'équations: $\begin{cases} x \equiv a_1 \quad [m_1] \\ x \equiv a_2 \quad [m_2] \\ \qquad\qquad . \\ \qquad\qquad . \\ \qquad\qquad . \\ x \equiv a_n \quad [m_n] \end{cases}$

Première méthode

Posons:

$$M = m_1 \times m_2 \times ... \times m_n$$

,

$$M_i = M/m_i \quad \forall i \in 1, 2, ..., n$$

Il admet une unique solution modulo M donnée par la formule suivante:

$$x = \sum_{i=0}^{n} a_i \times M_i \times Y_i \quad avec \quad Y_i \equiv M_i^{-1} \ [m_i]$$

Seconde méthode

Une seconde méthode pour résoudre un système de congruence comme énoncé précédemment est de remplacer les deux premières équations:
$\begin{cases} x \equiv a_1 \quad [m_1] \\ x \equiv a_2 \quad [m_2] \end{cases}$
par une unique équation :
$x \equiv b \quad [m_1 \times m_2]$
(On suppose toujours que les m_i sont premiers deux à deux).
Pour ce faire, on écrit:
$\begin{cases} x = a_1 + m_1 u \\ x = a_2 + m_2 t \end{cases}$
où u et t sont des entiers qui doivent satisfaire:
$a_1 + m_1 u = a_2 + m_2 t$ où *encore* $m_1 u - m_2 t =$a$_2 - a_1$.
Cette équation a une solution si et seulement si m1,m2 divisent $a_2 - a_1$. Elle se résout à l'aide de l'identité de Bézout.
Comme $x = m_1 u + a_1 = b$.
On peut maintenant remplacer les deux premières équations par l'unique équation:
$x \equiv b \quad [m_1 \times m_2]$, et continuer de manière récursive jusqu'à ce qu'il ne reste qu'une seule équation.

Algorithme
Entrées :
m_1, m_2, ... , m_n sont premiers deux à deux
a_1, a_2, ... , a_n
Sortie: x résolvant le système de congruence
Corps: $M \leftarrow m_1$
$x \leftarrow a - 1$
Pour $i \in \{1, 2, ..., n\}$ faire: Euclide étendu: calculer (u, v) tel que:
$um_i + vM = 1$
$x \leftarrow um_i x + vMa_i$
$M \leftarrow m_i M$
$x \leftarrow xmodM$
Exemple: Résolvant le système:
$\begin{cases} x \equiv 3 \quad [11] \\ x \equiv 6 \quad [8] \\ x \equiv -1 \quad [15] \end{cases}$
par cette seconde méthode.On commence par résoudre:
$\begin{cases} x \equiv 3 \quad [11] \\ x \equiv 6 \quad [8] \end{cases}$ en cherchant t et u tels que:

$3 + 11t = 6 + 8u \quad ou \quad 11t - 8u = 3$
On peut résoudre ceci par l'algorithme d'Euclide et par l'identité de Bézout mais ici, on constate que t=u=1 est une solution.
Donc $x \equiv 14 \quad [88]$ est une solution des deux premières équations. On remplace ces deux équations par cette dernière et le système devient:
$\begin{cases} x \equiv 14 \quad [88] \\ x \equiv -1 \quad [15] \end{cases}$
On résout ce dernier système de la même manière en cherchant t et u tels que:
$14 + 88t = -1 + 15u$
On peut prendre t=0 et u=1 ce qui nous donne la solution finale:
$x \equiv 14 \quad [1320]$.

4 Étude algorithmique:

Problème Calcul du résultant de deux polynômes P et Q. Ce qui revient au calcul du déterminant de la matrice de Sylvester associée. Comme vu dans la partie déterminant la méthode des cofacteurs présente une complexité très grande. On va donc étudier plusieurs algorithmes dans le but d'optimiser le temps de calcul. On a présenté dans l'introduction le problème sur des polynômes à deux variables.On peut les voir comme des polynômes à une variable principale dont les coefficients sont des polynômes avec la variable secondaire. Une application sur notre algorithme serait d'avoir des matrices dont les termes sont des polynômes.Le déterminant qui résultera sera donc un polynôme à une seule variable. Dans le but de simplifier les explications des algorithmes, nous allons continuer à utiliser l'exemple précèdent présenté pour la matrice de Sylvester.

4.1 Algorithme 1: L'algorithme de Gauss

La première méthode qu'on peut utiliser vient de l'algèbre linéaire. Elle consiste à triangulariser notre matrice en premier lieu. En suivant l'algorithme suivant:
Entrée:
A: la matrice de Sylvester correspondante
n: dimension de la matrice
Corps:
Pour i=0 à n-1 faire
: Pour k=i+1 à n-1 faire
: Pour j=i+1 à n-1 faire
: $a_{k,j} \leftarrow a_{k,j} - (a_{k,i}/a_{i,i}) * a_{i,j}$ . . .(*)
Sortie: retourne A
D'après le théorème des cofacteurs, on peut déduire que le déterminant correspond au produit de la diagonale d'une matrice triangulaire. Mais cet algorithme oblige que les éléments de la diagonale soient différents de 0. C'est à dire qu'on considère qu'on a pas besoin de faire de permutations durant l'execution de l'algorithme de Gauss. Le deuxième problème qui se pose est qu'avec cet algorithme, on est confronté au problème de croissance des calculs intermédiaires (Intermediate Expression Swell).
Exemple: On applique notre algorithme à la matrice A précédente:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 6 & 0 & 1 & 2 & 1 & 0 \\ 0 & 6 & 0 & 0 & 2 & 1 \\ 0 & 0 & 6 & 0 & 0 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 \\ 0 & 6 & 0 & 0 & 2 & 1 \\ 0 & 0 & 6 & 0 & 0 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 6 & 0 & 0 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & -6 & 0 & 2 \end{pmatrix}$$

$$\rightarrow \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 6 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix}$$

En faisant le produit des termes de la diagonale on retrouve bien $det(A) = -2$

Définition:
Mot machine: est une unité de base manipulée par un microprocesseur. Sa taille est utilisée pour classer les microprocesseurs (32 bits, 64 bits...). Plus le mot est long, plus le microprocesseur est rapide car il traite plus de données à chaque cycle.

Complexité: $\frac{n(n-1)(2n-1)}{6} + \frac{n(n-1)}{2}$, on peut donc dire que c'est du $O(n^3)$. C'est donc une complexité polynomiale par rapport à la taille des entrées dans le cas où les coefficients de notre matrice tiennent en un mot machine. Cependant, le nombre d'opérations mot machine que l'algorithme utilise va aussi dépendre des dénominateurs et numérateurs des résultats intermédiaires. On cherche donc à connaître la borne supérieure de ces valeurs intermédiaires.
On considère la kème étape de l'élimination gaussienne. Si b_k est la borne supérieure de la valeur absolue du numérateur et du dénominateur de tout les $a_{i,j}^{(k)}$ pour $1 < i, j < n$, tel que en particulier $|a_{i,j}| \leq b_1$. D'après (*), on a alors:

$$b_k \leq 2b_{k-1}^4 \leq 2^{1+4} b_{k-2}^{4^2} \leq ... \leq 2^{1+4+...+4^{k-2}} b_1^{4^{k-1}} = 2^{(4^{k-1}-1)/3} b_1^{4^{k-1}} \tag{11}$$

Cette borne supérieure est exponentiellement grande dans la taille d'entrée. Le temps polynomial de l'algorithme de Gauss est remis en question car la longueur des valeurs intermédiaires et le nombre d'opérations sur les mots pour l'élimination gaussienne sur $\mathbb{Q}$ est polynomial dans la taille d'entrée. (La preuve est non triviale). D'autant plus que comme présenté dans notre introduction, nous manipulons des polynômes à deux variables dont les coefficients sont relativement grands.

4.2 Algorithme 2: Big prime

Dans le but de se débarrasser du problème de croissance des valeurs intermédiaires, on utilise une approche alternative pour atteindre le même but mais en gardant une complexité polynomiale. On passe à un calcul modulaire, de ce fait nos valeurs seront toujours représentables sur un mot machine.

Définition:
Intermediate Expression Swell: Croissance des calculs intermédiaires: C'est un phénomène qui survient lorsque les coefficients dans un algorithme grandissent au court de l'exécution de celui-ci. En d'autres termes, les entiers apparaissant au cours des calculs sont de très grandes tailles par rapport à ceux qui figurent à la fois dans l'entrée et dans la sortie de l'algorithme.
Exemple: L'algorithme d'Euclide (Les coefficients grandissent au plus avec un facteur de 3 en une division.
algorithme: Soit F un groupe, $f, g \in F[X]$ avec $degf = n \geq degg = m$

$$\begin{array}{lll} p_0 r_0 = f & p_0 s_0 = 1 & p_0 t_0 = 0 \\ p_1 r_1 = g & p_1 s_1 = 0 & p_1 t_1 = 1 \\ p_2 r_2 = r_0 - q_1 r_1 & p_2 s_2 = s_0 - q_1 s_1 & p_2 t_2 = t_0 - q_1 t_1 \\ \vdots & \vdots & \vdots \\ p_{i+1} r_{i+1} = r_{i-1} - q_i r_i & p_{i+1} s_{i+1} = s_{i-1} - q_i s_i & p_{i+1} t_{i+1} = t_{i-1} - q_i t_i \\ \vdots & \vdots & \vdots \\ 0 = r_{l-1} - q_l rl & s_{l+1} = s_{l-1} - q_l s_l & t_{l+1} = t_{l-1} - q_l t_l \end{array}$$

application numérique: illustration de la croissance des coefficients intermédiaires:
f= $824x^5 - 65x^4 - 814x^3 - 741x^2 - 979x - 764$
$g = 216x^4 + 663x^3 + 880x^2 + 916x + 617$
$r_0 = x^5 - \frac{65}{824}x^4 - \frac{407}{412}x^3 - \frac{741}{824}x^2 - \frac{979}{824}x - \frac{191}{2016}$
$r_1 = x^4 + \frac{221}{72}x^3 + \frac{110}{72}x^2 + \frac{229}{54}x + \frac{617}{216}$
$q1 = x - \frac{5837}{1854}$
$a_2 = \frac{614269}{133488}x^3 + \frac{1539085}{200232}x^2 + \frac{931745}{100116}x + \frac{3230125}{400464}$
$p_2 = \frac{614269}{133488}$
$r_2 = x^3 + \frac{3078170}{1842807}x^2 + \frac{3726980}{1842807}x + \frac{3230125}{1842807}$
$q_2 = x + \frac{61877369}{44227368}$
$a_3 = -\frac{1292018949205}{4527916852332}x^2 - \frac{386731352527}{131979213083}x + \frac{91496541567}{2263958426166}$:
$a_5 = -\frac{1289900328081598608308367775585297495}{752235756701500871961459942888522916}$ $p_5 = -\frac{1289900328081598608308367775585297495}{752235756701500871961459942888522916}$
$r_5 = 1$
$q_5 = x + \frac{73158648698031843}{867315257966502554}$
$a_6 = 0$
On remarque donc que les numérateurs et dénominateurs ont bien une taille qui augmente de façon très rapide comparée à la taille de l'entrée.

On choisit d'abord $p \in \mathbb{Z}$ un nombre premier tel que: $p > 2|d|$ (d étant le déterminant) puis on applique l'algorithme de Gauss sur la matrice $\overline{A}^p$.
r le déterminant de A vu comme une matrice dans $\mathbb{Z}/p\mathbb{Z}$, On a alors : $-p/2 < r < p/2$ et $-p/2 < d < p/2$
Ce qui donne: $-p < d - r < p$ et sachant que $p/d - r$, On en conclut que: $d - r = 0$

$$d = r \tag{12}$$

Remarque :

$$\begin{aligned} f &: \mathbb{Z} \to \mathbb{Z}/p\mathbb{Z} \\ & x \mapsto xmodp = \overline{x}^p \end{aligned}$$

f est homomorphisme d'anneaux.

On a alors:

$$\overline{det(A)}^p = det(Amodp) \tag{13}$$

c'est à dire

$$d \equiv r[p] \tag{14}$$

On en déduit que le calcul du déterminant d'une matrice dans $\mathbb{Z}$ peut être fait modulo p. Pour choisir p, on a besoin d'une borne pour d donnée par l'inégalité d'Hadamard.
Soit B la valeur maximale des valeurs absolues des entrées de A.

$$|d| \leq n^{n/2}B^n \tag{15}$$

On note H la borne d'Hadamard. p est un nombre premier qui satisfait: $p > 2 \times H$

Exemple: Soit A= $\begin{pmatrix} 4 & 9 \\ 2 & 7 \end{pmatrix}$
avec l'algorithme de Gauss on a: $\begin{pmatrix} 4 & 9 \\ 0 & 5/2 \end{pmatrix}$
et donc $detA = 10$
Par l'algorithme d'Hadamard $|detA| \leq 2 * 9^2 = 162$. Soit $p = 199$, l'inverse du pivot 4 est 50
det(Amod199)=det $\begin{pmatrix} 4 & 9 \\ 0 & 102 \end{pmatrix} = \overline{408}^{199} = \overline{10}^{199}$

Complexité: $O(n^5 log^2 H)$. La longueur de la borne d'hadamard $\lambda(H)$ est d'à peu près $\frac{1}{64} log_2 H = \frac{1}{64} n(\frac{1}{2} log_2 n + log_2 B$. Elle est donc polynomiale en taille d'entrée $n^2\lambda(B)$. On a un algorithme probabiliste pour trouver le nombre premier p entre $2H$ et $4H$ en temps polynomial. Les calculs modulo p peuvent donc être fait en un temps polynomial $O(log_2 C$ opération (mot machine). Toute les entrées de la matrice sont de valeurs absolues plus petites que p. Il ne se passe rien pour les calculs intermédiaires lors de la réduction modulo p. Le coup de l'algorithme est donc de $O(n^3)$ opérations modulo p. Alors le determinant de la matrice peut être calculé en $O(n^3 n^2((logn + logB)^2)$ ou $O(n^5 log^2 B)$

4.3 Algorithme 3: Small Primes

Algorithme:
Entrée:
A matrice de taille n*n $\in \mathbb{Z}^{n*n}$ avec $a_{i,j} \leq B \ \forall$ i,j
1. $C \leftarrow n^{n/2} B^n$, $r \leftarrow log_2(2C + 1)$
On choisit r nombres premiers distincts $m_0, ..., m_{r-1} \in \mathbb{N}$
2. Pour i=0,..., r-1 calculer $Amodm_i$
3. Pour i=0,..., r-1:
: calculer $d_i \in \{0, ..., m_i - 1\}$ tel que $d_i = detAmodm_i$ avec l'algorithme de Gauss sur $\mathbb{Z}_{m_i}$
4. Appel de l'algorithme de calcul des restes chinois pour déterminer $d \in \mathbb{Z}$ avec la plus petite valeur absolue avec $d = d_i mod m_i$ pour $0 \leq i < r$
5.retourner d
Sortie: $detA \in \mathbb{Z}$

Il y a un algorithme probabiliste qui retourne un nombre premier entre B+1 et 2B pour tout entier positif B d'une longueur β avec une probabilité 3/4. Si $M \in \mathbb{Z}$ tel que $6ln|M| \leq B$ alors p est premier et ne divise pas M avec une probabilité 1/2. Sa complexité est: $O(\beta^2 M(\beta) log\beta)$

Complexité:
Cet algorithme nous permet de calculer r nombres premiers avec une complexité $O(rlog^2 rloglogr)$ sachant que $logm_i \in O(logr)$ pour tout i. ($\sum_{0 \leq i < r} logm_i \in O(rlogr)$). Etant donné qu'une opération arithmétique peut être faite en $O(log^2 r)$, le coût total de l'élimination gaussienne est en $0(n^3 logBrlog^2 r)$. La réduction de A modulo m_i est en $O(logBlogr)$, alors la réduction par rapport à tous les nombres premiers est en $O(n^2 logBrlogr)$. Le coût de l'étape 4 est dominant en $O(r^2 log^2 r)$. Étant donné que $r \in O(nlog(nB))$ On déduit finalement que la complexité totale est:

$$O((n^4 log(nB) + n^3 log^2(nB))(log^2 n + (loglogB)^2))||ouO(n^4 logB + n^3 log^2 B) \tag{16}$$

<u>Conclusion:</u> L'approche avec l'algorithme small primes est plus rapide de deux ordres de magnitude que l'algorithme big prime. Ceci est aussi valable dans le cas ou la matrice a des entrées dans $F[x]$ ou F est un groupe.

5 Implantation et étude expérimentale

5.1 Implantation: (choix techniques effectués et leurs motivations)

En calcul formel, beaucoup d'algorithmes manipulant des polynômes, des fractions rationnelles ou des matrices à coefficients entiers souffrent de la croissance des expressions intermédiaires.
Les méthodes modulaires permettent de remédier à ce problème. Au lieu d'effectuer les calculs sur $\mathbb{Z}$, nous effectuons les calculs dans $F_p = \mathbb{Z}/(p\mathbb{Z})$ pour un (ou plusieurs) nombre(s) premier(s) p. Deux cas sont alors à différencier :

1. Si notre algorithme consiste à décider si une propriété est vraie ou fausse, ou à calculer un degré ou une dimension, alors l'algorithme appliqué aux entrées réduites modulo p donne un algorithme probabiliste répondant à la question.
2. Lorsqu'une borne sur la taille des entiers du résultat est connue, on peut alors utiliser une stratégie basée sur le théorème des restes chinois qui implique qu'un entier inférieur à un produit de nombres premiers $p_1...p_k$ peut être reconstruit à partir de ses réductions modulo $p_1, ..., p_k$. On effectue alors le calcul modulo suffisamment de nombres premiers et on reconstruit le résultat. Cette stratégie est alors du même type que l'évaluation-interpolation.

5.2 Partie expérimentale: Résultant modulaire de deux polynômes dans K[X,Y]

5.2.1 Un algorithme d'Euclide pour calculer le résultant

Lemme:
Soient $f, g \in K[X]$ et $f = q \times g + r$ la division euclidienne .
On a : Res(f,g)=$(-1)^{mn} \times g_m^{n-deg_r} \times Res(g, r)$.

Preuve :
$Soient f = \sum_{i=0}^{n} f_i x^i$ *et* $g = \sum_{j=0}^{m} g_j x^j$ deux polynômes dans K[X] de degré respectifs n et m tel que f=q$\times g + r$ *et* $r, q \in K[x]$.
On applique successivement les deux égalités du corollaire ci−dessus , en remarquant:
f = q$\times g + r$ implique f (b_j) = $r(b_j)$ pour tout j car b_j est racine de g :
On a: Res(f,g)=$(-1)^{nm} \times g_m^n \prod_{j=1}^{m} f(b_j) = (-1)^{nm} \times g_m^n \prod_{j=1}^{m} r(b_j)$.
Et comme: Res(f,g)=$f_n^m \prod_{i=1}^{n} g(a_i)$
Et : $Res(f, g) = g_m^{degr} \prod_{j=1}^{m} r(b_j)$
Alorsona : $\prod_{j=1}^{m} r(b_j) = \left(\frac{Res(g,r)}{g_m^{degr}}\right)$

Et res(f,g)=$(-1)^{nm} \times g_m^n \prod_{j=1}^{m} r(b_j) = (-1)^{nm} \times g_m^n \times \left(\frac{Res(g,r)}{g_m^{degr}}\right)$

On applique successivement la division euclidienne, on obtient:
$Res(f, g) = (-1)^{mn} \times g_m^{n-deg_r} \times Res(g, r)$.

Algorithme

Entrée: Deux polynômes non nuls $f, g \in K[x]$
Sortie: $\text{Res}_x(f, g)$
$Corps$:
$c \leftarrow 1$
Tant que $degg > 0$
Effectuer la division euclidienne:$f = q \times g + r$
$Remplacer\ c \leftarrow (-1)^{mn} \times g_m^{n-deg_r} \times c$
$Remplacer\ (f, g) \leftarrow (g, r) Retourner\ 0 si$ g=0 *et* c$\times g_m^{degf}$ *sinon*

Remarque: Cet algorithme est implémenté par la fonction _nmod_poly_resultant_Euclidean(mp_srcptr poly1 ,slong len1 , mp_srcptr poly2 , slong len2 , nmod_t mod) de FLINT .

On choisit donc "M" maximal 2^{52} tel que $deg(f), deg(g) \notin M$ et on calcule $Res(\overline{f}, \overline{g})$ par un algorithme d'Euclide sur (A/M)[X] : comme A/M est un corps fini, il n'y a pas explosion des coefficients . On en déduit Res(f, g) (mod M) pour suffisamment de M pour pouvoir appliquer le lemme chinois.

Par exemple, si A = Z, on peut calculer Res(f, g) modulo un produit de premiers N = p 1 . . . p l ,

majorer:
$|Res(f,g)| \leq B := \|f\|_2^{degg} \times \|g\|_2^{degf}$
par la borne de Hadamard et il suffit que N >2B pour déterminer Res(f, g) $\in Z$.

Si $A = k[Y]$, on peut de même borner $\deg_Y \times Res_X(f,g)| \leq B$
$B := max(deg_x f \times deg_y g,\ deg_x g \times deg_y f)$.
et calculer Res(f, g) mod (Y a) pour (B+1) *éléments* a $\in k$ distincts ; on reconstruit alors Res(f, g) $\in k[Y]$ par interpolation de Lagrange (qui est une version du lemme chinois).

5.2.2 Évaluation-Interpolation:

Le principe d'évaluation-interpolation est utilisé pour développer des algorithmes manipulant des polynômes. Soit une opération R = Résultant(f(x,y),g(x,y)) à effectuer sur deux polynômes f et g à coefficients dans un anneau effectif A.
Le principe de l'évaluation-interpolation pour calculer R = Résultant(f(x,y),g(x,y)) est alors le suivant :

1. On choisit un certain nombre de points d'évaluation $a_1, ..., a_n \in A$. Le nombre de points dépend du degré du résultant R, qui s'obtient de la manière suivante:
   Si f et g deux polynômes appartenant à $A[X,Y]$ tels que $deg_x f = n$ et $deg_x g = m$ et $deg_y f \leq d$. alors $deg_y R_x \leq ((n+m) \times d + 1)$ ou encore $deg_y R_x \leq n \times m$.
   Et comme c'est un problème d'interpolation il faut choisir ((n+m)$\times d + 1$) points deux à deux distincts ayant pour abscisse les a_i et pour coordonnées Resultant(f(a_i), $g(a_i)$) à fin de construire le résultant modulaire univarié de degré ((n+m)$\times d$) par interpolation.

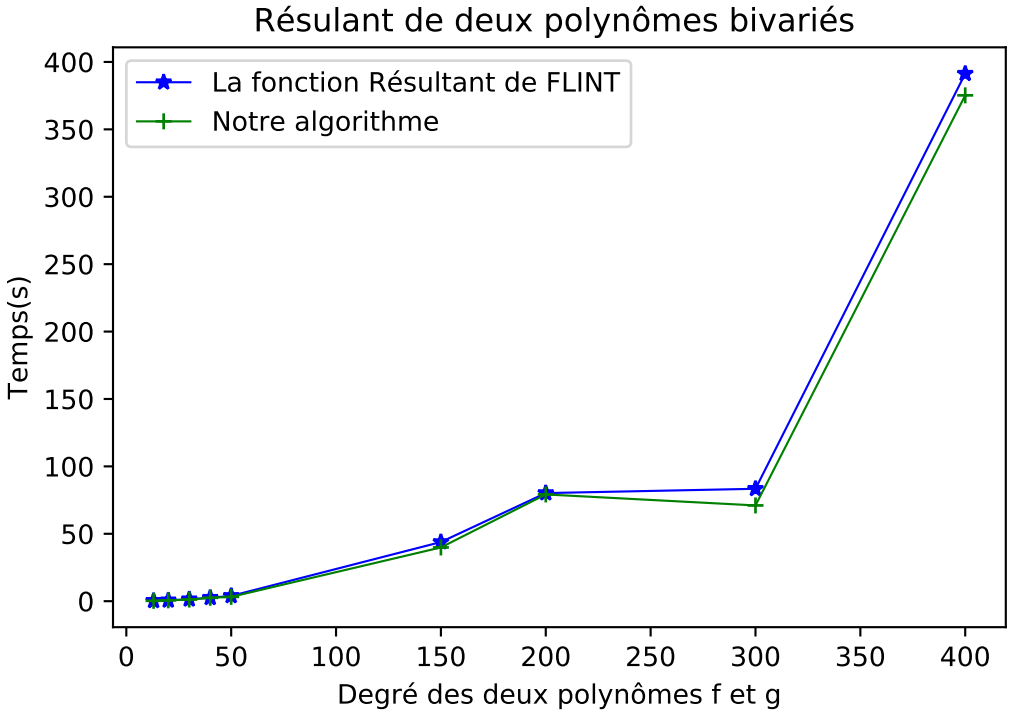
2. On évalue les polynômes f et g en $a_1, ..., a_n$ (évaluation).

3. On en déduit l'évaluation de R en $a1, ..., a_n$.

4. À partir des valeurs $R(a_1), ..., R(a_n)$, on reconstruit R (interpolation polynomiale).

   L'efficacité de la stratégie repose alors sur le choix des points $a_1, ..., a_n$ dans l'anneau A. Le nombre n de points d'évaluation doit être suffisamment grand pour garantir la faisabilité de l'étape 4. De plus les points eux mêmes doivent être choisis pour optimiser le coût des étapes 2 et 4. Par exemple, choisir comme points d'évaluation des racines de l'unité, où des points en progression arithmétique peut s'avérer particulièrement intéressant.

5.2.3 Expériences et conclusion:



Nous avons testé notre algorithme implémentant les méthodes illustrées ci-dessus afin de calculer le résultant de deux polynômes dans K[X,Y] et nous avons enregistré le temps d'exécution de l'algorithme en fonction du degré des deux polynômes. Nous les avons comparé aux temps de calculs trouvés en utilisant le résultant de FLINT. On remarque donc bien que nos notre algorithme est plus rapide.

6 Conclusion:

Calculer le résultant de deux polynômes est une question omniprésente en calcul symbolique avec des applications dans la théorie d'élimination des quantificateurs, la modélisation géométrique, l'étude topologique des courbes algébriques, la résolution de systèmes polynomiaux... Ces deux dernières ont motivé notre étude qui a pour but de retrouver les singularités des applications qui encodent les robots.
Le calcul formel développe des réponses exactes contrairement au calcul scientifique qui calcule des solutions approchées. Même si les architectures des processeurs actuels sont construites pour le calcul numérique, il est possible de construire des représentations exactes en représentant un entier relatif par une liste d'entiers machine ou d'adopter une représentation modulaire. C'est cette dernière méthode qu'on a choisit d'adopter dans l'implantation de notre programme.
Aucun résultat quasi-linéaire n'est connu pour le calcul d'un résultant de deux polynômes à deux variables sur $\mathbb{K}$, ni dans le cas simple d'une variable. Cependant, on a essayé développer une solution qui serait moins coûteuse en terme de temps de calcul mais qui resterait tout de même efficace pour des coefficients de tailles considérablement grandes. On se rend compte de l'efficacité de cette démarche à travers les résultats trouvés au cours d'une étude algorithmique et expérimentale.

References

[1] *GMP documentation.* official website
https://gmplib.org/manual/Formatted-Output-Strings.html
https://gmplib.org/manual/Parameter-Conventions.html#Parameter-Conventions
https://en.wikipedia.org/wiki/GNU_Multiple_Precision_Arithmetic_Library.

[2] Michel Van Caneghem. *Congruences et théorème des restes chinois.* University of Western Ontario London
https://www.apprendre-en-ligne.net/crypto/rabin/resteschinois.html, 2003.

[3] Jean-Marie Chesneaux. *Algorithme de Gauss et calcul du determinant.* Sorbonne Université, 2019.

[4] Jean-Guillaume Dumas. *Algorithmes parallèles efficaces pour le calcul formel :algèbre linéaire creuse et extensions algébriques.* Institut national polytechnique de Grenoble, 2000.

[5] Pavel Emeliyanenko. Modular resultant algorithm for graphics processors. pages 427–440. https://www.researchgate.net/publication/221312853_Modular_Resultant_Algorithm_for_Graphics_Processors, 05 2010.

[6] Laurent KOELBLEN et Patrick POLO. *Algèbre linéaire 2, espaces affines.* Université Pierre et Marie Curie, 2015.

[7] D. Godone G. Garneroa. *COMPARISONS BETWEEN DIFFERENT INTERPOLATION TECHNIQUE.* The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Science
https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XL-5-W3/139/2013/isprsarchives-XL-5-W3-139-2013.pd, 2013.

[8] Alain Kraus. *Arithmétique et Algébre.* Université Paris 6, 2016.

[9] Dinesh Manocha. *Algorithms for computing selected solutions of polynomial equations.* University of North Carolina.

[10] Marc Moreno Maza. *Modular computation of the determinant.* University of Western Ontario London
http://www.csd.uwo.ca/~moreno/CS424/Lectures/EuclideanMethods.html/node11.html, 2008.

[11] Joachim von zur Gathen. *Modern Computer Algebra.* Combridge University Press, 1999.

[12] Michel Waldschmidt. *Algorithme de Bézout et le résultant de deux polynômes.* Université Pierre et Marie Curie.

[13] Sebastian Pancartz William Hart, Frederic Johanson. *FLINT.*
http://www.flintlib.org/downloads.html
https://github.com/kedlaya/flint2-1
http://www.flintlib.org/, 2008.

[11] [8] [6] [3] [12] [9] [4] [10] [2] [5] [7] [13] [1]