A parallel multi-modular algorithm for computing Lagrange resolvents

Nicolas Rennert

IP6, MFI, Université Paris VI, 4 place Jussieu, F-75252, Paris Cedex 05, France

Accepted 9 June 2000

Abstract

The aim of this paper is to exploit the algorithms of paper Experimental Math. 8 (1999) in order to produce a new algebraic method for computing efficiently absolute Lagrange resolvent, a fundamental tool in constructive algebraic Galois theory. This article is composed of two parts.

The main idea of the first part is to break up the calculation of absolute resolvent into smaller computations. Since a multi-resolvent is a factor of a resolvent, the whole resolvent may be computed by means of several multi-resolvents.

The idea of the second part is that an irreducible polynomial over \( \mathbb{Z} \) might be reducible over \( \mathbb{Z}/p\mathbb{Z} \) for certain integer \( p \). So the first part can be applied and then, the Chinese remainder theorem allows to lift up the resolvent over \( \mathbb{Z} \).

© 2003 Published by Elsevier Ltd

Keywords: Algorithm; Lagrange resolvents

1. Introduction

Since (Lagrange, 1869) resolvents are central tools for effective Galois theory because their computation are sufficient to find the Galois group of polynomials (Arnaudiès and Valibouze, 1997) (the direct problem) and can be useful to compute a polynomial of some fixed Galois group (the reverse problem) (Valibouze, 1995). Fast algorithms for computing resolvents improve Galois group recognition. Several kinds of methods have been developed to compute resolvents. Some are based on numerical computations, such as the one describe by Stauduhar (1973) found in many systems.

E-mail address: nicolas.rennert@oorrsg.org (N. Rennert).

1 GALOIS project of the symbolic computational UMS-CNRS MEDICIS.
(GALP Eichenlaub and Olivier, 1996); this method has now a version based on p-adic approximation (Yokoyama, 1997; Geissler and Klüners, in press). Pure algebraic methods have the advantage to avoid numerical instability problem. They can be very efficient, for example, the algorithms based on symmetric functions for some special invariants (Valibouze, 1989; Casperson and McKay, 1994). In the article (Rennert and Valibouze, 1999) it is established that in the category of general algebraic algorithm (for any given invariant), the “Cauchy moduli” algorithm is among the fastest. However it appears that for some invariant this last algorithm reaches its limit quite early, for degree around 6 or 7.

Based on the fact that multi-resolvents (a kind of resolvent for factored polynomial) can be computed efficiently, we expose an algorithm for calculating resolvents founded on multi-resolvent computations suitable for the case of reducible polynomials (see Algorithm 3.1). In an other part, we show how to apply Algorithm 3.1 in order to compute resolvent for any polynomial with integer coefficients. This leads to an efficient and parallel algorithm for computing absolute resolvent.

In this article we consider as given,

\[ R \text{ a gcd domain;} \]
\[ x_1, \ldots, x_n \text{ and } x, n + 1 \text{ indeterminates over } R; \]
\[ f \text{ a polynomial of } R[x] \text{ such that } \deg(f) = n; \]
\[ \Theta \text{ a polynomial of } R[x_1, \ldots, x_n]. \]

The goal of this article is to expose an algorithm for computing the univariate polynomial \( L(\Theta, f) \), which is the Lagrange resolvent of \( f \) by \( \Theta \) (see Definition 2.7).

We adopt the following notations,

\[ K \text{ is the fraction field of } R; \]
\[ \hat{K} \text{ denotes an algebraic closure of } K; \]
\[ S_n \text{ denotes the symmetric group of degree } n; \]
\[ R[x_1, \ldots, x_n] \text{ denotes the ring of polynomials in variables } x_1, \ldots, x_n; \]
\[ \Omega_f \in K^n \text{ denotes a fixed vector of the } n \text{ roots of the polynomial } f. \]

2. Definitions

The definitions found in this article are based on the article Valibouze (1999) where a new point of view on Galois theory is exposed.

2.1. Actions of groups

**Definition 2.1.** Let \( \Psi \) be a polynomial of \( R[x_1, \ldots, x_n] \), \( L \) be a subgroup of the symmetric group \( S_n \), and \( H \) be a subgroup of \( L \).

The permutation \( \sigma \) of the symmetric group \( S_n \) acts on the polynomial \( \Psi \) by permuting its variables:

\[ (\sigma \cdot \Psi)(x_1, \ldots, x_n) = \Psi(x_{\sigma(1)}, \ldots, x_{\sigma(n)}). \]

**Definition 2.2.** The orbit \( L \cdot \Psi \) of \( \Psi \) under the action of the group \( L \) is the set of the actions of the permutations of \( L \) on \( \Psi \):

\[ L \cdot \Psi = \{ \tau \cdot \Psi \mid \tau \in L \}. \]
Definition 2.3. The polynomial $\Psi$ is called an $L$-primitive $H$-invariant if the group $H$ is the stabilizer of the polynomial $\Psi$ in the group $L$:

$\text{Stab}_L(\Psi) = H$.

Definition 2.4. Let $\tau_1 H, \ldots, \tau_e H$ be the left classes of $L \mod H$. The set $\{\tau_1, \ldots, \tau_e\}$ is called a left transversal of $L$ modulo $H$.

2.2. Ideal and Galois group

Definition 2.5. The ideal $I_{\Omega_f}$ of the $\Omega_f$-relations (over $R$) is defined by

$I_{\Omega_f} = \{ R \in \mathbb{R}[x_1, \ldots, x_n] \mid R(\Omega_f) = 0 \}$.

Definition 2.6. The Galois group $G_{\Omega_f}$ of $\Omega_f$ (over $R$), is defined by

$G_{\Omega_f} = \{ \sigma \in \mathfrak{S}_n \mid (\forall R \in I_{\Omega_f}) \sigma . R \in I_{\Omega_f} \}$.

2.3. Lagrange resolvent

Definition 2.7. Let $L$ be a subgroup of the Galois group $G_{\Omega_f}$ (over $R$), $H$ be a subgroup of $L$, $\Theta \in \mathbb{R}[x_1, \ldots, x_n]$ be an $L$-primitive $H$-invariant. The $L$-relative resolvent of $\Omega_f$ by $\Theta$ is a univariate polynomial which roots are the polynomials of the orbit $L. \Theta$ evaluated in $\Omega_f$,

$L_{\Theta, L, \Omega_f} = \prod_{\Psi \in L. \Theta} (T - \Psi(\Omega_f))$. \hspace{1cm} (2.1)

Remark 2.8. The Galois theory assures that the resolvent $L_{\Theta, L, \Omega_f}$ belongs to $\mathbb{R}(T)$.

When $L = \mathfrak{S}_n$, the resolvent $L_{\Theta, \Omega_f}$ is written $L_{\Theta, f}$ and called (absolute) resolvent of $f$ by $\Theta$. The full definition of the multi-resolvent is given in Section 3.1.

3. Computing resolvents using multi-resolvents

In this section, we suppose that the polynomial $f$ is reducible over $\mathbb{R}$ and that the polynomial $\Theta$ is an $\mathfrak{S}_n$-primitive $H$-invariant for a given group $H$.

The polynomials $f_1, \ldots, f_k$ are the factors of $f$ over $\mathbb{R}$ with degree $f_i = d_i > 1$ ($i \in [1, k]$).

We suppose that, in the vector $\Omega_f$, the roots of $f_i$ are before those of $f_{i+1}$ for each $i$ in $[1, k - 1]$.

The group $S$ is defined by the group product $\mathfrak{S}_{d_1} \times \cdots \times \mathfrak{S}_{d_k}$.

Definition 3.1. The absolute multi-resolvent of $f_1, \ldots, f_k$ by $\Theta$ denoted by $L_{\Theta, f_1, \ldots, f_k}$ is the resolvent $L_{\Theta, S, \Omega_f}$. The multi-resolvent is well defined because the Galois group $G_{\Omega_f}$ is included in the group $S$.

Notice that in multi-resolvent what only matters is the order of the polynomials, how we order the root of each polynomial doesn’t modify the polynomial.
Since multi-resolvents are much faster to compute than resolvents (Rennert and Valibouze, 1999), we would like to compute resolvent by the use of multi-resolvent.

3.1. The multi-resolvent is a factor of a resolvent

By Definition 3.1, the multi-resolvent \( L_{\Theta, f_1, \ldots, f_k} \) is the polynomial given by
\[
\prod_{\varphi \in S, \Theta} (T - \varphi(\Omega_{f}))
\]
and we want to compute the resolvent \( L_{\Theta, f} \) given by
\[
\prod_{\varphi \in S_n, \Theta} (T - \varphi(\Omega_{f})).
\]
The multi-resolvent \( L_{\Theta, O_f} \) is a factor of the absolute resolvent \( L_{\Theta, f} \) because the group \( S \) is a subgroup of \( S_n \).

3.2. How to get the whole resolvent?

By Section 3.1, we can compute a factor of the absolute resolvent \( L_{\Theta, f} \) by the mean of the multi-resolvent \( L_{\Theta, f_1, \ldots, f_k} \). A natural idea is to express its other factors as multi-resolvents of conjugate of the invariant \( \Theta \). For this reason, we need to introduce the double transversal.

Definition 3.2. Given a group \( L \) and two subgroups \( S \) and \( H \) of \( L \), not necessarily distinct, the set of the elements \( SxH \), where \( x \) is some fixed element of \( L \), is called a double coset of \( L \) by \( S \) and \( H \). The set of all the cosets of \( L \) by \( S \) and \( H \) is written \( H \backslash G / L \). A set \( \{\tau_1, \ldots, \tau_m\} \) of representatives of the double cosets \( H \backslash G / L \) is called a double transversal of \( L \) by \( S \) and \( H \).

Our problem is to find a set of permutations \( T \) such that the product of the multi-resolvent \( \prod_{\tau \in T} L_{\tau, \Theta, f_1, \ldots, f_k} \) equals the absolute resolvent \( L_{\Theta, f} \).

By the use of double transversal, the following lemma gives such a set \( T \).

Lemma 3.3. The double transversal \( T = S \backslash S_n / H \) is such that
\[
\prod_{\tau \in T} L_{\tau, \Theta, f_1, \ldots, f_k} = L_{\Theta, f}.
\]

Proof. Let \( T = S \backslash S_n / H \).
\[
\{S. (\tau, \Theta) | \tau \in T\} = S. (T. \Theta) = (ST). \Theta = (S(S \\backslash S_n / H)). \Theta = (S_n / H). \Theta = S_n. \Theta. \quad \Box
\]

So using Lemma 3.3 we can easily produce a parallel algorithm in order to compute the resolvent by the use of multi-resolvent.

3.3. Langrange resolvent with multi-resolvent algorithm

Algorithm 3.1. Resolvent\((R, \Theta, F)\)
inputs: $\mathcal{R}$, a gcd-domain,
$\Theta$ and $H$, $\Theta$ a polynomial of $\mathcal{R}[x_1, \ldots, x_n]$ which is $\mathfrak{S}_n$-primitive $H$-invariant,
$F = \{f_1, \ldots, f_k\}$, a list of polynomials of $\mathcal{R}[x]$ of degree greater than 0.

output: The absolute resolvent $L_{\Theta, f_1,\ldots,f_k}$

Begin
For $i$ in $1..k$

d_i \leftarrow \text{degree}(f_i)$
End for
$S \leftarrow \mathfrak{S}_{d_1} \times \cdots \times \mathfrak{S}_{d_k}$
$T \leftarrow \text{representatives}(S \setminus \mathfrak{S}_n/H)$
$m \leftarrow \text{cardinal}(T)$
For $i$ in $1..m$
\[
\tau \leftarrow T[i]
\]
\[
r_i \leftarrow \text{multiResolvent}([f_1, \ldots, f_k], \tau, \Theta)
\]
End for
Return$(r_1 \cdots r_m)$
End

Where:

- $\text{representatives}(S \setminus \mathfrak{S}_n/H)$ returns a double transversal of $S \setminus \mathfrak{S}_n/H$ as a vector of permutations;
- $\text{multiResolvent}([f_1, \ldots, f_k], \Theta)$ return the multi-resolvent $L_{\Theta, f_1,\ldots,f_k}$;
- $\text{ParDo}$ realizes a parallelized loop.

**Remark 3.4.** In order to simplify the factorization of the resolvent, the product $r_1 \cdots r_m$ returned by the Algorithm 3.1 can be retrieved as developed or as partially factored.

### 4. Absolute resolvent for irreducible polynomials with integer coefficients

It is a well known fact that all irreducible polynomials over $\mathbb{Z}$ are reducible over $\mathbb{F}_p$ for many integer $p$. Therefore Algorithm 3.1 can be applied in order to calculate the resolvent $L_{\Theta, f}$ over $\mathbb{F}_p$. After computing resolvents $L_{\Theta, f}$ over $\mathbb{F}_p$ for a sufficient number of primes $p$, the resolvent $L_{\Theta, f}$ over $\mathbb{Z}$ can be recovered using the Chinese Remainder Algorithm.

In this section, we now consider $f$ as a polynomial of $\mathbb{Z}[x]$. We take $p$ a prime integer such that $f$ is reducible polynomial over $\mathbb{F}_p[x]$.

To distinguish polynomials over $\mathbb{Z}$ from polynomials over $\mathbb{F}_p$, we will adopt the following notations: for a polynomial $g$ of $\mathbb{Z}[x]$, $\bar{g}$ denotes $g$ mod $p$ and $\tilde{f}_1, \ldots, \tilde{f}_k$ denotes the factors of $f$ in $\mathbb{F}_p[x]$.

#### 4.1. Computing the absolute resolvent $L_{\Theta, f}$ in $\mathbb{F}_p[x]$

We can apply Algorithm 3.1 with $\mathcal{R} = \mathbb{F}_p$ and $F = \{\tilde{f}_1, \ldots, \tilde{f}_k\}$ in order to compute the absolute resolvent $L_{\Theta, f}$, because $\mathbb{F}_p$ is a gcd domain. In order to apply Algorithm 3.1 it is necessary to compute multi-resolvents over finite fields.
4.2. Computing multi-resolvents in $\mathbb{F}_p[x]$

The computations over finite fields often require adapted algorithms because non-zero characteristic brings new property. In this section we discuss how to adapt the multi-resolvent algorithms based on resultant and Cauchy moduli to the case of finite field. For more informations please consult Rennert and Valibouze (1999) and Lehobey (1997).

The resolvent $L_{\Theta, f}$ (Algorithm 6.2 in Rennert and Valibouze, 1999) or the multi-resolvent $L_{\Theta, f_1, ..., f_k}$ (Algorithm 7.1 in Rennert and Valibouze, 1999) calls the function “$n$th Root” whose role is to compute the polynomial $g_1/s$ for some polynomial $g$ and integer $s \geq 1$. Lehobey exposes in (Lehobey, 1997) an algorithm that compute the $n$th root of a monic polynomial that is very fast. This last algorithm has particular restriction over finite fields (see Section 6.3 of Lehobey, 1997), the prime integer $p$ must be greater than the degree of $g$ and shall not divide $s$. We can deduce that if the characteristic of the field is greater than the degree of the resolvent the “$n$th Root” algorithm is always valid. In the rest of the article, we place ourself under this hypothesis.

4.3. Recovering the resolvent $L_{\Theta, f}$

Let $p_1, ..., p_s$ be $s$ prime integers, $g$ be an unknown polynomial of $\mathbb{Z}[x]$ such that $g \mod p_i$ can be computed for each $i \in [1, s]$. Let us consider $\text{lift}_{p_1, ..., p_s}(g)$ the unique polynomial of $\mathbb{Z}[x]$ such that:

\[
\text{lift}_{p_1, ..., p_s}(g) \mod p_i = g \mod p_i \text{ for each } i \in [1, s] \\
2|C| \leq p_1 \cdots p_s \text{ for each coefficient } C \text{ of } g.
\]

This polynomial is computable by the Chinese Remainder Theorem (Geddes et al., 1993).

If the product $p_1 \cdots p_s$ is larger than twice the maximum of the absolute value of the coefficients of $g$, the polynomial $\text{lift}_{p_1, ..., p_s}(g)$ equals $g$ otherwise the $\text{lift}_{p_1, ..., p_s}(g)$ can be different. To determinate the number of necessary prime integers, we have two possibilities. The first one is to find an upper bound on the coefficients of the absolute resolvent $L_{\Theta, f}$ (see Section 4.3.1). The second one is to use the classical method in computer algebra that consists on computing $\text{lift}_{p_1, ..., p_m}(L_{\Theta, f})$ for a number $m$ of prime integer such that:

\[
\text{lift}_{p_1, ..., p_m}(L_{\Theta, f}) = \text{lift}_{p_1, ..., p_{m-1}}(L_{\Theta, f}).
\]

The polynomial $\text{lift}_{p_1, ..., p_m}(L_{\Theta, f})$ has a great probability to equal the resolvent $L_{\Theta, f}$.

4.3.1. Bounding the coefficients of the resolvent $L_{\Theta, f}$

Let $g$ be a polynomial of $\mathbb{Z}[x]$ given by $g = x^n + a_1x^{n-1} + \cdots + a_n$ and $\Psi$ be a polynomial of $\mathbb{Z}[x_1, ..., x_d]$ given by $\Psi = \sum_{i=1, \ldots, d} b_i x_1^{d_{i,1}} \cdots x_d^{d_{i,d}}$.

$C_{\Psi}$ is the “largest” coefficient of the polynomial $\Psi$.

\[C_{\Psi} = \max |b_i|.
\]

$C_g$ is the “largest” coefficient of the polynomial $g$.

\[C_g = \max |a_i|.
\]
$R_g$ is the “largest” root of the polynomial $g$,

$$R_g = \max_{g(\alpha) = 0} \|\alpha\|.$$ 

$D_\Psi$ is the “largest” of the monomial degree of the polynomial $\Psi$,

$$D_\Psi = \max_{1 \leq i \leq j} \left| \sum_{1 \leq l \leq n} d_{l,i} \right|.$$ 

By Definition 2.7, the searched resolvent $L_{\Theta,f}$ is given by:

$$\prod_{\Psi \in \mathfrak{S}_n : \Theta} (T - \Psi(\Omega)).$$

Thus, the coefficient of the resolvent $L_{\Theta,f}$ can be bounded by:

$$C_{L_{\Theta,f}} \leq \prod_{\Psi \in \mathfrak{S}_n : \Theta} C_\Psi (R_f)^{D_\Psi}. \quad (4.4)$$

Since the polynomial $\Psi$ in (4.4) is a conjugate of the polynomial $\Theta$, $D_\Psi = D_\Theta$ and $C_\Psi = C_\Theta$

$$C_{L_{\Theta,f}} \leq \left( (C_\Theta R_f)^{D_\Theta} \right)^{[\mathfrak{S}_n : H]}, \quad (4.5)$$

where $[\mathfrak{S}_n : H]$ is the index of $H$ in $\mathfrak{S}_n$. For simplicity of the formulae let us write $d = \deg(L_{\Theta,f}) = [\mathfrak{S}_n : H]$, $R_f$ can be bound by $C_f + 1$ (Mignotte, 1982), therefore, Eq. (4.5) can be rewritten as:

$$C_{L_{\Theta,f}} \leq (C_\Theta (C_f + 1))^{d D_\Theta}. \quad (4.6)$$

To have a rough estimation of the number of necessary prime integers we will consider the binary size of the bound:

$$\log(C_{L_{\Theta,f}}) \leq d D_\Theta \log(C_\Theta (C_f + 1)).$$

Suppose that the size of the coefficients of the polynomial $f$ is about the same as the size of the chosen prime, that the size $C_\Theta$ is negligible, we can see that we need about $d D_\Theta$ prime integers.

For high degree resolvent (over 100) this bound seems useless, we prefer use the probabilistic method.

5. Algorithm

We can now write a parallel algorithm for computing absolute resolvents for polynomials of $\mathbb{Z}[x]$. 

Algorithm 5.1. Resolvent($\Theta, f$)

inputs: $\Theta$, a polynomial of $\mathbb{Z}[x_1, \ldots, x_n]$ $\Theta$-primitive $H$-invariant, $f$, a polynomial of $\mathbb{R}[x]$.

output: The absolute resolvent $L_{\Theta,f}$

Begin

1. $lp \leftarrow \emptyset$ – the list of the prime integers
2. $lr \leftarrow \emptyset$ – the list of the resolvent $\tilde{L}_{\Theta,f}$ already computed

While Not enoughPrime($lp, lr, f, \Theta$) ParDo

1. $p \leftarrow$ selectNewPrime($lp$)
2. $r \leftarrow$ resolvent($F_p, \Theta, f$)
3. $lp \leftarrow \{p\} \cup lp$
4. $lr \leftarrow \{r\} \cup lr$

End While

Return(Chinese($lr, lp$))

End

Where:

- the function enoughPrime ($lp, lr, f, \Theta$) can either use the previous bound (Section 4.3.1) or the probabilistic method (Section 4.3);
- the function selectNewPrime ($lp$) choose a prime integer that do not belong to the set $lp$, usually the program take the largest prime integer that fits in a machine word integer and after the previous prime, it might be clever to choose prime integers that gives small factors of $f$;
- the function resolvent ($F_p, \Theta, f$) computes $L_{\Theta,f} \mod p$ by Algorithm 3.1 using warnings of Section 4.1;
- the function Chinese ($lr, lp$) computes lift($lp(L_{\Theta,f})$).

6. A practical example

A small example is sufficient to illustrate the algorithms of the article. Since the calculations can lead to wide expressions, the unnecessary ones are not printed.

Example 6.1. Let us choose $f = x^6 - 243x^2 + 729$ and $\Theta = x_3x_5x_4x_6(x_3x_5 + x_4x_6) + x_2x_4x_5x_6(x_2x_4 + x_5x_6) + x_4x_5x_1x_6(x_4x_5 + x_1x_6) + x_2x_3x_3x_6(x_2x_3 + x_3x_6) + x_1x_3x_5x_6(x_1x_3 + x_5x_6) + x_2x_6x_1x_5(x_2x_6 + x_1x_5) + x_3x_4x_2x_6(x_3x_4 + x_2x_6) + x_1x_4x_3x_6(x_1x_4 + x_3x_6) + x_1x_2x_4x_6(x_1x_2 + x_4x_6) + x_2x_3x_1x_6(x_2x_3 + x_1x_6) + x_2x_3x_4x_5(x_2x_3 + x_4x_5) + x_3x_4x_1x_5(x_3x_4 + x_1x_5) + x_1x_4x_2x_5(x_1x_4 + x_2x_5) + x_1x_2x_3x_5(x_1x_2 + x_3x_5) + x_1x_3x_2x_4(x_1x_3 + x_2x_4).$

According to the nomenclature of Butler and McKay (1983) the stabilizer of $\Theta$ in $S_6$ is the transitive group $T_{14}$.

Let us first describe Algorithm 3.1 as it is employed in Algorithm 5.1 for the calculation of the resolvent $L_{\Theta,f}$. Let us fix $p = 4294967291$ and compute the polynomial $\tilde{L}_{\Theta,f}$.

Over $F_{4294967291}$ the polynomial $f$ can be factored as:
\[ f = (x + 3453399294)(x + 4242233739)(x + 841567997) \times (x + 52733552)(x^2 + 1182059616). \]

\( \tilde{f}_i \) is the \( i \)th factor of \( f \) for \( i \in [1, s] \):

\[ \tilde{f}_1 = x + 3453399294, \ldots, \tilde{f}_5 = x^2 + 1182059616. \]

Thus the group \( S \) of Algorithm 3.1 induced by this factorization is

\[ S = Id \times Id \times Id \times Id \times S_2 \]

and the set \( T \) is

\[ T = \text{Representative} \left( S \setminus \mathbb{S}_6 / T_{14} \right) = [Id, (4, 5), (4, 6, 5)]. \]

The set \( T \) is computed with the computer system GAP (Schönhert et al., 1995; GAP, 1997).

\[ \bar{L}_{\Theta, \tilde{f}} = \bar{L}_{\Theta, \tilde{f}_1, \ldots, \tilde{f}_5}, \bar{L}_{\Theta, \tilde{f}_1, \ldots, \tilde{f}_5}, \bar{L}_{\Theta, \tilde{f}_1, \ldots, \tilde{f}_5}, \bar{L}_{\Theta, \tilde{f}_1, \ldots, \tilde{f}_5}. \]

The computations of the three multi-resolvents \( \bar{L}_{\Theta, \tilde{f}_1, \ldots, \tilde{f}_5}, \bar{L}_{\Theta, \tilde{f}_1, \ldots, \tilde{f}_5} \) and \( \bar{L}_{\Theta, \tilde{f}_1, \ldots, \tilde{f}_5} \) are done using the Cauchy moduli resolvent program presented in Rennert and Valibouze (1999).

In order to compute the resolvent \( \bar{L}_{\Theta, f} \) over \( \mathbb{Z} \), Algorithm 5.1 computes resolvents \( \bar{L}_{\Theta, \tilde{f}} \) over some prime integers. Here the following prime integers: 4294967111, 4294966591, 4294966447 are sufficient to recover the resolvent over \( \mathbb{Z} \) using the Chinese Remainder Algorithm.

7. Conclusion

The Example 6.1 executed sequentially, takes less than 1 minute compare to 11 minutes with the method presented in Rennert and Valibouze (1999). Even when the implementation is not parallel, the gain is already substantial. This let us think that this algorithm has great potentiality. Some aspects not treated in this article will have to be considered. In effect the fact that we can computed the resolvents \( \bar{L}_{\Theta, \tilde{f}} \) in a partially factored form can be exploited to reduce the complexity of the factorization of the resolvent \( \bar{L}_{\Theta, f} \), using techniques similar those found in Encarnación (1997).

Algorithm 5.1 can be generalized in order compute relative resolvents using idea contented in Aubry and Valibouze (in press).

Acknowledgement

I would like to thank A. Hulpke for his help on group theory and GAP usage.

References


[http://www.math.u-bordeaux.fr/A2X/Logiciels.html](http://www.math.u-bordeaux.fr/A2X/Logiciels.html).