Factorisation par fractions continues

Margot Funk, Antoine Hugounet

Février 2021

Table des matières

1 Explication du programme

1.1 Architecture du programme

1.1.1 Terminologie

Enonçons pour commencer quelques définitions qui seront utiles pour décrire le code.

Définition 1.1. On dira qu'un couple (A_{n-1}, Q_n) est une *paire* (A, Q).

Définition 1.2. Un ensemble de paires (A, Q) indexé par $n_1, \ldots, n_r$ est dit *valide* si le produit $\prod_{i=1}^{r}(-1)^{n_i}Q_{n_i}$ est un carré (dans $\mathbb{Z}$ et non uniquement dans $\mathbb{Z}/N\mathbb{Z}$).

Définition 1.3. Si B est la base de factorisation utilisée par le programme, on désignera par l'expression *vecteur exposant associé à* Q_n le B-vecteur exposant $v_B((-1)^n Q_n)$.

1.1.2 Structure générale

Notre programme comprend deux étapes principales. La première consiste à générer, à partir du développement en fractions continues de $\sqrt{kN}$, des paires (A, Q) avec Q_n friable pour une base de factorisation préalablement déterminée. On associe à chaque Q_n ainsi produit son vecteur exposant `mpz_t exp_vect`. Ce vecteur permet de retenir les nombres premiers qui interviennent dans la factorisation de Q_n avec une valuation impaire. Dans le but d'augmenter le nombre de paires (A, Q) acceptées lors de cette étape, nous avons implémenté la *large prime variation*. Celle-ci permet d'accepter une paire si Q_n se factorise grâce aux premiers de la base de factorisation et à un grand facteur premier supplémentaire. Les fonctions de cette phase de collecte sont rassemblées dans le fichier `step_1.c`. Elles font appel, pour mettre en oeuvre la *large prime variation*, aux fonctions du fichier `lp_var.c`.

Ces données sont traitées lors de la seconde phase dans l'espoir de trouver un facteur non trivial de N. Il s'agit de trouver des ensembles valides de paires (A, Q) par pivot de Gauss sur la matrice dont les lignes sont formées des vecteurs exposants. Chaque ensemble valide est à l'origine d'une congruence de la forme $A^2 \equiv Q^2 \pmod{N}$ permettant potentiellement de trouver un facteur non trivial de N. Les fonctions de cette phase sont regroupées dans le fichier `step_2.c`.

Avant d'effectuer la première étape, il convient de se doter d'une base de factorisation. Ceci est permis par une des fonctions de `init_algo.c`. Ces dernières se chargent plus généralement de l'initialisation et du choix par défaut des paramètres.

Finalement, en mettant bout à bout les deux étapes, la fonction `contfract_factor` du fichier `fact.c` recherche un facteur non trivial de N et `print_results` affiche les résultats.

1.1.3 Entrées et sorties

Nous avons regroupé dans une structure `Params` les paramètres d'entrée de la fonction de factorisation, à savoir :

- `N` : le nombre à factoriser, supposé produit de deux grands nombres premiers.
- `k` : le coefficient multiplicateur.
- `n_lim` : le nombre maximal de paires (A, Q) que l'on s'autorise à calculer. Ce nombre prend en compte toutes les paires produites et non uniquement les paires avec Q_n friable ou résultant de la *large prime variation*.
- `s_fb` : la taille de la base de factorisation.
- `nb_want_AQp` : le nombre désiré de paires (A, Q) avec Q_n friable ou résultant de la *large prime variation*.
- des booléens indiquant si la *early abort strategy* ou la *large prime variation* doivent être utilisées et des paramètres s'y rapportant.

Le programme stocke dans une structure `Results` un facteur non trivial de `N` trouvé (si tel est le cas) ainsi que des données permettant l'analyse des performances de la méthode.

Remarque 1.4. L'efficatité de la méthode dépend du choix des paramètres ci-dessus. Pour avoir plus de latitude dans les tests, nous les considérons comme des paramètres d'entrée du programme. C'est pourquoi notre programme ne s'attèle pas à la factorisation complète d'un entier, qui aurait nécessité une sous-routine déterminant des paramètres optimaux en fonction de la taille de l'entier dont on cherche un facteur.

Remarque 1.5. Notre programme n'est pas supposé prendre en entrée un nombre admettant un petit facteur premier (inférieur aux premiers de la base de factorisation par exemple). En effet, comme il ne teste pas au préalable si `N` est divisible par de petits facteurs, il mettra autant de temps à trouver un petit facteur qu'un grand facteur.

1.2 Pivot de Gauss et recherche d'un facteur non trivial : `step_2.c`

Avant de nous pencher sur les détails de la phase de collecte, regardons l'implémentation de la seconde phase, qui aide à mieux comprendre la forme sous laquelle nous collectons les données.

1.2.1 Utilisation des données collectées

A l'issue de la première phase, on espère avoir collecté `nb_want_AQp` paires (A, Q) avec Q_n friable [1]. Le nombre réel de telles paires est stocké dans le champ `nb_AQp` d'une structure `Results`. Une paire (A, Q) collectée est caractérisée par :

— la valeur A_{n-1}

— la valeur Q_n

— le vecteur exposant associé à Q_n

— un vecteur historique (voir ci-contre)

Les données de ces `nb_AQp` paires sont stockées dans quatre tableaux : `mpz_t *Ans`, `mpz_t *Qns`, `mpz_t *exp_vects` et `mpz_t *hist_vects`. A un indice correspond une paire (A, Q) donnée.

Le vecteur historique sert à indexer les paires collectées pour former un analogue de la matrice identité utilisée pendant le pivot de Gauss. Plus précisément, `hist_vects[i]` est le vecteur $(e_{nb\_AQp-1}, \cdots, e_0)$ où $e_j = \delta_{ij}$.

A partir de ces quatre tableaux, la fonction `find_factor` cherche un facteur de `N`. Elle utilise pour cela les fonctions auxiliaires `gauss_elimination` et `calculate_A_Q`.

1.2.2 La fonction `gauss_elimination`

La fonction `gauss_elimination` effectue un pivot de Gauss sur les éléments de `mpz_t *exp_vects`, vus comme les vecteurs-lignes d'une matrice. Comme pour un pivot de Gauss classique, les calculs effectués sur les vecteurs exposants sont reproduits en parallèle sur la matrice identité, c'est-à-dire sur les éléments de `mpz_t *hist_vects`. Si le xor de deux vecteurs exposants donne le vecteur nul, cela signifie qu'une relation de dépendance a été trouvée. On inscrit alors dans un tableau l'indice de ce vecteur nul. Le vecteur historique dudit indice indique les paires (A, Q) de l'ensemble valide trouvé. La procédure que nous avons implémentée est décrite ci-dessus.

1. ou résultant de la *large prime variation* mais cela n'a aucune incidence sur les fonctions de cette partie.

Algorithme 1 : PIVOT DE GAUSS

Entrées : tableau EXP_VECTS$[0 \cdots nb_AQp - 1]$ des vecteurs exposants
tableau HIST_VECTS$[0 \cdots nb_AQp - 1]$ des vecteurs historiques

Sorties : HIST_VECTS$[0 \cdots nb_AQp - 1]$ après le pivot, le nombre nb_lin_rel de relations linéaires trouvées, LIN_REL_IND$[0 \cdots nb_lin_rel - 1]$ contenant les indices des lignes où une relation linéaire a été trouvée

créer tableau MSB_IND$[0 \cdots nb_AQp - 1]$
créer tableau LIN_REL_IND
$nb\_lin\_rel \leftarrow 0$

`/* Initialisation du tableau MSB_IND : Msb(x) renvoie 0 si x est nul, l'indice du bit de poids fort de x sinon. Les indices des bits sont numérotés de 1 à l'indice du bit de poids fort. */`

pour $i \leftarrow 0$ ***à*** $nb\_AQp - 1$ **faire**
    MSB_IND$[i] \leftarrow$ MSB(EXP_VECTS$[i]$)

pour $j \leftarrow MAX(MSB\_IND)$ ***à*** 1 **faire**
    $pivot \leftarrow \begin{cases} \min\left\{i \in \{0, nb\_AQp - 1\} \middle| \text{MSB\_IND}[i] = j\right\} \\ \emptyset \text{ si pour tout } i \in \{0, nb\_AQp - 1\} \text{ MSB\_IND}[i] \neq j \end{cases}$
    si $pivot \neq \emptyset$ **alors**
        pour $i \leftarrow pivot + 1$ ***à*** $nb\_AQp - 1$ **faire**
            si $MSB\_IND[i] = j$ **alors**
                EXP_VECTS$[i] \leftarrow$ EXP_VECTS$[i] \oplus$ EXP_VECTS$[pivot]$
                HIST_VECTS$[i] \leftarrow$ HIST_VECTS$[i] \oplus$ HIST_VECTS$[pivot]$
                MSB_IND$[i] \leftarrow$ MSB(EXP_VECTS$[i]$)
                si $EXP\_VECTS[i] = 0$ **alors**
                    ajouter i au tableau LIN_REL_IND
                    $nb\_lin\_rel \leftarrow nb\_lin\_rel + 1$

retourner $HIST\_VECTS[0 \cdots nb\_AQp - 1]$, $LIN\_REL\_IND[0 \cdots nb\_lin\_rel - 1]$, nb_lin_rel

1.2.3 La fonction `calculate_A_Q`

Algorithme 2 : Extraction de racine carrée

Entrées : Des entiers $Q_1, \cdots, Q_r \in \mathbb{Z}$ tels que $\prod_{i=1}^{r} Q_i$ est un carré
Sorties : $\sqrt{\prod_{i=1}^{r} Q_i} \pmod N$

$Q \leftarrow 1$
$R \leftarrow Q_1$
pour $i \leftarrow 2$ ***à*** r **faire**
  $X \leftarrow \text{pgcd}(R, Q_i)$
  $Q \leftarrow XQ \pmod N$
  $R \leftarrow \frac{R}{X} \cdot \frac{Q_i}{X}$
$X \leftarrow \sqrt{R}$
$Q \leftarrow XQ \pmod N$
retourner Q

Pour démontrer la correction de l'algorithme, on peut utiliser l'invariant de boucle suivant : $Q\sqrt{R.Q_i \cdots Q_r} \pmod N = \sqrt{\prod_{i=1}^{r} Q_i} \pmod N$. La conservation de l'invariant découle de l'égalité

$$Q\sqrt{R.Q_i \cdots Q_r} \pmod N = (QX \bmod N)\sqrt{\frac{R}{X}\frac{Q_i}{X}Q_{i+1} \cdots Q_r} \pmod N.$$

1.3 Collecte des paires (A, Q) : `step_1.c` et `lp_var.c`

Décrivons à présent la phase de collecte des données. Concernant les vecteurs historiques, il suffit d'initialiser à la fin de la collecte `hist_vects[i]` pour `0 <= i < nb_AQp`. C'est ce que fait la fonction `init_hist_vects`. La collecte des autres données requiert un peu plus d'explications.

1.3.1 La fonction `create_AQ_pairs`

Sachant que seules les paires (A, Q) dont on a pu factoriser Q_n nous intéressent pour la seconde phase, nous avons décidé de ne stocker que celles-ci. Ce choix a en outre un avantage : étant donné un nombre `nb_want_AQp` représentant le nombre voulu de telles paires, il est possible d'arrêter le développement en fraction continue dès que ce nombre est atteint. Cela évite d'avoir à stocker toutes les paires (A, Q), pour ensuite sélectionner celles qui nous intéressent, en courant le risque d'en avoir trop ou pas assez.

Ce choix amène à avoir une grande fonction, en l'occurence `create_AQ_pairs`, qui au fur à mesure du développement de $\sqrt{kN}$ en fraction continue, teste si le Q_n qui vient

d'être calculé est factorisable. Si c'est le cas, on crée son vecteur exposant et ajoute les données de la paire aux tableaux `Ans`, `Qns` et `exp_vects`. Pour ce faire, la fonction utilise les sous-routines `is_Qn_factorisable` et `init_exp_vect`.

1.3.2 La *early abort strategy*

La fonction `is_Qn_factorisable` teste si un Q_n est friable [2] par divisions successives avec les premiers de la base de factorisation. Un moyen d'améliorer les performances de la méthode est de décider de ne pas poursuivre les divisions successives si après un nombre `eas_cut` de divisions la partie non factorisée de Q_n est trop grande (supérieure à une borne `eas_bound_div` proportionnelle à la borne déjà connue $\sqrt{kN}$).

1.3.3 La *large prime variation*

Etant donnée une base de factorisation $B = \{p_1, \cdots, p_m\}$, la *large prime variation* consiste à accepter lors de la collecte, non seulement des Q_n B-friables mais aussi des Q_n produits d'un entier B-friable et d'un entier lp_n inférieur à p_m^2. On dira que Q_n est *presque friable* et l'on appelera *grand premier (large prime)* le premier lp_n en question.

Pour que des Q_n presque friables soient exploitables, il faut qu'ils aient un grand premier lp en commun. En effet, si on trouve deux entiers presque friables $Q_{n_1} = X_{n_1} lp$ et $Q_{n_2} = X_{n_2} lp$, on peut former une nouvelle paire (A, Q) avec laquelle on peut travailler pour chercher une congruence de carrés.

Remarquons pour cela qu'on a les conguences :

$$\begin{cases} A_{n_1-1}^2 \equiv (-1)^{n_1} X_{n_1} lp \pmod N \\ A_{n_2-1}^2 \equiv (-1)^{n_2} X_{n_2} lp \pmod N \end{cases}$$

En les multipliant, on obtient :

$$(A_{n_1-1}A_{n_2-1})^2 \equiv \underbrace{(-1)^{n_1+n_2} X_{n_1} X_{n_2}}_{\substack{\text{associé au vecteur exposant} \\ v_B\left((-1)^{n_1}X_{n_1}\right)+v_B\left((-1)^{n_2}X_{n_2}\right)}} \underbrace{lp^2}_{\text{carré qui ne pose pas problème}} \pmod N$$

On forme donc la nouvelle paire $(A_{n_1-1}A_{n_2-1} \pmod N, Q_{n_1}Q_{n_2})$ associée au vecteur exposant $v_B\Big((-1)^{n_1}X_{n_1}\Big) + v_B\Big((-1)^{n_2}X_{n_2}\Big)$. Elle sera traitée lors de la deuxième phase exactement de la même manière que les paires « classiques ».

En pratique, pour repérer les paires qui ont le même grand premier, nous constituons au fur et à mesure de la collecte une liste chainée dont les noeuds stockent les

2. ou presque friable, voir paragraphe suivant.

données d'une paire dont le Q_n est presque friable (les entiers Q_n, A_{n-1}, le vecteur exposant et le grand premier associé à Q_n). Nous maintenons cette liste triée par taille des grands premiers. Lorsque que survient un Q_n presque friable, il est repéré par la fonction `is_Qn_factorisable` qui fournit également son grand premier lp. La liste chainée est alors parcourue pour savoir si l'on a déjà rencontré ce lp. Deux cas se présentent alors. Si lp est absent de la liste, on crée à la bonne place un noeud. Si lp est déjà présent dans la liste, au lieu de rajouter un noeud, on utilise le noeud possédant ce lp pour obtenir une nouvelle paire (A, Q) selon la méthode énoncée plus haut et ajoute ses composantes aux tableaux `Ans`, `Qns` et `exp_vects`. La fonction `insert_or_elim_lp` se charge de cela.