Sélection polynomiale pour le crible NFS

Thomas Prest

30 juillet 2010

Table des matières

Introduction

Mon stage, dont l'intitulé exact est "Polynomial Selection for NFS with Non-linear Polynomials", est un stage en cryptographie, reposant essentiellement sur des notions d'algèbre linéaire finie. J'ai travaillé du 1er juin au 31 juillet a l'INRIA* Nancy - Grand Est, au sein de l'équipe Caramel** et sous la direction de Paul Zimmermann.
L'équipe Caramel se concentre sur l'aspect arithmétique de la cryptographie, c'est-à-dire essentiellement sur des problèmes de cryptanalyse tels que la factorisation d'entiers ou le logarithme discret, ainsi que sur l'usage des courbes algébriques en cryptographie, et ce aussi bien d'un point de vue théorique que pratique.
Actuellement, l'un des thèmes de recherche majeurs de Caramel est le crible NFS***, un algorithme de factorisation d'entiers. Le dernier record du monde en date dans ce domaine est d'ailleurs le fruit d'une collaboration entre Caramel et d'autres équipes de recherche à travers le monde, et a été réalisé avec le crible NFS.
Mon travail concerne l'étape initiale de l'algorithme NFS, la sélection polynomiale. J'étais chargé de l'implémenter de façon logicielle, et de réfléchir à des améliorations possibles, l'une et l'autre de ces tâches constituant respectivement les parties opérationnelle et recherche de mon stage.

* Institut National de Recherche en Informatique et Automatique
** Cryptologie, Arithmétique : Matériel et Logiciel
*** Number Field Sieve, c'est-à-dire Crible sur les Corps de Nombres

1 Le crible NFS et la sélection polynomiale

Soit N l'entier qu'on cherche à factoriser.
La première étape de NFS consiste à trouver deux polynômes $f, g \in \mathbb{Z}[X]$ vérifiant les conditions suivantes :

- f et g sont tous deux irréductibles dans $\mathbb{Z}[X]$.
- f et g ont un résultant égal à N, ou tout du moins un multiple de N le plus petit possible (et non nul). La taille du résultant conditionne l'efficacité du crible NFS. Par exemple, si le résultant de f et g a une taille de l'ordre de N^2, NFS mettra autant de temps à factoriser N avec ce choix de polynômes que s'il factorisait un nombre de l'ordre de N^2.
- f et g doivent avoir des coefficients les plus petits possibles, et ce dans le but d'augmenter la rapidité des calculs. En pratique, nous cherchons à obtenir des coefficients de l'ordre de $N^{1/(d_1+d_2)}$, où $d_1 = \deg(f)$, $d_2 = \deg(g)$.
- Une condition facultative mais facile à réaliser est que les coefficients de f et g soient "tordus", autrement dit que pour $i \leq \deg(f)$ on ait $a_i \approx \frac{a_0}{S^i}$, où les a_i sont les coefficients de f, et S le coefficient de torsion (*skewness* en anglais) qui sera laissé à notre discrétion.

1.1 La méthode de la base m

Notons a_i les coefficients de la représentation en base m de N, soit $N = \sum_{i=0}^{d} a_i m^i$. Nous prenons alors le couple de polynômes suivants :

$$f = \sum_{i=0}^{d} a_i X^i$$

$$g = X - m$$

Cette méthode est encore la plus utilisée actuellement, c'est notamment grâce à elle que l'équipe Caramel a réussi à factoriser RSA-768.

Exemple.
Avec $N = 7164152076175143545513361647566709043406333222824787179542 9$ et $m = \lceil N^{1/4} \rceil$, on obtient :

- $f = 517358035741774X^3 + 75931651284441X^2 + 420823926208231X + 80608564750311$
- $g = X - 517358035741777$

1.2 La méthode des deux quadratiques (Montgomery)

Cette méthode, introduite par Peter Montgomery puis développée par Brian Murphy dans sa thèse en 1999, permet d'obtenir deux polynômes de degré 2. Personne n'a pour l'instant réussi à la généraliser à des degrés supérieurs.

L'idée est la suivante : soient $f(X) = a_2X^2 + a_1X + a_0$ et $g(X) = b_2X^2 + b_1X + b_0$ dans $\mathbb{Z}[X]$.

Considérons alors les vecteurs $\mathbf{a} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$ et $\mathbf{c} = \begin{bmatrix} 1 \\ m \\ m^2 \end{bmatrix}$.

f et g ont une racine commune m modulo N si et seulement si $\mathbf{a}$ et $\mathbf{b}$ sont tous deux orthogonaux (sur $\mathbb{Z}_N$) à $\mathbf{c}$.

La méthode des deux quadratiques s'effectue ainsi :

1. On choisit p premier tel que $p < N^{1/2}$ et $\left(\frac{N}{p}\right) = 1$. La deuxième condition sert à assurer l'existence d'une racine carrée de N modulo p.
2. Soit c_1 une racine carrée de N modulo p telle que $|c_1 - N^{1/2}| \leq p/2$.
3. Le vecteur
$$\mathbf{c} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} p \\ c_1 \\ (c_1^2 - N)/p \end{bmatrix} = p \begin{bmatrix} 1 \\ m \\ m^2 \end{bmatrix} \mod N$$
avec $m = c_1 p^{-1} \bmod N$, constitue une suite géométrique modulo N, $(c_i)_{i=0,1,2}$, dont les termes vérifient $c_i = O(N^{1/2})$.
4. Soit $s \in \mathbb{Z}/p\mathbb{Z}$ tel que $c_1 s = 1 \bmod p$. Alors les vecteurs
$$\mathbf{a}' = \begin{bmatrix} c_1 \\ -p \\ 0 \end{bmatrix} \text{ et } \mathbf{b}' = \begin{bmatrix} (c_1(c_2 s \bmod p) - c_2)/p \\ -(c_2 s \bmod p) \\ 1 \end{bmatrix}$$
sont tous deux orthogonaux à $\mathbf{c}$ sur $\mathbb{Z}_N$.
5. En effectuant une réduction LLL sur la base {**a'**,**b'**}, on trouve une base {**a**,**b**} telle que $\|\mathbf{a}\| \cdot \|\mathbf{b}\| = O(\|\mathbf{c}\|) = O(N^{1/2})$.

En pratique, $\|\mathbf{a}\|$ et $\|\mathbf{b}\|$ sont tous deux $O(N^{1/4})$, sans doute à cause des relations entre les coefficients. Chaque nombre premier p donne deux couples de polynômes distincts (en effet on a deux possibiités pour c_1, une par racine de N modulo p), nous pouvons donc générer ainsi une infinité de paires de polynômes, parmi lesquelles nous n'aurons plus qu'à chercher la meilleure.

Exemple.
Avec $N = 71641520761751435455133616475667090434063332228247871795429$:

1. Prenons par exemple $p = 7$; on a bien $\left(\frac{N}{p}\right) = 1$.
2. On obtient $c_1 = 267659337146589069735395147282$; on a bien $c_1^2 = 1 \pmod p = N \pmod p$.
3. $\mathbf{c} = \begin{bmatrix} 7 \\ 267659337146589069735395147282 \\ -106229264412112666619057115415 \end{bmatrix}$
4. $\mathbf{a}' = \begin{bmatrix} 267659337146589069735395147282 \\ -7 \\ 0 \end{bmatrix}, \mathbf{b}' = \begin{bmatrix} 16812380185692413508009110064 9 \\ -4 \\ 1 \end{bmatrix}$
5. Après réduction LLL, les vecteurs obtenus sont :
   $\mathbf{a} = \begin{bmatrix} -391799550615569 \\ -155498322989920 \\ -23601103928385 \end{bmatrix}$ et $\mathbf{b} = \begin{bmatrix} 196400087271641 \\ 77947726478583 \\ -671323072887913 \end{bmatrix}$
6. **Vérification.**
   $f(X) = -391799550615569X^2 - 155498322989920X - 23601103928385$ et $g(X) = 196400087271641X^2 + 77947726478583X - 671323072887913$ admettent $m = c_1 p^{-1}$ comme racine commune modulo N, et on a $\mathrm{Res}(f, g) = N$.

2 Sélection de polynômes de degré 2 optimaux pour le crible NFS

2.1 Objectifs

La première partie de mon stage a consisté à écrire un programme renvoyant des polynômes de degré 2 optimaux pour le crible NFS. Il s'agit donc tout d'abord de sélectionner un grand nombre de polynômes avec la méthode de Montgomery, puis d'évaluer ces couples selon des critères bien précis :

1. f et g doivent avoir de bonnes propriétés de friabilité, ce qui se traduit par la nécessité de minimiser les valeurs $\alpha(F)$ et $\alpha(G)$, où :
   - $F(x,y) = y^{\deg(f)} f(x/y)$ et $G(x,y) = y^{\deg(g)} g(x/y)$.
   - $\alpha(F) = \sum_{p \leq B} (\frac{1}{p-1} - \nu_p(F)) \log p$.
   - $\nu_p(F)$ désigne la p-valuation moyenne de $F(x,y)$ pour x et y premiers entre eux.
2. Soit $s > 0$. On note

$$L_s(F,G) = \iint_{(x,y)\in[-1,1]^2} F^2(sx, y/s)\ G^2(sx, y/s)\ dx\ dy$$

   Notons $L(F,G)$ le minimum de cette fonction pour F et G fixés, c'est-à-dire $L(F,G) = \min_{s>0} L_s(F,G)$.
   Nous cherchons les couples de polynômes minimisant la valeur $L(F,G)$.
3. Les polynômes sont donc "notés" ainsi :

$$notation(F,G) = \alpha(F) + \alpha(G) + \frac{1}{2} \log L(F,G)$$

Les couples de polynômes ayant la notation la plus faible seront sélectionnés pour le crible NFS.

2.2 Implémentation en C

Mon premier travail a donc été d'implémenter la méthode de Montgomery en C. Dans un premier temps, j'ai écrit un programme en Sage afin de tester son bon fonctionnement et son efficacité.
Une fois cela effectué, j'ai retranscrit le programme en C. J'ai pour cela utilisé la bibliothèque GMP, qui permet de gérer de façon efficace les grands nombres.
L'intégralité de mon programme est retranscrite en **annexe**.
Voici un bref aperçu des fonctions mathématiques utilisées dans mon programme et de la façon dont je les ai implémentées.

2.2.1 Racine carrée modulo p

La recherche d'une racine carrée se fait avec l'algorithme de Tonelli-Shanks (fonction `modular_square_root`). Cet algorithme a une complexité moyenne en $O(\log p)$

2.2.2 Réduction LLL

La réduction LLL (fonction `LLL`) s'apparente ici à un calcul de PGCD. Soit $\{\mathbf{a},\mathbf{b}\}$ la base à réduire, on effectue tout d'abord $\{a_0 = a, b_0 = b, i = 0\}$, puis :

- $i++$
- $a_i = \min\limits_{k\in\mathbb{Z}}(a_{i-1} + kb_{i-1})$
- $b_i = \min\limits_{k\in\mathbb{Z}}(b_{i-1} + ka_i)$

et ce jusqu'à ce que $a_i = a_{i-1}$ et $b_i = b_{i-1}$. La procédure renvoie alors $\{a_i, b_i\}$

2.2.3 Calcul de l'intégrale $L_s(F, G)$

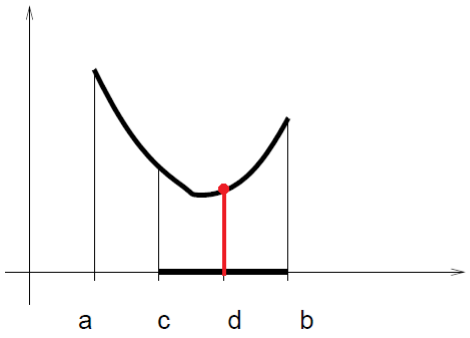
Le calcul de $L_s(F, G)$ est calculé de façon explicite en fonction de s et des coefficients de f et g (fonction `TTT`) :

$$\begin{aligned} L_s(F,G) \quad &= 4/1575s^2 \\ &\times(175a_0^2b_0^2 \\ &+75(2a_0^2b_0b_2 + a_0^2b_1^2 + 4a_0a_1b_0b_1 + (2a_0a_2 + a_1^2)b_0^2)s \\ &+63(a_0^2b_2^2 + 4a_1a_2b_0b_1 + a_2^2b_0^2 + (2a_0a_2 + a_1^2)b_1^2 + 2(2a_0a_1b_1 + (2a_0a_2 + a_1^2)b_0)b_2)s^2 \\ &+75(a_2^2b_1^2 + (2a_0a_2 + a_1^2)b_2^2 + 2(2a_1a_2b_1 + a_2^2b_0)b_2)s^3 \\ &+175a_2^2b_2^2s^4) \end{aligned}$$

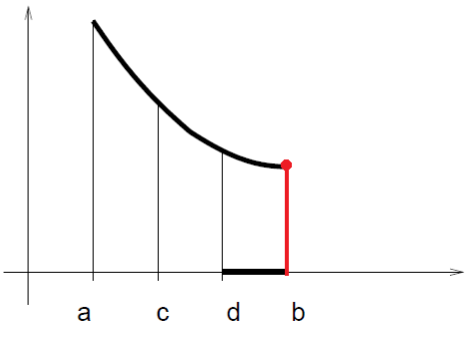
2.2.4 Approximation de $L(F, G)$

Le minimum de $L_s(F, G)$ est calculé de manière approchée par trichotomie (fonction `normeL2`). La trichotomie est une méthode informatique permettant d'approcher le minimum d'une fonction convexe sur un intervalle fini.

Méthode de la trichotomie.
La trichotomie repose sur le principe suivant : soit f une fonction convexe sur un intervalle $I = [a; b]$, et $a < c < d < b$ quatre points de I. Notons $x_0 = \underset{\{a,b,c,d\}}{\operatorname{argmin}} f(x)$. Alors f atteint son minimum sur l'intervalle I_{x_0} "entourant" x_0, autrement dit sur $[a; c]$ si $x_0 = a$, sur $[a; d]$ si $x_0 = c$, sur $[c; b]$ si $x_0 = d$, et sur $[d; b]$ si $x_0 = b$.



$\underset{\{a,b,c,d\}}{\mathbf{argmin}}\ \mathbf{f(x) = d}$



$\underset{\{a,b,c,d\}}{\mathbf{argmin}}\ \mathbf{f(x) = b}$

On peut alors, de façon récursive, réduire de plus en plus l'intervalle sur lequel évaluer f. À chaque itération, l'intervalle est réduit par un facteur 3 ou 3/2. L'algorithme approxime donc le minimum de f avec une vitesse de convergence en 3/2.

2.2.5 Calcul de α

La fonction effectuant ce calcul a été écrite par un autre membre de l'équipe.

L'implémentation en C est environ 1000 fois plus rapide qu'en Sage.

3 Recherche de polynômes de degrés ≥ 3 efficaces pour le crible NFS

Une fois effectuée l'implémentation de la méthode "Two quadratics" de Montgomery, ma seconde tâche a consisté à développer des méthodes pouvant générer des couples de polynômes adéquats de degrés ≥ 3.

3.1 Trouver un couple de polynômes de degré $d \geq 3$ via la méthode de la base m

Dans cette section nous utiliserons la methode de la base m pour trouver deux polynômes de degré $d \geq 3$ possédant une racine commune m modulo N.
Soient a_i les coefficients de la représentation en base m de N .

$$N = \sum_{i=0}^{d} a_i \ m^i$$

avec $0 \leq a_i \leq m$.
Nous devons au préalable imposer la condition $m^d < N < m^{d+1}$, afin d'assurer que f soit un polynôme de degré d.
Le polynôme $f(X) = \sum_{i=0}^{d} a_i \ X^i$ admet m comme racine modulo N.

La prochaine étape consiste à minimiser la taille de certains de ces coefficients. En prenant m très proche de $N^{1/d}$, nous pouvons garantir que le coefficient dominant de f sera en $O(1)$.

Nous pouvons faire mieux, par exemple en minimisant a_0. Soit c un petit entier (positif ou négatif) tel que $N - c$ soit friable :

$$N - c = \prod_{i \in I} p_i$$

Pour tout $i \in I$, nous avons $N - c = 0 \ [\text{mod } p_i]$, donc pour tout $J \subset I$, nous avons

$$N - c = 0 \ [\text{mod } \prod_{i \in J} p_i].$$

Soit $m_J = \prod_{i \in J} p_i$. En choisissant intelligemment J, nous pouvons nous assurer que $m_J^d < N < 2m_J^d$. Cela nous donnera donc $a_0 = c$ et $a_d = 1$
On obtient donc un polynôme $f = x^d + a_{d-1}x^{d-1} + ... + a_1 x + c$.

Exemple :
Avec $N = 71641520761751435455133616475667090434063332228247871795429$ et $d = 3$, nous avons :
$N-1 = 2^2 \times 7 \times 617 \times 4984428238241711837 \times 831967292613905141715382511023497619$
$\lfloor N^{1/3} \rfloor = 41532518328905347815$
$m = 7 \times 4984428238241711837 = 34890997667691982859 \approx 0.84 \times \lfloor N^{1/3} \rfloor$
$f = X^3 + 239578577799814931 07X^2 + 4518649086571656898X + 1$

Une fois que nous avons f, nous pouvons créer g ainsi :
$g = k \times f + c(X)$, où $c(X)$ est un multiple de $X - m$. On ne sait pas exactement quel est le meilleur choix pour $c(X)$, cela dit pour $d = 3$, les relations suivantes sont intéressantes :

1. $g = kf + (X - m) \Longrightarrow \text{Res}(f, g) = -a_3^2 N$
2. $g = kf + X(X - m) \Longrightarrow \text{Res}(f, g) = a_0 a_3 N$
3. $g = kf + X^2(X - m) \Longrightarrow \text{Res}(f, g) = -a_0^2 N$

Exemple :
Avec le nombre N de l'exemple précédent, on obtient
$g = 3X^3 + 1302471789227100335 5X^2 + 9037298173143313796X + 2$
Et nous obtenons effectivement $\text{Res}(f, g) = -N$

Cependant, ces polynômes sont loin d'être optimaux. En effet, leurs coefficients sont en $O(N^{1/d})$, alors que nous les voudrions en $O(N^{1/2d})$. De plus, ils donnent de très mauvais résultats pour le crible. Par exemple, pour le nombre N et les pôlynomes f et g mentionnés ci-dessus, le crible nous fournit 84 relations sur un intervalle donné, là où des quadratiques obtenus par la méthode de Montgomery nous en donnent des milliers.

Un autre choix est d'obtenir des polynômes avec de plus petits coefficients, mais en contrepartie un plus gros résultant.
Prenons encore une fois $f(X) = \sum_{i=0}^{d} a_i \ X^i$, où les a_i sont les coefficients de la représentation en base m de N.

En effectuant une réduction LLL sur la matrice

$$A = \begin{pmatrix} a_0 & 0 & ... & 0 & 0 \\ a_1 & -m & ... & 0 & 0 \\ a_2 & 1 & ... & 0 & 0 \\ ... & ... & ... & ... & ... \\ a_{d-1} & 0 & ... & 1 & -m \\ a_d & 0 & ... & 0 & 1 \end{pmatrix}$$

nous obtenons de plus petits vecteurs.
Pour $d = 3$, nous obtenons des coefficients en $O(N^{2/9})$. C'est très éloigné de la borne inférieure qui est $O(N^{1/6})$ et nous fournit un résultant qui est en théorie (et en pratique) en $O(N^{4/3})$.

3.2 Une généralisation de la méthode de Montgomery

Nous allons voir s'il est possible de généraliser la méthode de Montgomery à des degrés ≥ 3, et si oui, dans quelle mesure.
L'idée consiste donc à prendre une progression géométrique modulo N de la forme $(p^{d-1}, p^{d-2}c, ..., pc^{d-2}, c^{d-1}, \frac{c^d-N}{p})$ à $d+1$ éléments, et à effectuer une réduction LLL sur la matrice

$$A = \begin{pmatrix} 1 & 0 & ... & 0 & 0 \\ 0 & S & ... & 0 & 0 \\ ... & ... & ... & ... & ... \\ 0 & 0 & ... & S^{d-1} & 0 \\ 0 & 0 & ... & 0 & S^d \\ Kp^{d-1} & Kp^{d-2}c & ... & Kc^{d-1} & K\frac{c^d-N}{p} \end{pmatrix}$$

en espérant obtenir des vecteurs courts.

3.2.1 Généralisation avec une matrice 4×4

Considérons

$$A = \begin{pmatrix} S^\alpha & 0 & 0 & 0 \\ 0 & S & 0 & 0 \\ 0 & 0 & S^2 & 0 \\ Kp^2 & Kpc & Kc^2 & K\frac{c^3-n}{p} \end{pmatrix}$$

En effectuant une réduction LLL sur la matrice A, nous obtenons un vecteur

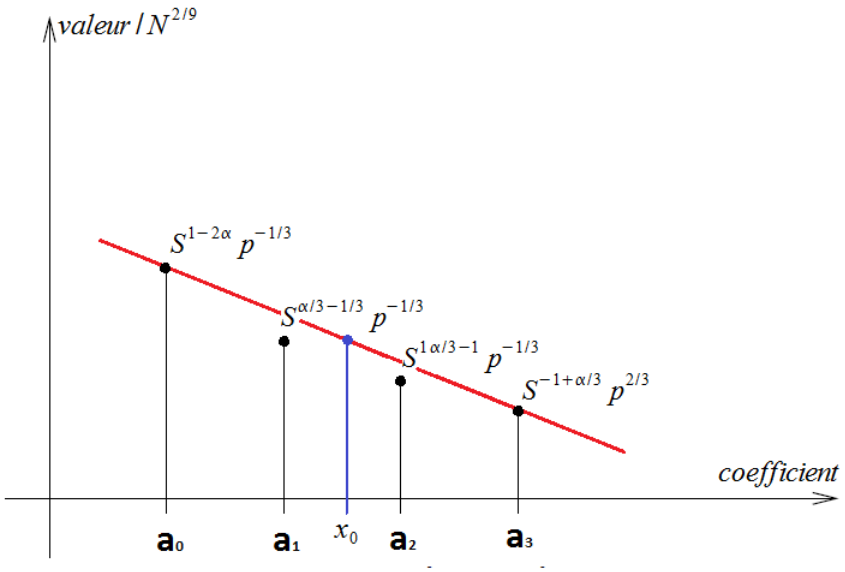
$$\mathbf{a} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

dont les coefficients vérifient

$$a_0 S^\alpha \approx K \Rightarrow a_0 \approx S^{1-2\alpha/3} N^{2/9} p^{-1/3}$$
$$a_1 S \approx K \Rightarrow a_1 \approx S^{\alpha/3} N^{2/9} p^{-1/3}$$
$$a_2 S^2 \approx K \Rightarrow a_2 \approx S^{-1+\alpha/3} N^{2/9} p^{-1/3}$$
$$a_0 \frac{c^3-n}{p} \approx a_2 c^2 \Rightarrow a_3 \approx S^{-1+\alpha/3} N^{2/9} p^{2/3}$$

Minimisation de la taille médiane des coefficients

Définissons à présent la taille médiane des coefficients. Notre but dans la suite sera de minimiser cette valeur.
Considérons la représentation logarithmique de ces coefficients.



Traçons la droite passant au-dessus de tous les points. La taille médiane des coefficients sera la valeur de cette droite (qui n'en est pas vraiment une puisque les coefficients sont représentés en taille logarithmique) en le point $x_0 = 3/2$.
Nous cherchons à minimiser la taille médiane. Cela est équivalent à considérer les droites x_{ij} passant par les points P_i et P_j. Comme $a_2 < a_3$, nous n'aurons qu'à considérer les droites suivantes :

$$x_{03} = \frac{\log(p)}{6} + \log(S)(-\frac{\alpha}{6}) + C$$

$$x_{13} = -\frac{\log(p)}{12} + \log(S)(\frac{\alpha}{3} - \frac{1}{4}) + C$$

$$x_{01} = -\frac{\log(p)}{3} + \log(S)(\frac{5\alpha}{6} - \frac{1}{2}) + C$$

Nous cherchons le maximum de x_{01}, x_{03} et x_{13} en $3/2$ en fonction de α. Cela nous donne la valeur

$$\alpha = \frac{\log(p) + \log(S)}{2\log(S)}$$

soit $\alpha = 1/2$ pour $\log(p) \ll \log(S)$.
En pratique nous prendrons $p = O(1)$ (afin de minimiser a_3), donc $\alpha = 1/2$.

La valeur médiane des coefficients est alors $S^{-12}N^{2/9}$, et le résultant est donc de l'ordre de $S^{-2}N^{4/3}$.
Comme on peut le voir, plus S est élevé et plus le résultant est faible.

Jusqu'où peut-on donc élever S ?
On a $\det(M) \approx K^4$, donc $K \approx S^{7/6}N^{2/9}$. Nous cherchons à ce que le polynôme linéaire $pX - c$ ne soit pas "détecté" par la réduction LLL. Or le vecteur correspondant au polynôme linéaire est le vecteur $\mathbf{v} = c\mathbf{v_1} - p\mathbf{v_2}$, où $\mathbf{v_1}$ et $\mathbf{v_2}$ sont les $1^{ère}$ et $2^{ème}$ colonnes de A.
$\mathbf{v}$ a une norme $n_0 \approx cS^{1/2} \approx N^{1/3}S^{1/2}$. $\mathbf{v}$ n'est détecté que si $n_0 \geq K$. On souhaite donc que $K \ll n_0$, c'est-à-dire $S \ll N^{1/6}$.
Cela nous donne une valeur médiane de l'ordre de $S^{-1/12}N^{2/9} = N^{5/24}$, et donc un résultant de l'ordre de $N^{5/4}$.

Exemple.
En prenant :
- N = 716415207617514354551336164756670904340633322282478717954 29
- c = $\lceil N^{1/3} \rceil$ = 41532518328905347816
- p = 1
- S = 1000000000
- K = 10000000000000000000000000000000000

On obtient :
- f = 42044X^3 - 58243X^2 + 216589713956652X + 309824665860518028
- g = 189599X^3 - 262649X^2 - 11115144906243X - 3123165185295940301

3.2.2 Généralisation avec une matrice 5×4

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & S & 0 & 0 \\ 0 & 0 & S^2 & 0 \\ 0 & 0 & 0 & S^3 \\ Kp^2 & Kpc & Kc^2 & K\frac{c^3-N}{p} \end{pmatrix}$$

Une réduction LLL de cette matrice nous donne des vecteurs de la forme $\begin{pmatrix} a_0 \\ a_1S \\ a_2S^2 \\ a_3S^3 \\ 0 \end{pmatrix}$.
Les a_i vérifient $K \approx a_iS^i$, et on cherche à minimiser la taille "médiane" de ces

coefficients, qui vaut ici $KS^{-3/2}$.
Pour cela, estimons tout d'abord K.
On a $K^4 \approx (\det({}^tM.M))^{1/2}$ et

$$\begin{aligned}\det({}^tM.M) &= (K^2S^6p^6 + K^2S^4c^2p^4 + K^2S^2c^4p^2 + K^2c^6 + S^6p^2 - 2K^2c^3N + K^2N^2)S^6/p^2 \\ &\approx \tfrac{K^2S^6}{p^2}(S^6p^6 + S^4c^2p^4 + S^2c^4p^2 + (c^3 - N)^2) \\ &\approx \tfrac{K^2S^6}{p^2}(S^6p^6 + S^4c^2p^4 + S^2c^4p^2 + N^{4/3})\end{aligned}$$

Nous distinguons alors deux cas :

- Si $Sp \ll N^{1/3}$, le terme dominant de l'expression ci-dessus est $\frac{K^2S^6}{p^2} \times S^2c^4p^2$ et alors $\det({}^tM.M) \approx S^8c^4K^2$.
  Or $K \approx (\det({}^tM.M))^{1/8}$, d'où l'on deduit $K \approx S^{4/3}N^{2/9}$.
  La valeur moyenne des coefficients est alors $KS^{-3/2} \approx S^{-1/6}N^{2/9}$
- Si $Sp \gg N^{1/3}$, le terme dominant est S^6p^6, et on a $\det({}^tM.M) \approx K^2S^{12}p^4$.
  On en déduit donc $K \approx S^2p^{2/3}$.
  La valeur moyenne des coefficients est alors $KS^{-3/2} \approx S^{1/2}p^{2/3} = (Sp)^{1/2}p^{1/6}$

Nous nous limiterons au premier cas, dans la mesure ou cette méthode ne nous renvoie que des polynômes linéaires dès que $Sp \geq N^{1/3}$, ce qui rend inexploitables les résultats obtenus pour $Sp \gg N^{1/3}$.
Nous pouvons par ailleurs obtenir une borne supérieure pour S. En effet, nous souhaitons que le polynôme linéaire $pX - c$ ne soit pas détecté,

Jusqu'où peut-on donc élever S ?
De la même façon qu'avant, nous cherchons à ce que le polynôme linéaire $pX - c$ ne soit pas "détecté" par la réduction LLL. Or le vecteur correspondant au polynôme linéaire est le vecteur $\mathbf{v} = c\mathbf{v_1} - p\mathbf{v_2}$, où $\mathbf{v_1}$ et $\mathbf{v_2}$ sont les $1^{ère}$ et $2^{ème}$ colonnes de A.
$\mathbf{v}$ a une norme $n_0 \approx c \approx N^{1/3}$. $\mathbf{v}$ n'est détecté que si $n_0 \geq K$. On souhaite donc que $K \ll n_0$, c'est-à-dire $S \ll N^{1/12}$.
Cela nous donne une valeur médiane de l'ordre de $N^{5/24}$, et donc un résultant de l'ordre de $N^{5/4}$.

Exemple.
En prenant :

- N = 7164152076175143545513361647566709043406333222828 47871795429
- S = 10000
- K, p et c pareils qu'à l'exemple précédent,

On obtient exactement les mêmes polynômes qu'avec la matrice 4×4 !

3.3 Recherche d'une suite géométrique à l'aide de l'algorithme de Coppersmith

3.3.1 Suites géométriques à 3 éléments

Fixons $0 < M < N$, et considérons la suite suivante :
- $x_0 = x$
- $x_1 = (M + s)x \bmod N = -z$
- $x_2 = (M + s)^2 x \bmod N = -w$

avec $x, z, w = O(N^{1/2})$ et $s = O(S)$. Considérons les polynômes $f = (M+s)x+z$ et $g = (M+s)^2x+w$. Si nous décomposons les polynômes $Nx, Nsx, Ns^2x, f, g, fs, fs^2, gs$ dans la base de monômes $x, sx, s^2x, z, w, zs, xs^3, zs^2, ws$, nous obtenons la matrice 8×9 suivante :

$$A = \begin{pmatrix} N & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & N & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & N & 0 & 0 & 0 & 0 & 0 & 0 \\ M & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ M^2 & 2M & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & M & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & M & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & M^2 & 2M & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Une réduction LLL (selon les lignes) nous donne une nouvelle matrice B. Sélectionnons 4 lignes de B, ces dernières nous fournissent un système d'équations non-linéaires à 4 variables (s, x, z et w) que nos pouvons résoudre dans $\mathbb{Z}^4$ (le plus simple étant en fait de le résoudre dans $\mathbb{R}^4$, puis d'isoler les solutions entières).
En pratique, les solutions entières non-triviales que nous trouvons sont de l'ordre de $N^{2/3}$ et nous donnent donc une progression géométrique modulo N en $O(N^{2/3})$ à 3 éléments.

3.3.2 Suites géométriques à 4 éléments

Nous pouvons adapter cette méthode pour obtenir une suite à 4 éléments. Considérons cette fois les polynômes :

- $f = (M + s)x + z$
- $g = (M + s)^2x + w$
- $h = (M + s)^3x + y$

En décomposant les polynômes $Nx, Nsx, Ns^2x, Ns^3x, f, g, h, fs, gs, hs, fs^2, gs^2, fs^3$ dans la base de monômes $x, sx, s^2x, z, w, zs, xs^3, zs^2, ws, y, xs^4, ws^2, sy, s^3z$, nous obtenons la matrice 13×14

$$A = \begin{pmatrix} N & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & N & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & N & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & N & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ M & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ M^2 & 2M & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ M^3 & 3M^2 & 3M & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & M & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & M^2 & 2M & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & M^3 & 3M^2 & 0 & 0 & 0 & 3M & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & M & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & M^2 & 0 & 0 & 0 & 2M & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & M & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Encore une fois, on effectue une réduction LLL sur les lignes, puis nous sélectionnons 5 lignes qui nous fournissent un système d'équations polynomiales en les variables x, s, z, w, y que nous résolvons, ce qui nous permet cette fois d'obtenir une progression géométrique modulo N à 4 éléments.
En pratique, nous obtenons une progression géométrique en $O(N^{3/4})$, ce qui est bien moins efficace que ce que nous obtenons par des méthodes plus triviales !

Conclusion

La partie de mon stage consistant à implémenter l'algorithme de Montgomery a été très gratifiante, en particulier par l'utilité immédiate et tangible de mon travail, et m'a permis en plus d'apprendre la programmation en C.
La deuxième partie, celle qui consistait à développer de nouveaux algorithmes, a été plus frustante, car les progrès étaient lents et laborieux. Je n'ai pas réussi à développer de meilleure méthode que celle de Montgomery, qui risque par ailleurs de bientôt s'imposer comme la méthode de référence pour la sélection polynomiale. J'ai cependant réussi, avec l'aide de mon maître de stage, à ouvrir quelques voies vers de nouveaux algorithmes de sélection polynomiale. L'objectif était d'obtenir un résultant en $O(N)$, je n'ai réussi à obtenir au mieux qu'un résultant en $O(N^{5/4})$, ce qui n'est pas un succès mais constitue un début.
Enfin, contrairement à mon stage de l'an dernier, qui s'était résumé dans l'esprit à de la lecture de bouquins, ce stage m'aura permis d'explorer toutes les facettes du métier de chercheur, car en plus de faire de la programmation et de la recherche, j'ai pu assister à des séminaires et à des conférences et échanger avec mes collègues.

Annexe

```
#include <stdio.h>
#include <stdlib.h>
#include <gmp.h>
#include <assert.h>
#include "cado.h"
#include "auxiliary.h"
#include "murphyE.h"
#include "rootsieve.h"
#include "rho.h"


//affiche un nombre GMP flottant en utilisant la notation scientifique
//etrangement, cette fonction ne semble pas implementee en GMP

void printscientific(mpf_t x) {

    mpf_t mantisse;
    mpf_init2(mantisse, 10);
    mpf_set(mantisse, x);
    int exposant = 0;
    while(mpf_cmp_ui(mantisse, 10) > 0) {
        mpf_div_ui(mantisse, mantisse, 10);
        exposant++;
    }
    gmp_printf ("%Ff", mantisse);
    printf("e%d", exposant);

    mpf_clear(mantisse);

}


//renvoie la racine carree de N modulo p
//L'algorithme utilise est Tonelli-Shanks
unsigned long int modular_square_root(mpz_t N, unsigned long int p) {

    mpz_t P;
    mpz_init_set_ui(P, p);

    assert(mpz_kronecker(N, P) == 1);

    mpz_t n;
    mpz_init(n);
    mpz_mod(n, N, P);

    mpz_t z;
    mpz_init_set_ui(z, 2);

    mpz_t c;
    mpz_init(c);

    mpz_t R;
    mpz_init(R);

    mpz_t t;
    mpz_init(t);

    mpz_t t0;
    mpz_init(t0);

    mpz_t b;
    mpz_init(b);

    if (p == 2) {
        return mpz_get_ui(n);
    }

    int Q = p-1;
    int S = 0;
    while (Q%2 == 0) {
        S++;
        Q = Q/2;
    }

    int M = S;

    while (mpz_legendre(z, P) != -1) {
        mpz_add_ui(z, z, 1);
    }
    mpz_powm_ui(c, z, Q, P);
    mpz_powm_ui(R, n, (Q+1)/2, P);
```

```
    mpz_powm_ui(t, n, Q, P);

    while (1 == 1) {
        if (mpz_cmp_ui(t, 1) == 0) {
            mpz_clear(P);
            mpz_clear(n);
            mpz_clear(z);
            mpz_clear(c);
            mpz_clear(t);
            mpz_clear(t0);
            mpz_clear(b);
            unsigned long int RR = mpz_get_ui(R);
            mpz_clear(R);
            return RR;
        }
        int i = 0;
        mpz_set(t0, t);
        while(mpz_cmp_ui(t, 1) != 0) {
            mpz_pow_ui(t, t, 2);
            mpz_mod(t, t, P);
            i++;
        }
        mpz_set(b, c);
        int j;
        for (j = 0; j < M - i - 1; j++) {
            mpz_pow_ui(b, b, 2);
            mpz_mod(b, b, P);
        }
        mpz_mul(R, R, b);
        mpz_mod(R, R, P);
        mpz_mul(c, b, b);
        mpz_mod(c, c, P);
        mpz_mul(t, t0, c);
        mpz_mod(t, t, P);
        M = i;
    }


}


//affecte a no la valeur de la norme euclidienne de a
void norm(mpz_t no, mpz_t a[3]) {

    mpz_mul(no, a[0], a[0]);
    mpz_addmul(no, a[1], a[1]);
    mpz_addmul(no, a[2], a[2]);

}


//une etape de la reduction LLL
void reduction(mpz_t k, mpz_t a[3], mpz_t b[3]) {

    mpz_t n;
    mpz_init(n);
    mpz_mul(n, a[0], b[0]);
    mpz_addmul(n, a[1], b[1]);
    mpz_addmul(n, a[2], b[2]);

    mpz_t d;
    mpz_init(d);
    norm(d, b);

    mpz_fdiv_q(k, n, d);


    mpz_t X1[3];
    int i;
    for (i = 0; i < 3; i++) {
        mpz_init_set(X1[i], a[i]);
        mpz_submul(X1[i], k, b[i]);
    }
    mpz_t norm1;
    mpz_init(norm1);
    norm(norm1, X1);

    mpz_t X2[3];
    for (i = 0; i < 3; i++) {
        mpz_init_set(X2[i], X1[i]);
        mpz_sub(X2[i], X2[i], b[i]);
    }
```

```
    mpz_t norm2;
    mpz_init(norm2);
    norm(norm2, X2);

    if (mpz_cmp(norm1, norm2) > 0) {
        mpz_add_ui(k, k, 1);
        for (i = 0; i < 3; i++) {
            mpz_set(a[i], X2[i]);
        }
    }
    else {
        for (i = 0; i < 3; i++) {
            mpz_set(a[i], X1[i]);
        }
    }

    mpz_clear(n);
    mpz_clear(d);
    for (i = 0; i < 3; i++) {
        mpz_clear(X1[i]);
        mpz_clear(X2[i]);
    }
    mpz_clear(norm1);
    mpz_clear(norm2);

}


//la reduction LLL sur deux vecteurs entiers
void LLL(mpz_t a[3], mpz_t b[3]) {

    mpz_t k;
    mpz_init_set_ui(k, 1);

    mpz_t l;
    mpz_init_set_ui(l, 1);

    while ( (mpz_cmp_si(k, 0) != 0) || (mpz_cmp_si(l, 0) != 0) ) {

        reduction(k, a, b);
        reduction(l, b, a);
    }

    mpz_clear(k);
    mpz_clear(l);

}


//affecte a a et b les coefficients d'une des deux paires de polynomes trouves via la m thode de Montgomery,
//etant donnes N et p.
void TQponctuel(mpz_t N, unsigned long int p, mpz_t a[3], mpz_t b[3], int i) {
assert((i == 0)||(i == 1));

    mpz_t P;
    mpz_init_set_ui(P, p);

    mpz_t z;
    mpz_init(z);
    mpz_sqrt(z, N);


    //On calcule les vecteurs a, b, c de la fa on decrite par Montgomery
    //Ces vecteurs forment un reseau sur Z^3

    mpz_t c0;
    mpz_init_set_ui(c0, p);

    mpz_t c1;
    mpz_init_set_ui(c1, modular_square_root(N, p));
    if (i == 1) {mpz_neg(c1, c1);}

    mpz_t X;
    mpz_init(X);
    mpz_add_ui(X, z, p/2);
    mpz_sub(X, X, c1);
    mpz_fdiv_q_ui(X, X, p);
    mpz_mul_ui(X, X, p);
    mpz_add(c1, c1, X);

    mpz_t c2;
    mpz_init(c2);
    mpz_mul(c2, c1, c1);
```

```
    mpz_sub(c2, c2, N);
    mpz_fdiv_q_ui(c2, c2, p);

    mpz_t c[3];
    mpz_init_set(c[0], c0);
    mpz_init_set(c[1], c1);
    mpz_init_set(c[2], c2);

    mpz_set(a[0], c1);
    mpz_set_si(a[1], -p);
    mpz_set_si(a[2], 0);

    mpz_invert(X, c1, P);
    mpz_mul(X, X, c2);
    mpz_mod(X, X, P);

    mpz_t b1;
    mpz_init_set(b1, X);
    mpz_neg(b1, b1);

    mpz_mul(X, c1, X);
    mpz_sub(X, X, c2);
    mpz_fdiv_q_ui(X, X, p);

    mpz_t b0;
    mpz_init_set(b0, X);

    mpz_set(b[0], b0);
    mpz_set(b[1], b1);
    mpz_set_ui(b[2], 1);

   //Puis on effectue une reduction LLL sur a et b

    LLL(a, b);

    mpz_clear(c0);
    mpz_clear(c1);
    mpz_clear(c2);
    mpz_clear(b0);
    mpz_clear(b1);
    mpz_clear(X);
    mpz_clear(z);
    mpz_clear(P);
    for (i = 0; i < 3; i++) {
        mpz_clear(c[i]);
    }

}


//TTT(x, a[3], b[3]) = 4/1575*(
//                       175*a0^2*b0^2
//                     + 75*(2*a0^2*b0*b2 + a0^2*b1^2 + 4*a0*a1*b0*b1 + (2*a0*a2 + a1^2)*b0^2)*x
//                     + 63*(a0^2*b2^2 + 4*a1*a2*b0*b1 + a2^2*b0^2 + (2*a0*a2 + a1^2)*b1^2 + 2*(2*a0*a1*b1 + (2*a0*a2
//                     + 75*(a2^2*b1^2 + (2*a0*a2 + a1^2)*b2^2 + 2*(2*a1*a2*b1 + a2^2*b0)*b2)*x^3
//                     + 175*a2^2*b2^2*x^4
//                              )/x^2
//                     = 4/1575*(Y[0] + Y[1]*x + Y[2]*x^2 + Y[3]*x^3 + Y[4]*x^4)/x^2
void TTT(mpf_t resultat, mpf_t x, mpz_t a[3], mpz_t b[3]) {

    mpz_t a0;
    mpz_init_set(a0, a[0]);
    mpz_t a1;
    mpz_init_set(a1, a[1]);
    mpz_t a2;
    mpz_init_set(a2, a[2]);

    mpz_t b0;
    mpz_init_set(b0, b[0]);
    mpz_t b1;
    mpz_init_set(b1, b[1]);
    mpz_t b2;
    mpz_init_set(b2, b[2]);

    mpz_t Y[5];
    int i = 0;
    for (i = 0; i < 5; i++) {
        mpz_init(Y[i]);
    }

    mpz_t Z;
    mpz_init(Z);
```

```
////Y[0]

mpz_set_ui(Y[0], 175);
mpz_mul(Y[0], Y[0], a0);
mpz_mul(Y[0], Y[0], a0);
mpz_mul(Y[0], Y[0], b0);
mpz_mul(Y[0], Y[0], b0);

////Y[1]

mpz_mul(Z, a0, a0);
mpz_mul(Z, Z, b0);
mpz_mul(Z, Z, b2);
mpz_mul_ui(Z, Z, 2);

mpz_set(Y[1], Z);

mpz_mul(Z, a0, a0);
mpz_mul(Z, Z, b1);
mpz_mul(Z, Z, b1);

mpz_add(Y[1], Y[1], Z);

mpz_set_ui(Z, 4);
mpz_mul(Z, Z, a0);
mpz_mul(Z, Z, a1);
mpz_mul(Z, Z, b0);
mpz_mul(Z, Z, b1);

mpz_add(Y[1], Y[1], Z);

mpz_mul(Z, a0, a2);
mpz_mul_ui(Z, Z, 2);
mpz_addmul(Z, a1, a1);
mpz_t macro;
mpz_init_set(macro, Z);   //macro = 2*a0*a2 + a1^2
mpz_mul(Z, Z, b0);
mpz_mul(Z, Z, b0);

mpz_add(Y[1], Y[1], Z);

mpz_mul_ui(Y[1], Y[1], 75);

////Y[2]

mpz_mul(Z, a0, a1);
mpz_mul(Z, Z, b1);
mpz_mul_ui(Z, Z, 2);
mpz_addmul(Z, macro, b0);
mpz_mul(Z, Z, b2);
mpz_mul_ui(Z, Z, 2);

mpz_set(Y[2], Z);

mpz_mul(Z, b1, b1);
mpz_mul(Z, Z, macro);

mpz_add(Y[2], Y[2], Z);

mpz_mul(Z, a2, b0);
mpz_mul(Z, Z, Z);

mpz_add(Y[2], Y[2], Z);

mpz_mul(Z, a1, a2);
mpz_mul(Z, Z, b0);
mpz_mul(Z, Z, b1);
mpz_mul_ui(Z, Z, 4);

mpz_add(Y[2], Y[2], Z);

mpz_mul(Z, a0, b2);
mpz_mul(Z, Z, Z);

mpz_add(Y[2], Y[2], Z);

mpz_mul_ui(Y[2], Y[2], 63);


////Y[3]

mpz_mul(Z, a1, b1);
```

```
    mpz_mul_ui(Z, Z, 2);
    mpz_addmul(Z, a2, b0);
    mpz_mul(Z, Z, a2);
    mpz_mul(Z, Z, b2);
    mpz_mul_ui(Z, Z, 2);

    mpz_set(Y[3], Z);

    mpz_mul(Z, b2, b2);
    mpz_mul(Z, Z, macro);

    mpz_add(Y[3], Y[3], Z);

    mpz_mul(Z, a2, b1);
    mpz_mul(Z, Z, Z);

    mpz_add(Y[3], Y[3], Z);

    mpz_mul_ui(Y[3], Y[3], 75);

    ////Y[4]

    mpz_mul(Z, a2,  b2);
    mpz_mul(Z, Z, Z);
    mpz_mul_ui(Z, Z, 175);

    mpz_set(Y[4], Z);


    ////Y[0] + Y[1]*x + Y[2]*x^2 + Y[3]*x^3 + Y[4]*x^4

    mpf_t F;
    mpf_init(F);

    mpf_t RES;
    mpf_init(RES);

    mpf_set_z(F, Y[4]);
    mpf_set(RES, F);

    for(i = 0; i < 4; i++) {
        mpf_mul(RES, RES, x);
        mpf_set_z(F, Y[3-i]);
        mpf_add(RES, RES, F);
    }

    ////TTT(x, a, b)

    mpf_mul_ui(RES, RES, 4);
    mpf_div_ui(RES, RES, 1575);
    mpf_div(RES, RES, x);
    mpf_div(RES, RES, x);

    mpf_set(resultat, RES);


    mpz_clear(a0);
    mpz_clear(a1);
    mpz_clear(a2);
    mpz_clear(b0);
    mpz_clear(b1);
    mpz_clear(b2);
    mpz_clear(Z);
    for(i = 0; i < 5; i++) {
        mpz_clear(Y[i]);
    }

    mpf_clear(F);
    mpf_clear(RES);
    mpz_clear(macro);

}


/// norme := min TTT(x, a, b)  (a et b fix s)
/// s0 := argmin(TTT(x, a, b))
void normeL2(mpf_t s0, mpf_t norme, mpz_t a[3], mpz_t b[3]) {

    mpf_t x0;
    mpf_t x1;
    mpf_t x2;
    mpf_t x3;
```

```
mpf_init_set_d(x0, 0.000001);
mpf_init_set_d(x1, 1);
mpf_init_set_d(x2, 1);
mpf_init_set_d(x3, 2);

mpf_t T0;
mpf_t T1;
mpf_t T2;
mpf_t T3;

mpf_init(T0);
mpf_init(T1);
mpf_init(T2);
mpf_init(T3);

TTT(T0, x0, a, b);
TTT(T1, x1, a, b);
TTT(T2, x2, a, b);
TTT(T3, x3, a, b);


while(mpf_cmp(T2, T3) > 0) {
    mpf_set(x2, x3);
    mpf_set(T2, T3);
    mpf_add(x3, x3, x3);
    TTT(T3, x3, a, b);
}

mpf_t ecart;
mpf_init(ecart);
mpf_sub(ecart, x3, x0);

mpf_t F;
mpf_init(F);


while(mpf_cmp_d(ecart, 0.001) > 0) {
    mpf_div_ui(F, ecart, 3);
    mpf_add(x1, x0, F);
    TTT(T1, x1, a, b);
    mpf_add(x2, x1, F);
    TTT(T2, x2, a, b);

    int j = 0;
    TTT(F, x0, a, b);

    if (mpf_cmp(F, T1) > 0) {
        j = 1;
        mpf_set(F, T1);
    }
    if (mpf_cmp(F, T2) > 0) {
        j = 2;
        mpf_set(F, T2);
    }
    if (mpf_cmp(F, T3) > 0) {
        j = 3;
        mpf_set(F, T3);
    }

    if (j == 0) {
        mpf_set(x3, x1);
    }
    if (j == 1) {
        mpf_set(x3, x2);
    }
    if (j == 2) {
        mpf_set(x0, x1);
    }
    if (j == 3) {
        mpf_set(x0, x2);
    }
    mpf_sub(ecart, x3, x0);
}

mpf_sqrt(s0, x0);
mpf_sqrt(s0, s0);

TTT(norme, x0, a, b);

mpf_clear(x0);
mpf_clear(x1);
mpf_clear(x2);
mpf_clear(x3);
```

```
    mpf_clear(T0);
    mpf_clear(T1);
    mpf_clear(T2);
    mpf_clear(T3);
    mpf_clear(ecart);
    mpf_clear(F);

}

//x := exp(z)
void expo(mpf_t x, mpf_t z) {

    mpf_t x0;
    mpf_init_set_ui(x0, 0);

    mpf_t zz;
    mpf_init_set_ui(zz, 1);

    int i;
    for (i = 1; i < 100; i++ ) {
        mpf_add(x0, x0, zz);
        mpf_mul(zz, zz, z);
        mpf_div_ui(zz, zz, i);
    }

    mpf_set(x, x0);

    mpf_clear(x0);
    mpf_clear(zz);

}


// note := exp(alpha(a,2000))*exp(alpha(b,2000))*norme(a,b)      (cf. normeL2)
void notation(mpf_t note, mpz_t a[3], mpz_t b[3], mpf_t norme) {

    mpf_t alpha_a;
    double aa = get_alpha(a, 2, 2000);
    mpf_init_set_d(alpha_a, aa);

    mpf_t alpha_b;
    double bb = get_alpha(b, 2, 2000);
    mpf_init_set_d(alpha_b, bb);

    mpf_t X;
    mpf_init(X);
    mpf_sqrt(X, norme);

    mpf_t exp_a;
    mpf_init(exp_a);

    mpf_t exp_b;
    mpf_init(exp_b);

    expo(exp_a, alpha_a);
    expo(exp_b, alpha_b);

    mpf_mul(X, X, exp_a);
    mpf_mul(X, X, exp_b);

    mpf_set(note, X);

    mpf_clear(alpha_a);
    mpf_clear(alpha_b);
    mpf_clear(X);
    mpf_clear(exp_a);
    mpf_clear(exp_b);

}


//Etant donnes deux entiers N et k,
//teste k paires de polynomes quadratiques ayant un resultant egal    N,
//et affiche la paire ayant la meilleure notation,
//accompagne de sa torsion, norme, alpha(f), alpha(g) et du nombre premier p ayant permis de l'obtenir

void
twoquadratics (mpz_t N, unsigned long int k)
{
    unsigned int compteur = 0;
    unsigned long int p = 2;

    mpz_t P;
```

```
mpz_init_set_ui(P, p);

mpz_t a[3];
mpz_t b[3];
mpz_t a0[3];
mpz_t b0[3];

mpf_t no1;
mpf_init(no1);

mpf_t no;
mpf_init(no);

mpf_t s1;
mpf_init(s1);

mpf_t s0;
mpf_init(s0);

mpf_t alpha_a;
mpf_init(alpha_a);

mpf_t alpha_b;
mpf_init(alpha_b);

mpf_t note0;
mpf_init(note0);

mpf_t note1;
mpf_init(note1);

mpz_t P0;
mpz_init(P0);


int i;
for (i = 0; i < 3; i++ ) {
    mpz_init(a[i]);
    mpz_init(b[i]);
    mpz_init(a0[i]);
    mpz_init(b0[i]);
}

while(compteur < k) {
    while(mpz_kronecker(N, P) != 1) {
        mpz_nextprime(P, P);
    }
    p = mpz_get_ui(P);
    TQponctuel(N, p, a, b, 0);
    //gmp_printf ("\n\na = %Zd + %Zd * X + %Zd * X^2 \n", a[0], a[1], a[2]);
    //gmp_printf ("b = %Zd + %Zd * X + %Zd * X^2 \n\n", b[0], b[1], b[2]);
    normeL2(s1, no1, a, b);
    notation(note1, a, b, no1);
    if(compteur == 0) {
        mpz_set(a0[0], a[0]);
        mpz_set(a0[1], a[1]);
        mpz_set(a0[2], a[2]);
        mpz_set(b0[0], b[0]);
        mpz_set(b0[1], b[1]);
        mpz_set(b0[2], b[2]);
        mpf_set(s0, s1);
        mpf_set(no, no1);
        mpf_set(note0, note1);
    }
    //printf ("note = ");
    //printscientific(note1);
    //printf("\n");
    if (mpf_cmp(note0, note1) > 0) {
        mpz_set(a0[0], a[0]);
        mpz_set(a0[1], a[1]);
        mpz_set(a0[2], a[2]);
        mpz_set(b0[0], b[0]);
        mpz_set(b0[1], b[1]);
        mpz_set(b0[2], b[2]);
        mpf_set(s0, s1);
        mpf_set(note0, note1);
        mpz_set(P0, P);
    }
    TQponctuel(N, p, a, b, 1);
    //gmp_printf ("\na = %Zd + %Zd * X + %Zd * X^2 \n", a[0], a[1], a[2]);
    //gmp_printf ("b = %Zd + %Zd * X + %Zd * X^2 \n\n", b[0], b[1], b[2]);
    normeL2(s1, no1, a, b);
    notation(note1, a, b, no);
```

```
        if (mpf_cmp(note0, note1) > 0) {
            mpz_set(a0[0], a[0]);
            mpz_set(a0[1], a[1]);
            mpz_set(a0[2], a[2]);
            mpz_set(b0[0], b[0]);
            mpz_set(b0[1], b[1]);
            mpz_set(b0[2], b[2]);
            mpf_set(s0, s1);
            mpf_set(note0, note1);
            mpz_set(P0, P);
        }
        //printf ("note = ");
        //printscientific(note1);
        //printf("\n");
        mpz_nextprime(P, P);
        compteur += 2;
        //printf("\n%d\n", compteur);
    }

    printf("\n\nbest polynomials found:\n");
    gmp_printf ("a = %Zd + %Zd * X + %Zd * X^2 \n", a0[0], a0[1], a0[2]);
    gmp_printf ("b = %Zd + %Zd * X + %Zd * X^2 \n\n", b0[0], b0[1], b0[2]);

    printf ("note = ");
    printscientific(note0);
    gmp_printf ("\ns0 = %Ff\n", s0);
    double aaa = get_alpha(a0, 2, 2000);
    printf ("alpha(a) = %f\n", aaa);
    double bbb = get_alpha(b0, 2, 2000);
    printf ("alpha(b) = %f\n", bbb);
    normeL2(s0, no, a0, b0);
    printf ("norme = ");
    printscientific(no);
    printf("\n");
    gmp_printf ("P = %Zd\n", P0);


    mpf_clear(no1);
    mpf_clear(no);
    mpf_clear(s1);
    mpf_clear(s0);
    mpf_clear(alpha_a);
    mpf_clear(alpha_b);
    mpf_clear(note0);
    mpf_clear(note1);
    mpz_clear(P0);
    mpz_clear(P);

    for (i = 0; i < 3; i++ ) {
        mpz_clear(a[i]);
        mpz_clear(b[i]);
        mpz_clear(a0[i]);
        mpz_clear(b0[i]);
    }

}

void
usage ()
{
  fprintf (stderr, "Usage: twoquadratics [-k nnn] [N]\n");
  exit (EXIT_FAILURE);
}

int
main (int argc, char *argv[])
{
    mpz_t N;
    int k = 100;

    while (argc >= 2 && argv[1][0] == '-')
      {
        if (argc >= 3 && strcmp (argv[1], "-k") == 0)
          {
            k = atoi (argv[2]);
            argc -= 2;
            argv += 2;
          }
        else
          {
            fprintf (stderr, "Unknown option: %s\n", argv[1]);
            usage ();
          }
```

```
    }

  if (argc == 1) /* test on the c59 */
    mpz_init_set_str (N, "7164152076175143545513361647566709043406333222824787179542 9", 10);
  else
    mpz_init_set_str (N, argv[1], 10);

  twoquadratics (N, k);

  mpz_clear (N);

  return 0;
}
```