Fast Rational Function Reconstruction [*]

Sara Khodadad
School of Computing Science,
Simon Fraser University,
Burnaby, B.C. V5A 1S6, CANADA.
skhodada@cecm.sfu.ca.

Michael Monagan
Department of Mathematics,
Simon Fraser University,
Burnaby, B.C. V5A 1S6, CANADA.
mmonagan@cecm.sfu.ca.

ABSTRACT

Let F be a field and let f and g be polynomials in $F[t]$ satisfying $\deg f > \deg g$. Recall that on input of f and g the extended Euclidean algorithm computes a sequence of polynomials (s_i, t_i, r_i) satisfying $s_i f + t_i g = r_i$. Thus for i with $\gcd(t_i, f) = 1$, we obtain rational functions $r_i/t_i \in F(t)$ satisfying $r_i/t_i \equiv g \pmod f$.

In this paper we modify the fast extended Euclidean algorithm to compute the smallest r_i/t_i, that is, an r_i/t_i minimizing $\deg r_i + \deg t_i$. This means that in an output sensitive modular algorithm when we are recovering rational functions in $F(t)$ from their images modulo $f(t)$ where $f(t)$ is increasing in degree, we can recover them as soon as the degree of f is large enough and we can do this fast.

We have implemented our modified fast Euclidean algorithm for $F = \mathbb{Z}_p$, p a word sized prime, in Java. Our fast algorithm beats the ordinary Euclidean algorithm around degree 200. This has application to polynomial gcd computation and linear algebra over $\mathbb{Z}_p(t)$.

Categories and Subject Descriptors: I.1.2 [Symbolic and Algebraic Manipulation]: Algorithms – Algebraic algorithms;

General Terms: Algorithms, Theory.

Keywords: Rational Reconstruction, Fast Euclidean Algorithm, Modular Algorithms.

1. INTRODUCTION

Rational number reconstruction, originally developed by Paul Wang in [16], (see [2] or [4] for an accessible reference), has found many applications in computer algebra. It enables us to design efficient *modular algorithms* for computing with polynomials, vectors and matrices over $\mathbb{Q}$. Such algorithms first solve a problem modulo a sufficiently large integer m which is usually a product of primes or a power of a prime. Then they apply rational reconstruction to recover the rational numbers in the solution from their images modulo m. The same basic strategy can also be used to recover fractions in $F(t)$ from their image modulo a polynomial $f(t) \in F[t]$ where F is a field. Some applications where rational reconstruction has been used include polynomial gcd computation over $\mathbb{Q}(\alpha)$, solving linear systems over $\mathbb{Q}$ and Gröbner basis computation over $\mathbb{Q}$.

A key advantage of rational reconstruction is that it enables us to make modular algorithms "output sensitive", that is, to make the the size of the modulus m needed, and hence overall efficiency, depend on the size of the rationals in the output and not on bounds for their size which might be much larger. For example, consider the problem of computing the monic gcd g of two polynomials f_1 and f_2 in $L[x]$ where L is a number field. In [3], Encarnacion modified Langemyr and MacCallum's modular GCD algorithm [6] to use rational reconstruction to make it output sensitive. Because g is often much smaller in size than f_1 and f_2, Encarnacion's algorithm is often much faster in practice.

Wang's Algorithm

Let $n/d \in \mathbb{Q}$ with $d > 0$ and $\gcd(n, d) = 1$. Let $m \in \mathbb{Z}$ with $m > 0$ and $\gcd(m, d) = 1$. Suppose we have computed $u = n/d \bmod m$ and we want to recover the rational n/d. Recall that extended Euclidean algorithm (EEA), on input of m and u with $m > u \geq 0$, computes a sequence of triples $(s_i, t_i, r_i) \in \mathbb{Z}^3$ for $i = 0, 1, \ldots, l, l+1$ satisfying $r_{l+1} = 0$ and $s_i m + t_i u = r_i$. It does this by initializing $(r_0, s_0, t_0) = (m, 1, 0)$ and $(r_1, s_1, t_1) = (u, 0, 1)$ and computing

$$(r_{i+1}, s_{i+1}, t_{i+1}) = (r_{i-1} - q_i r_i, s_{i-1} - q_i s_i, t_{i-1} - q_i t_i)$$

for $i = 1, 2, \ldots, l$ where q_i is the quotient of r_{i-1} divided by r_i. Observe that $s_i m + t_i u = r_i$ implies $r_i/t_i \equiv u \pmod m$ for all i with $\gcd(m, t_i) = 1$. In [16], Wang observed that if $m > 2|n|d$ then the rational $n/d = r_i/t_i$ for some $0 \leq i \leq l+1$. In fact, it is the r_i/t_i satisfying $r_{i-1} > |n| \geq r_i$, that is, we just need to compute up to the first remainder less than or equal to $|n|$.

One way to use Wang's observations to recover the rational number n/d in the output from its image u modulo m is as follows. First bound the size of n and d, that is, compute $N \geq |n|$ and $D \geq d$. Then solve the problem modulo a sequence of primes $p_1, p_2, \ldots$ satisfying $m > 2ND$ where $m = p_1 \times p_2 \times \ldots$. Then run the Euclidean algorithm until $r_{i-1} > N \geq r_i$, and output r_i/t_i after checking that $\gcd(t_i, m) = 1$.

However, as remarked earlier, the bounds are often much

[*]Supported by NSERC of Canada and the MITACS NCE of Canada

too big. To make a modular algorithm output sensitive we let m increase in size and *periodically* apply rational reconstruction as follows. Given the image u of the rational n/d modulo m, Wang computes $N = D = \lfloor\sqrt{m/2}\rfloor$ and runs the Euclidean algorithm with input $m > u$ stopping when $r_{i-1} > N \geq r_i$. One then checks if $|t_i| \leq D$ and $\gcd(t_i, m) = 1$. If yes then we output r_i/t_i else rational reconstruction "fails". Thus Wang's algorithm succeeds in reconstructing n/d when m becomes bigger than $2\max(n^2, d^2)$.

If one uses the ordinary Euclidean algorithm, the complexity of Wang's algorithm is $O(\log^2 m)$. In 2002 Pan and Wang in [13] modified the fast Euclidean algorithm of Schönhage [14] to solve the rational number reconstruction problem in time $O(\mathsf{M}(k)\log k)$ where $k = \log m$ is the length of the modulus m and $\mathsf{M}(k)$ is the cost of multiplying integers of length k. The authors did not implement their algorithm and remarked during their presentation at ISSAC 2002 that the algorithm might not be practical. In 2005 Lichtblau in [7] implemented a variation on the fast Euclidean algorithm for rational number reconstruction for Mathematica and found that it is practical. In fact, Steel (see [15]) had already implemented fast rational number reconstruction in Magma version 2.8 in 2000.

Maximal Quotient Rational Reconstruction

There is an inefficiency in Wang's approach because of the choice of $N = D = \lfloor\sqrt{m/2}\rfloor$. This choice means we are using half of the bits of m to recover the numerator and half for the denominator. To recover n/d, we require $m > 2|n|d$ but this choice for N and D means the modulus $m > 2\max(n^2, d^2)$. This is efficient if the numerator n and denominator d are of the same length. But if $|n| \gg d$ or $|n| \ll d$, it requires m to be up to twice as long as is necessary. This inefficiency was noted by Monagan in [11]. In particular, for gcd problems in $L[x]$, Monagan has observed that the denominators in g are often much smaller than numerators.

Monagan in [11] observed that if $m \gg 2|n|d$ then with high probability (we make some remarks about the probability in the conclusion) there will be only one small rational r_i/t_i in the Euclidean algorithm, namely n/d. In fact, if m is just a few bits longer than $2|n|d\log_2 m$, the smallest rational will be n/d with high probability. Thus another way to solve the rational reconstruction problem is to simply select and output the smallest r_i/t_i. How do we do this without explicitly multiplying $r_i \times t_i$? Monagan observed that if the size of the rational r_i/t_i is small compared with m, that is, $|r_i t_i| \ll m$ then $q_i = \lfloor r_{i-1}/r_i\rfloor$ is necessarily large, indeed q_i satisfies $\frac{m}{3} < q_i r_i |t_i| \leq m$. Hence, it is sufficient to select the rational r_i/t_i corresponding to the largest quotient q_i. Moreover, since the quotients are available and they are mostly very small integers, this selection is efficient.

In this way, it does not really matter whether n is much longer or much shorter than d, for as soon as m is a few bits longer than $2|n|d\log_2 m$, we can select n/d from the r_i/t_i with high probability. If m is a product of primes and one is using the Chinese remainder theorem, one saves up to half the number of primes. Thus in an application where the size of the numerators might be much larger or smaller than the size of the denominators, Monagan's algorithm is preferred.

Monagan's algorithm, like Wang's algorithm, is also a simple modification of the extended Euclidean algorithm, and thus also has complexity $O(\log^2 m)$ if the ordinary extended Euclidean algorithm is used. Just as Pan and X. Wang modified the fast Euclidean algorithm to accelerate Wang's algorithm, can Monagan's algorithm also be accelerated? In this paper we answer this question in the affirmative. We show how to modify the fast Euclidean algorithm to output the smallest rational r_i/t_i without increasing the asymptotic time complexity of the fast Euclidean algorithm. The key reason this is possible is that the fast Euclidean algorithm, which does not compute all remainders r_i explicitly, can be designed to compute all quotients q_i explicitly.

Rather than modifying the fast Euclidean algorithm for $\mathbb{Z}$, we modify the fast Euclidean algorithm for $\mathbb{Z}_p[t]$ where p is a prime to recover the rational function for $\mathbb{Z}_p(t)$ of least degree. We call our algorithm FMQRFR for fast maximal quotient rational function reconstruction. We have implemented it in Java. In comparing it to an implementation using the ordinary extended Euclidean algorithm for $\mathbb{Z}_p[t]$ we found that the fast Euclidean algorithm beats the ordinary extended Euclidean algorithm at around degree 200. In order to achieve such a result, one must implement fast multiplication in $\mathbb{Z}_p[t]$ carefully. For this we have implemented an "in-place" version of Karatsuba's algorithm (see Maeder [9]) so that fast multiplication in $\mathbb{Z}_p[t]$ already beats classical multiplication at degree 50.

Our paper is organized as follows. In section 2 we describe the maximal quotient rational reconstruction algorithm for $F[x]$. In section 3 we describe the fast extended Euclidean algorithm (FEEA) for $F[x]$. Our presentation of the FEEA follows the presentation given by von zur Gathen and Gerhard in [4]. We give timings for our implementation of the FEEA for $F = \mathbb{Z}_p$ where p is a word size prime, comparing it with the ordinary extended Euclidean algorithm. In section 4 we show how to modify the FEEA to compute the smallest rational function r_i/t_i. We also show how to accelerate Wang's algorithm for $\mathbb{Z}_p[t]$ using the FEEA to compute the rational function r_i/t_i satisfying $\deg r_{i-1} > (\deg f)/2 \geq \deg r_i$. We have implemented both algorithm and we compare their efficiency. In section 5 we make some remarks about the failure probability of our algorithm.

2. MAXIMAL QUOTIENT RATIONAL FUNCTION RECONSTRUCTION

Let F be a field. A rational function $n/d \in F(x)$ is said to be in *canonical form* if $\mathrm{lc}(d) = 1$ and $\gcd(n, d) = 1$. Let $f, g \in F[x]$ with $\deg f > \deg g$. Let r_i and t_i be the elements of the ith row of the Extended Euclidean Algorithm (EEA) with inputs f and g. Then any rational function n/d with $n = r_i/\mathrm{lc}(t_i)$ and $d = t_i/\mathrm{lc}(t_i)$ satisfies $n/d \equiv g \mod f$, provided that $\gcd(f, t_i) = 1$. Moreover, if n/d is a canonical form solution to $n/d \equiv g \mod f$ satisfying $\deg n + \deg d < \deg f$, then there exists some row j in the EEA for inputs f and g such that $n = r_j/\mathrm{lc}(t_j)$ and $d = t_j/\mathrm{lc}(t_j)$. Thus the EEA with inputs f and g generates all rational functions n/d (up to scalar multiples in F) satisfying $n/d \equiv g \mod f$, $\gcd(f, d) = 1$ and $\deg n + \deg d < \deg f$. Refer to [4, Lemma 5.15] for the proof.

If degree bounds $N \geq \deg n$ and $D \geq \deg d$ satisfying $N + D < \deg f$ are known, then the rational function n/d is uniquely determined by running the EEA on inputs f and g. But we do not always know the values of N and D in advance. In this section we will present an efficient algorithm that with high probability finds the correct solution for $\deg f > \deg n + \deg d + 1$. The following example illustrates how our algorithm works.

Example 2.1. *Consider* $f = \prod_{i=5}^{12}(x-i)$ *and* $g = 10x^7 + x^6 + 2x^5 + 10x^4 + 12x^3 + 7x^2 + 12x + 8$ *in* $\mathbb{Z}_{13}[x]$. *The Extended Euclidean Algorithm with inputs f and g yields the following table.*

i	$\deg r_i$	$\deg t_i$	$\deg r_i + \deg t_i$	$\deg q_i$
1	7	0	7	1
2	6	1	7	1
3	5	2	7	1
4	2	3	5	3
5	1	6	7	1
6	0	7	7	1

The data in the table suggest that we simply return a rational function r_i/t_i *where* $\deg r_i + \deg t_i$ *is minimal. As illustrated in the table,* r_4/t_4 *has minimal total degree of* 5. *Notice that* r_4/t_4 *also corresponds to the quotient* q_4 *of maximal degree* 3. *The reason for this is easily explained by the following lemma.*

Lemma 2.2. *Let F be a field and* $f, g \in F[x]$. *In the EEA for f and g we have*

$$\deg r_i + \deg t_i + \deg q_i = \deg f$$

for $1 \le i \le l$ *where l is the total number of division steps in the EEA for inputs f and g.*

Proof. We know $\deg t_i = \deg f - \deg r_{i-1}$, thus

$$\deg r_i + \deg t_i + \deg q_i =$$
$$\deg r_i + (\deg f - \deg r_{i-1}) + \deg r_{i-1} - \deg r_i = \deg f.$$

□

The algorithm presented at the end of this section selects an (r_i, t_i) of minimal total degree as the output. Later, when we modify the algorithm to use the fast Euclidean algorithm, this selection cannot be done this way because the remainders, the r_i, are not explicitly computed in the fast Euclidean algorithm. Instead, we make the selection based on a quotient q_i of maximal degree.

The following lemma states that when $\deg f$ is large enough then there would only be one pair of (r_j, t_j) such that $\deg r_j + \deg t_j$ is minimal.

Lemma 2.3. *Let F be a field, and* $n, d \in F[x]$ *with* $\mathrm{lc}(d) = 1$ *and* $\gcd(n, d) = 1$. *Let f, g be two polynomials in* $F[x]$ *satisfying* $\gcd(f, d) = 1$ *and* $g = n/d \bmod f$. *Let j denote the index of a quotient with maximal degree in the Extended Euclidean Algorithm with inputs f and g. If* $\deg f > 2(\deg n + \deg d)$ *then j is unique,* $n = r_j$ *and* $d = t_j$.

Proof. As discussed in the beginning of this section, since $\deg f > \deg n + \deg d$ then in the Extended Euclidean Algorithm with inputs f and g there exists an index j such that $r_j/t_j = n/d$. According to Lemma 2.2 we have $\deg q_j > 1/2 \deg f$. On the other hand, we know $\sum_{i=1}^{l} \deg q_i = \deg f - \deg r_l \le \deg f$ where l is the total number of division steps. This implies that q_j is the only quotient with maximal degree and if $\gcd(r_j, t_j) = 1$ then $n = r_j$ and $d = t_j$. □

Maximal Quotient RFR Algorithm (MQRFR)
Input: $f, g \in \mathbb{Z}_p[x]$ with $\deg f > \deg g$, and $T \in \mathbb{N}$
Output: Either $n, d \in \mathbb{Z}_p[x]$ satisfying $n/d \equiv g \bmod f$, $\mathrm{lc}(d) = 1$, $\gcd(n, d) = 1$, and $\deg n + \deg d + T < \deg f$, or FAIL implying no solution exists

```
1. if $g = 0$ then
       if $\deg f \ge T$ then
           return $(0, 1)$
       else
           return FAIL

2. $(r_0, r_1) \leftarrow (f, g)$
   $(t_0, t_1) \leftarrow (0, 1)$
   $(n, d) \leftarrow (r_1, t_1)$

3. while $r_1 \ne 0$ do
       if $\deg n + \deg d > \deg r_1 + \deg t_1$ then
           $(n, d) \leftarrow (r_1, t_1)$
       $q \leftarrow r_0 \operatorname{quo} r_1$
       $(r_0, r_1) \leftarrow (r_1, r_0 - q r_1)$
       $(t_0, t_1) \leftarrow (t_1, t_0 - q t_1)$

4. if $\deg n + \deg d + T \ge \deg f$ or $\gcd(n, d) \ne 1$ then
       return FAIL

5. return $(n/\mathrm{lc}(d), d/\mathrm{lc}(d))$
```

This algorithm is a simple modification of the (half) extended Euclidean algorithm. It's complexity is known to be quadratic in the degree of f.

3. THE FAST EUCLIDEAN ALGORITHM

In 1971 Schönhage in [14] presented a fast integer GCD algorithm with time complexity $O(n \log^2 n \log\log n)$. An asymptotically fast rational number reconstruction algorithm based on Schönhage's algorithm was presented by Pan and Wang in [13]. Before that Allan Steel had implemented in Magma a fast rational number reconstruction algorithm based on the half-gcd algorithm presented in Montgomery's PhD thesis [12] for polynomials in $F[x]$. Currently, Mathematica v. 5.0 and Magma v. 2.10 both have a fast GCD and fast rational number reconstruction. Maple v. 10 is using the GMP integer arithmetic package which has fast integer multiplication and division but no fast integer GCD yet.

Assuming a multiplication algorithm of time complexity $O(n \log^a n)$ is available for polynomials of degree n in $F[x]$, in 1973 Moenck in [10] adapted Schoñhage's algorithm into an $O(n \log^{a+1} n)$ algorithm for polynomial GCD computation in $F[x]$. In 1980 Brent, Gustavson, and Yun in [1] gave two speedups for Moenck's algorithm. They also pointed out (but did not prove) a generalization of Moenck's algorithm. Later in 1992, Montgomery in his PhD thesis [12] independently stated and proved a similar generalization of Moenck's algorithm with some of the same speedups.

In this section we describe the Fast Euclidean Algorithm and in the next section we show how to modify it to compute the smallest r_i/t_i fast. Our presentation follows that of von zur Gathen and Gerhard in [4].

Let F be a field and $r_0, r_1 \in F[x]$ with $\deg r_0 \ge \deg r_1$. Let

$$\rho_{i+1} r_{i+1} = r_{i-1} - q_i r_i,$$
$$\rho_{i+1} s_{i+1} = s_{i-1} - q_i s_i,$$
$$\rho_{i+1} t_{i+1} = t_{i-1} - q_i t_i,$$

for $1 \leq i \leq l$, be the results of the Extended Euclidean Algorithm for inputs r_0 and r_1, where $s_0 = t_1 = 1$, $s_1 = t_0 = 0$ and $r_{l+1} = 0$. We let ρ_i denote the leading coefficient of the ith remainder. Let $R_i = Q_i \dots Q_1 R_0$, for $1 \leq i \leq l$, where

$$Q_i = \left[\begin{array}{cc} 0 & 1 \\ 1/\rho_{i+1} & -q_i/\rho_{i+1} \end{array}\right], \quad R_0 = \left[\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array}\right]$$

in $F[x]^{2\times 2}$. Then it can be easily proved by induction on i that

$$R_i = \left[\begin{array}{cc} s_i & t_i \\ s_{i+1} & t_{i+1} \end{array}\right].$$

This matrix is of great importance in the design of the Fast Extended Euclidean Algorithm.

Let $f = f_n x^n + f_{n-1}x^{n-1} + \ldots + f_0 \in F[x]$ and $f_n \neq 0$. The truncated polynomial $f \upharpoonright k$ is defined by

$$f \upharpoonright k = f \operatorname{quo} x^{n-k} = f_n x^k + f_{n-1}x^{k-1} + \ldots + f_{n-k},$$

for $k \in \mathbb{Z}$. The polynomial $f \upharpoonright k$ is of degree k for $k \geq 0$ and its coefficients are the $k+1$ highest coefficients of f. The pairs (f, g) and (f^*, g^*) *coincide up to* k if

$$f \upharpoonright k = f^* \upharpoonright k,$$
$$g \upharpoonright (k - (\deg f - \deg g)) = g^* \upharpoonright (k - (\deg f^* - \deg g^*)),$$

where $f, g, f^*, g^* \in F[x]\backslash\{0\}$, $\deg f \geq \deg g$, $\deg f^* \geq \deg g^*$ and $k \in \mathbb{Z}$.

Following [4], the positive integer $\eta_{f,g}(k)$ is defined for any $k \in \mathbb{N}$ and $f, g \in F[x]$ by

$$\eta_{f,g}(k) = \max_{0 \leq j \leq l}\{j : \sum_{i=1}^{j} m_i \leq k\},$$

where $m_i = \deg q_i$ and l denotes the number of division steps in the Euclidean algorithm with inputs f and g. The following lemma implies that the first $\eta_{f,g}(k)$ results of the Euclidean Algorithm only depend on the top part of the inputs, which is the basic idea leading to a fast GCD algorithm.

Lemma 3.1. *[4, Lemma 11.3] Let $k \in \mathbb{N}$, $h = \eta_{r_0,r_1}(k)$ and $h^* = \eta_{r_0^*,r_1^*}(k)$, with r_0, r_1, r_0^*, r_1^* monic polynomials in $F[x]$. If (r_0, r_1) and (r_0^*, r_1^*) coincide up to $2k$ and $k \geq \deg r_0 - \deg r_1$, then*

1. *$h = h^*$,*
2. *$q_i = q_i^*$ for $1 \leq i \leq h$,*
3. *$\rho_i = \rho_i^*$ for $2 \leq i \leq h$,*

where $q_i, q_i^* \in F[x]$ and $\rho_i, \rho_i^* \in F$ are defined by

$$r_{i-1} = q_i r_i + \rho_{i+1} r_{i+1} \qquad (1 \leq i \leq l), \qquad r_{l+1} = 0,$$
$$r_{i-1}^* = q_i^* r_i^* + \rho_{i+1}^* r_{i+1}^* \qquad (1 \leq i \leq l^*), \qquad r_{l^*+1}^* = 0.$$

Refer to [4] for a detailed proof of this lemma. To improve the efficiency of the EEA a divide-and-conquer algorithm, called *Fast Extended Euclidean Algorithm*, is designed based on the above lemma. Von zur Gathen and Gerhard in [4, Ch. 11] present Schönhage's Fast Extended Euclidean Algorithm for polynomials in $F[x]$, however, the algorithm presented in the book needs some minor corrections. At our request the authors sent us a corrected version of their algorithm which is described below. Though, we have removed some outputs unnecessary for our purposes.

Fast Extended Euclidean Algorithm (FEEA)
Input: r_0 and r_1 two *monic* polynomials in $F[x]$ with $n_0 = \deg r_0 > n_1 = \deg r_1 \geq 0$ and $k \in \mathbb{N}$ with $n_0/2 \leq k \leq n_0$
Output: $h = \eta_{r_0,r_1}(k) \in \mathbb{N}$, $\rho_{h+1} \in F$, $R_h = \left[\begin{array}{cc} s_h & t_h \\ s_{h+1} & t_{h+1} \end{array}\right]$

```
1. if $r_1 = 0$ or $k < n_0 - n_1$ then
       return 0, 1, $\left[\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array}\right]$
   else if $n_0 <$ cutoff then
       return EEA($r_0$, $r_1$, $k$)

2. $k_1 \leftarrow \lfloor k/2 \rfloor$
   $r_0^* \leftarrow r_0 \upharpoonright 2k_1, r_1^* \leftarrow r_1 \upharpoonright (2k_1 - (n_0 - n_1))$
   $j - 1$, $\rho_j^*$, $R_{j-1}^* \leftarrow$FEEA($r_0^*$, $r_1^*, k_1$)

3. compute $\rho_j, R_{j-1}, r_{j-1}, r_j$ and $n_j = \deg r_j$
   (precise computing instructions follow)

4. if $r_j = 0$ or $k < n_0 - n_j$ then
       return $j - 1$, $\rho_j$, $R_{j-1}$

5. $q_j \leftarrow r_{j-1} \operatorname{quo} r_j$
   $\rho_{j+1} \leftarrow \operatorname{lc}(r_{j-1} - q_j r_j)$
   $r_{j+1} \leftarrow (r_{j-1} - q_j r_j)/\rho_{j+1}$
   $n_{j+1} \leftarrow \deg r_{j+1}$
   $R_j \leftarrow \left[\begin{array}{cc} 0 & 1 \\ 1/\rho_{j+1} & -q_j/\rho_{j+1} \end{array}\right] R_{j-1}$

6. $k_2 \leftarrow k - (n_0 - n_j)$
   $r_j^* \leftarrow r_j \upharpoonright 2k_2$, $r_{j+1}^* \leftarrow r_{j+1} \upharpoonright (2k_2 - (n_j - n_{j+1}))$
   $h - j$, $\rho_{h+1}^*$, $S^* \leftarrow$FEEA($r_j^*, r_{j+1}^*, k_2$)

7. compute $\rho_{h+1}, S, r_h$ and $r_{h+1}$

8. return $h$, $\rho_{h+1}$, $SR_j$
```

As illustrated above, besides the two monic polynomials r_0 and r_1, the algorithm gets a third input $k \in \mathbb{N}$. This input is used as an upper bound for the sum of the degrees of quotients computed in each recursive call to the algorithm. That is, if $h = \eta_{r_0,r_1}(k)$ denotes the index of the last computed quotient, then we will have

$$\sum_{i=1}^{h} \deg q_i \leq k < \sum_{i=1}^{h+1} \deg q_i.$$

The FEEA divides the problem into two subproblems of almost the same size, i.e., the sum of the degrees of the quotients computed in each recursive call is at most $k/2$. Note that in this algorithm all elements of the EEA, i.e., the q_i's, s_i's and t_i's, are computed except the remainders, the r_i's. However, having s_h and t_h as the entries of the second row of the output matrix R_h one can easily compute a single remainder r_h by writing $r_h = s_h r_0 + t_h r_1$. It is not hard to see that $r_h = \gcd(r_0, r_1)$, if we set $k = \deg r_0$.

According to Lemma 3.1, ρ_j^* is not necessarily equal to ρ_j, and thus R_{j-1} and R_{j-1}^* are not equal either. Therefore we use the following relations

$$\left[\begin{array}{c} r_{j-1} \\ \tilde{r}_j \end{array}\right] = R_{j-1}^* \left[\begin{array}{c} r_0 \\ r_1 \end{array}\right], \quad R_{j-1} = \left[\begin{array}{cc} 1 & 0 \\ 0 & 1/\operatorname{lc}(\tilde{r}_j) \end{array}\right] R_{j-1}^*,$$
$$\rho_j = \rho_j^* \operatorname{lc}(\tilde{r}_j), \quad r_j = \tilde{r}_j/\operatorname{lc}(\tilde{r}_j),$$

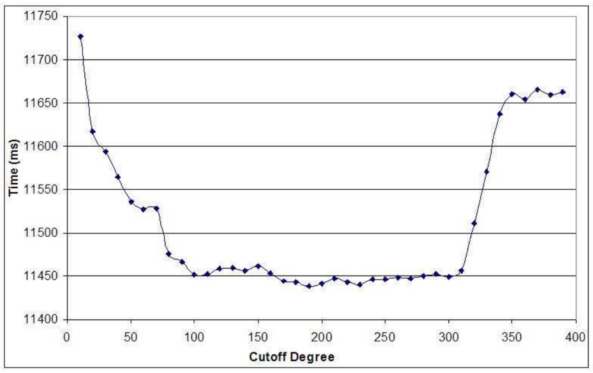
in step 3 to compute ρ_j, R_{j-1}, r_{j-1} and r_j. Similar computations are performed in step 7 to compute ρ_{h+1}, S, r_h and

r_{h+1}. The algorithm has a time complexity of $O(\mathsf{M}(k)\log k)$, where $\mathsf{M}(k)$ denotes the number of field operation required to multiply two univariate polynomials of degree k. Refer to [5, p. 27] for a detailed cost analysis and a detailed proof of correctness of the algorithm.

We have implemented the FEEA for polynomials in $F[x] = \mathbb{Z}_p[x]$ in Java. We used Karatsuba's algorithm for univariate polynomial multiplication in our implementation which is of time complexity $O(n^{\log_2 3})$ for polynomials of degree n. The algorithm is not effective in practice for polynomials of low degree. We use the classical multiplication method for polynomials of degree less than 50 and switch to Karatsuba's when the input polynomials have a degree greater than 50. The following table includes timings (in milliseconds) for our implementation of the Classical and Karatsuba multiplication algorithms over $\mathbb{Z}_p[x]$, where p is a 15 bit prime and both input polynomials have degree n. As illustrated below, the timings of Karatsuba's algorithm increase by a factor close to 3 as the degree doubles which confirms that our implementation is of time complexity $O(n^{\log_2 3})$.

n	Karatsuba(ms)	Classical(ms)
128	0.34	0.38
256	0.98	1.40
512	2.93	5.40
1024	8.93	21.62
2048	26.48	84.43
4096	79.78	345.67
8192	245.04	1375.42

It turns out that in practice the EEA performs better than the FEEA as well for polynomials of low degree. Our implementation of the FEEA beats the EEA when $\deg r_0 = 200$. Thus we have used 200 as the value of the *cutoff* in step 1 of the FEEA. The following figure illustrates the timings (in ms) of the FEEA on two random polynomials of degree 10000 for different *cutoff* degrees.



Our Java implementation of the EEA accepts 3 inputs and returns the same outputs as the FEEA. We are using the "monic" Euclidean algorithm. The following table includes our timings for the EEA and the FEEA on random polynomials of degree n. It shows that we see a significant speedup by $n = 1000$.

n	EEA(ms)	FEEA(ms)	r_1	r_2
1000	373.80	295.63	0.00052	1.26
2000	1427.18	942.83	0.00050	1.51
4000	5602.18	2972.08	0.00049	1.88
8000	22295.47	9588.76	0.00048	2.33
16000	88766.90	31278.50	0.00049	2.84
32000	354085.71	99273.77	0.00048	3.54

$r_1 = \text{FEEA}/(n^{\log_2 3}\log n)$, $r_2 = \text{EEA}/\text{FEEA}$

4. MQRFR USING FEEA

To make the MQRFR algorithm more efficient we use the FEEA instead of the EEA. As pointed out before, the FEEA does not compute the intermediate remainders, but it does compute all the quotients. Also s_i and t_i are available as the entries of the first row of R_i. Thus according to lemma 2.2 instead of selecting r_i and t_i such that $\deg r_i + \deg t_i$ is minimal, we can return q_i the quotient with maximal degree along with corresponding values of s_i and t_i. The remainder r_i is then obtained from s_i and t_i using two long multiplications ($r_i = s_i f + t_i g$). The following algorithm presents the FEEA modified to return the quotient of maximal degree.

Modified FEEA (MFEEA)
Input: r_0 and r_1 two *monic* polynomials in $F[x]$ with $n_0 = \deg r_0 > n_1 = \deg r_1 \geq 0$ and $k \in \mathbb{N}$ with $n_0/2 \leq k \leq n_0$
Output: $h = \eta_{r_0,r_1}(k) \in \mathbb{N}$, $\rho_{h+1} \in F$, $R_h = \begin{bmatrix} s_h & t_h \\ s_{h+1} & t_{h+1} \end{bmatrix}$, $q_{\max}$, $s_{\max}$, $t_{\max}$

```
1. if $r_1 = 0$ or $k < n_0 - n_1$ then
       return 0, 1, $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, 1, 1, 0
   else if $n_0 <$ cutoff then
       return EEA($r_0$, $r_1$, $k$)

2. $k_1 \leftarrow \lfloor k/2 \rfloor$
   $r_0^* \leftarrow r_0 \upharpoonright 2k_1, r_1^* \leftarrow r_1 \upharpoonright (2k_1 - (n_0 - n_1))$
   $j-1, \rho_j^*, R_{j-1}^*, q_{\max}, s_{\max}, t_{\max} \leftarrow$ MFEEA($r_0^*, r_1^*, k_1$)

3. compute $\rho_j, R_{j-1}, r_{j-1}, r_j$ and $n_j = \deg r_j$

4. if $r_j = 0$ or $k < n_0 - n_j$ then
       return $j-1$, $\rho_j$, $R_{j-1}$, $q_{\max}$, $s_{\max}$, $t_{\max}$

5. $q_j \leftarrow r_{j-1} \operatorname{quo} r_j$
   $\rho_{j+1} \leftarrow \operatorname{lc}(r_{j-1} - q_j r_j)$
   $r_{j+1} \leftarrow (r_{j-1} - q_j r_j)/\rho_{j+1}$
   $n_{j+1} \leftarrow \deg r_{j+1}$
   $R_j \leftarrow \begin{bmatrix} 0 & 1 \\ 1/\rho_{j+1} & -q_j/\rho_{j+1} \end{bmatrix} R_{j-1}$
   if $\deg q_j > \deg q_{\max}$ then
       $q_{\max}, s_{\max}, t_{\max} \leftarrow q_j, R_j[1,1], R_j[1,2]$

6. $k_2 \leftarrow k - (n_0 - n_j)$
   $r_j^* \leftarrow r_j \upharpoonright 2k_2$, $r_{j+1}^* \leftarrow r_{j+1} \upharpoonright (2k_2 - (n_j - n_{j+1}))$
   $h-j, \rho_{h+1}^*, S^*, q_{\max}^*, s_{\max}^*, t_{\max}^* \leftarrow$ MFEEA($r_j^*, r_{j+1}^*, k_2$)
   if $\deg q_{\max}^* > \deg q_{\max}$ then
       $q_{\max} \leftarrow q_{\max}^*$
       $\begin{bmatrix} s_{\max} & t_{\max} \end{bmatrix} \leftarrow \begin{bmatrix} s_{\max}^* & t_{\max}^* \end{bmatrix} R_j$

7. compute $\rho_{h+1}, S, r_h$ and $r_{h+1}$

8. return $h$, $\rho_{h+1}$, $SR_j$, $q_{\max}$, $s_{\max}$, $t_{\max}$
```

As illustrated above the only modification we have made to the FEEA is to return three more outputs, i.e., $q_{\max}$, $s_{\max}$,

$t_{\max}$. Thus assuming the FEEA works correctly, we require to prove that $q_{\max}$ is the quotient with maximal degree and $s_{\max}$ and $t_{\max}$ have the same index as $q_{\max}$ in the Euclidean Algorithm with inputs r_0 and r_1.

We see by induction on k that the results of the recursive call in step 2 are correct, that is, $q_{\max}$ represents the quotient with maximal degree in $\{q_1, \ldots, q_{j-1}\}$ and $s_{\max}$ and $t_{\max}$ are in the same row with $q_{\max}$. In step 4 the correct result is returned, since no other quotient has been computed. We have

$$R_j = \begin{bmatrix} s_j & t_j \\ s_{j+1} & t_{j+1} \end{bmatrix},$$

thus in step 5 if $\deg q_j > \deg q_{\max}$ then $s_{\max}$ and $t_{\max}$ are easily update by the entries of the first row of R_j. Again by induction, in step 6 $q^*_{\max}$ represents the quotient with maximal degree in $\{q_{j+1}, \ldots, q_h\}$. But $s^*_{\max}$ and $t^*_{\max}$ are not on the same row as $q^*_{\max}$ in the Euclidean algorithm for r_0 and r_1. Let l represent the index of $q^*_{\max}$ in the EEA for r_0 and r_1. In step 6, if $\deg q^*_{\max} > \deg q_{\max}$ then we require to update $s_{\max}$ and $t_{\max}$ by s_l and t_l, respectively. According to the definition of R_l we have

$$\begin{aligned} \begin{bmatrix} s_l & t_l \\ s_{l+1} & t_{l+1} \end{bmatrix} &= R_l = Q_l Q_{l-1} \ldots Q_{j+1} R_j \\ &= \begin{bmatrix} s^*_{\max} & t^*_{\max} \\ m_1 & m_2 \end{bmatrix} R_j, \end{aligned}$$

where $m_1, m_2 \in F[x]$, hence

$$\begin{bmatrix} s_l & t_l \end{bmatrix} = \begin{bmatrix} s^*_{\max} & t^*_{\max} \end{bmatrix} R_j.$$

So to update $s_{\max}$ and $t_{\max}$ we simply multiply the vector $\begin{bmatrix} s^*_{\max}, t^*_{\max} \end{bmatrix}$ by matrix R_j. Therefore, at the end of step 6, $q_{\max}$ holds the quotient with maximal degree in $\{q_1, \ldots, q_h\}$ and $s_{\max}$ and $t_{\max}$ have the same index as $q_{\max}$ in the EEA for r_0 and r_1. This implies that the final results in step 8 are correct. Note that the EEA should be modified as well to return the maximal quotient and the corresponding values of s and t in step 1. We now show how to call MFEEA to compute the desired rational function.

Fast Maximal Quotient RFR Algorithm(FMQRFR)
Input: $f, g \in \mathbb{Z}_p[x]$ with $g \neq 0$, $\deg f > \deg g \geq 0$, and $T \in \mathbb{N}$
Output: Either $n, d \in \mathbb{Z}_p[x]$ satisfying $n/d \equiv g \bmod f$, $\mathrm{lc}(d) = 1$, $\gcd(n, d) = 1$, and $\deg n + \deg d + T < \deg f$, or FAIL implying no solution exists

```
1. $r_0 \leftarrow f/\mathrm{lc}(f)$
   $r_1 \leftarrow g/\mathrm{lc}(g)$

2. $h, \rho_{h+1}, R_h, q, \tilde{s}, \tilde{t} \leftarrow$ MFEEA$(r_0, r_1, \deg r_0)$
   if $\deg q \leq T$ then return FAIL

3. $\tilde{r} \leftarrow \tilde{s} r_0 + \tilde{t} r_1$
   if $\gcd(\tilde{r}, \tilde{t}) \neq 1$ then return FAIL

4. $n \leftarrow \mathrm{lc}(g)/\mathrm{lc}(\tilde{t}) \cdot \tilde{r}$
   $d \leftarrow 1/\mathrm{lc}(\tilde{t}) \cdot \tilde{t}$
   return $(n, d)$
```

As pointed out earlier r is obtained from s and t using $r = sf + tg$, but $\tilde{s}$ and $\tilde{t}$ that are returned as the corresponding values of q, the quotient with maximal degree, are off by a constant factor. From the definitions of s and t we find that $s = \tilde{s}/\mathrm{lc}(f)$ and $t = \tilde{t}/\mathrm{lc}(g)$ and hence

$$\frac{r}{t} = \frac{\dfrac{\tilde{s}}{\mathrm{lc}(f)} f + \dfrac{\tilde{t}}{\mathrm{lc}(g)} g}{\dfrac{\tilde{t}}{\mathrm{lc}(g)}} = \frac{\mathrm{lc}(g)(\tilde{s} r_0 + \tilde{t} r_1)}{\tilde{t}} = \mathrm{lc}(g) \cdot \frac{\tilde{r}}{\tilde{t}}.$$

If we let $m = \deg f$, then step 2 takes $O(\mathsf{M}(m) \log m)$ operations in $\mathbb{Z}_p$. To compute $\tilde{r}$ in step 3, we perform two multiplications on polynomials of size at most m and one addition. The total cost for computing $\tilde{r}$ is thus $2\mathsf{M}(m) + O(2m)$ operations in $\mathbb{Z}_p$. Checking the coprimality of $\tilde{r}$ and $\tilde{t}$, using the FEEA, takes $O(\mathsf{M}(m) \log m)$ operations in $\mathbb{Z}_p$. Steps 1 and 4 both cost $O(m)$ operations in $\mathbb{Z}_p$. Thus the asymptotic cost of the algorithm is $O(\mathsf{M}(m) \log m)$.

The following algorithm is an extension of Wang's algorithm for $F[x]$ and uses the FEEA instead of the EEA.

Fast Wang's Rational Function Reconstruction Algorithm
Input: $f, g \in F[x]$ with F a field, $g \neq 0$ and $M = \deg f > \deg g \geq 0$
Output: Either $n, d \in F[x]$ satisfying $n/d \equiv g \bmod f$, $\mathrm{lc}(d) = 1$, $\gcd(n, d) = 1$ and $\deg n + \deg d < M$, or FAIL implying no such n/d exists

```
1. $N \leftarrow \lfloor M/2 \rfloor$
   $D \leftarrow M - N - 1$
   $r_0 \leftarrow f/\mathrm{lc}(f)$, $t_0 \leftarrow 0$
   $r_1 \leftarrow g/\mathrm{lc}(g)$, $t_1 \leftarrow 1$

2. $h, \rho_{h+1}, R_h \leftarrow$ FEEA$(r_0, r_1, \deg r_0 - N - 1)$

3. $n \leftarrow r_{h+1} = s_{h+1} r_0 + t_{h+1} r_1$
   $d \leftarrow t_{h+1}$
   if $\gcd(n, d) \neq 1$ then return FAIL

4. $n \leftarrow \mathrm{lc}(g)/\mathrm{lc}(d) \cdot n$
   $d \leftarrow 1/\mathrm{lc}(d) \cdot d$
   return $(n, d)$
```

If the FEEA is also used for computing $\gcd(n, d)$ in step 3, then the time complexity of Wang's algorithm would be $O(\mathsf{M}(M) \log M)$ as well. Algorithm FMQRFR normally must compute all the quotients to determine the largest but Wang's algorithm stops half way, and hence, is expected to take half the time (we will confirm this in the next table of timings). On the other hand Wang's algorithm outputs n/d if

$$\deg f \geq 2 \max(\deg n, \deg d).$$

But the Maximal Quotient algorithm only requires

$$\deg f > \deg n + \deg d + T,$$

which requires only one more point than the minimum necessary when T is chosen to be 1, i.e., we require the degree of the maximal quotient to at least 2. The following table compares the running time of both algorithms. Columns 2 and 3 illustrate the timings when the EEA is used and columns 4 and 5 show the timings when the FEEA is used. We have chosen n/d and f such that $\deg n = \deg d$ and $\deg n + \deg d + 2 = \deg f$. Note, this choice, $\deg n = \deg d$, is the worst case for the maximal quotient algorithm. The coefficients of f, n and d are chosen at random from $\mathbb{Z}_p$. The data shows that Wang's algorithm (both versions) is almost 2 times faster than the maximal quotient algorithm (both versions) as predicted. All timings are in milliseconds.

deg f	MQRFR	Wang	FMQRFR	Fast Wang
64	2.42	1.24	2.73	1.04
128	7.81	5.04	8.36	4.65
256	29.13	14.88	24.63	14.71
512	118.12	59.87	118.90	44.47
1024	479.20	236.81	430.23	182.26
2048	1825.78	950.20	1352.52	749.38
4096	7264.75	3809.14	4442.87	2374.47

5. OPEN PROBLEMS

Let p be a prime and let n/d be a rational function in $\mathbb{Z}_p(t)$. Suppose we pick m distinct points α_i from $\mathbb{Z}_p$ at random and suppose we have computed $g \in \mathbb{Z}_p[t]$ satisfying $g \equiv n/d \bmod f$ where $f = (t-\alpha_1)\times ... \times(t-\alpha_m)$. Now suppose we are attempting to reconstruct n/d but $m = \deg f \leq \deg n + \deg d$, that is, we have insufficient points α_i to reconstruct n/d.

Suppose we apply maximal quotient rational function reconstruction (algorithm MQRFR in section 2) to inputs f and g with $T = 1$, that is, we require that the degree of a maximal quotient q is at least 2 before accepting the output. Let x be the probability that algorithm MQRFR succeeds, that is, it outputs some $\bar{n}/\bar{d} \neq n/d$ with $\deg q > 1$. We make the following conjecture

Conjecture 5.1.

$$\mathrm{Prob}(x) = \mathrm{Prob}(\deg q > 1) \simeq \frac{m-1}{p}.$$

For $m = 2$ we can prove equality. For if $m = 2$, $\deg g < 2$ and $\deg q > 1$ can only happen if the linear coefficient of the input g is 0 which occurs with probability $1/p$. The difficulty in proving the conjecture for $m > 2$ is that not all monic polynomials of degree m are possible for f. Otherwise we would argue as follows.

Let P_d be the set of polynomials in $\mathbb{Z}_p[t]$ of degree d. Suppose f is selected at random from P_m and g is selected at random from P_k with $m > k \geq 0$. Let N be the number of division steps in the Euclidean algorithm with inputs f and g. In [8], Ma and von zur Gathen show that the *expected* number of division steps $E[N] = k + 1 - k/p$. This is the maximum possible number of steps $k + 1$ less k/p.

Let $y = \mathrm{Prob}(N = k + 1)$. Then

$$E[N] \leq y(k + 1) + (1 - y)k.$$

Substituting for $E[N]$ we have

$$k + 1 - k/p \leq y(k + 1) + (1 - k)k$$

from which we obtain $y \geq 1 - k/p$. Returning to our application where $\deg f = m$ and $\deg g < m$ we find that

$$\mathrm{Prob}(\deg q = 1) = \mathrm{Prob}(\deg g = m - 1 \text{ and } N = m)$$
$$\geq \frac{p-1}{p}\,(1 - \frac{m-1}{p}).$$

Thus

$$\mathrm{Prob}(x) = 1 - \mathrm{Prob}(\deg q = 1)$$
$$\leq 1 - \left[\frac{p-1}{p}(1 - \frac{m-1}{p})\right] = \frac{m}{p} - \frac{m-1}{p^2} < \frac{m}{p}.$$

Acknowledgment

We gratefully thank Petr Lisonek and the anonymous referees for their input.

6. REFERENCES

[1] Richard P. Brent, Fred G. Gustavson, and David Y. Y. Yun. Fast solution of Toeplitz systems of equations and computation of Padé approximants. *Journal of Algorithms*, **1**:259–295, 1980.

[2] G. E. Collins and M. J. Encarnacion. Efficient Rational Number Reconstruction. *J. Symbolic Computation*, **20**:287–297, 1995.

[3] Mark J. Encarnacion. Computing GCDs of Polynomials over Algebraic Number Fields. *J. Symbolic Computation*, **20**(3):299–313, 1995.

[4] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, second edition, 2003.

[5] Sara Khodadad. Fast Rational Function Reconstruction. Master's thesis, Simon Fraser University (SFU), Burnaby, BC, Canada, 2005.

[6] L. Langemyr and S. McCallum. The computation of polynomial gcd's over an algebraic number field. *J. Symbolic Computation*, **8**:429–448, 1989.

[7] Daniel Lichtblau. Half-gcd and Fast Rational Recovery. In *Proceedings of ISSAC '05*, pages 231–236. ACM Press: New York, NY, 2005.

[8] Keju Ma and Joachim von zur Gathen. Analysis of Euclidean Algorithms for Polynomials over Finite Fields. *J. Symbolic Computation*, **9**:429–455, 1990.

[9] Roman Maeder. Storage Allocation for the Karatsuba Integer Multipliation Algorithm. In *DISCO '93: Proceedings of the International Symposium on Design and Implementation of Symbolic Computation Systems*, pages 59–65. Springer-Verlag, 1993.

[10] R. T. Moenck. Fast computation of gcds. In *STOC '73: Proceedings of the fifth annual ACM Symposium on Theory of Computing*, pages 142–151. ACM Press: New York, NY, 1973.

[11] Michael Monagan. Maximal quotient rational reconstruction: An almost optimal algorithm for rational reconstruction. *Proceedings of ISSAC '04*, pages 243–249. ACM Press: New York, NY, 2004.

[12] Peter Lawrence Montgomery. *An FFT extension of the elliptic curve method of factorization*. PhD thesis, Los Angeles, CA, USA, 1992.

[13] Victor Y. Pan and Xinmao Wang. Acceleration of Euclidean Algorithm and Extensions. *Proceedings of ISSAC '02*, pages 207–213. ACM Press: New York, NY, 2002.

[14] A. Schönhage. Schnelle Berechnung von Kettenbruchentwicklungen. *Acta Informatica*, **1**:139–144, 1971.

[15] Allan Steel. Private communication.

[16] Paul S. Wang. A p-adic Algorithm for Univariate Partial Fractions. In *Proceedings of the fourth ACM Symposium on Symbolic and Algebraic Computation*, pages 212–217. ACM Press: New York, NY, 1981.