Computing Characteristic Polynomials of Matrices of Structured Polynomials

Marshall Law and Michael Monagan

Department of Mathematics
Simon Fraser University
Burnaby, British Columbia, Canada
mylaw@sfu.ca and mmonagan@cecm.sfu.ca

Abstract. We present a parallel modular algorithm for finding characteristic polynomials of matrices with integer coefficient bivariate polynomials. For each prime, evaluation and interpolation gives us the bridge between polynomial matrices and matrices over a finite field so that the Hessenberg algorithm can be used.

1 Introduction

We are interested in specific structured matrices obtained from [9] which arise from combinatorial problems. The goal is to compute their respective characteristic polynomials. Let $A(x, y)$ represent the matrix of interest with dimension $n \times n$. The entries of A are polynomials in x and y of the form

$$A_{ij}(x, y) = c_{ij} x^a y^b$$

with $a, b \in \mathbb{N} \cup \{0\}, c_{ij} \in \mathbb{Z}, 1 \leq i, j \leq n$. Please see appendix A1 for the 16 by 16 example. Let $C(\lambda, x, y) \in \mathbb{Z}[\lambda, x, y]$ be the characteristic polynomial, which is

$$C(\lambda, x, y) = \det(A - \lambda I_n)$$

by definition, where I_n is the $n \times n$ identity matrix.

The matrix sizes range from 16 to 256, so using the general purpose routine in a computer algebra system like Maple will work only for the small cases. Finding the characteristic polynomial using Maple takes over one day for the 128 by 128 case. Fortunately, there is much structure in the coefficients of the characteristic polynomial. We are able to automatically find this structure and take advantage of it. This paper presents the optimizations to computing the characteristic polynomial. On multi-core machines, we can compute the characteristic polynomial of the largest size 256 matrix in less than 24 hours.

Paper Outline

Section 2 discusses some background along with core routines in our method. Section 3 summarizes a naive first approach. Section 4 is the query phase of our algorithm, which determines the structure to be taken advantage of. Section 5 presents how the optimizations work, based on the structure discovered from section 4. Section 6 is on the parallel algorithm. Section 7 and onwards include timings, appendix and references.

2 Background

One method for computing the characteristic polynomial $C(\lambda, x, y)$ for a polynomial matrix is to construct the characteristic matrix $A - \lambda I_n$ and use the Bareiss fraction-free algorithm [2] to compute its determinant. Magma uses this method. The Bariess algorithm is a modification of Gaussian elimination based on Sylvester's identity. Since the core routine is similar to that of Gaussian elimination, it does $O(n^3)$ arithmetic operations in the ring $\mathbb{Z}[\lambda, x, y]$ which include polynomial subtraction, multiplication and exact division.

Maple uses the Berkowitz method [3] when the "CharacteristicPolynomial" routine in the "LinearAlgebra" package is called. This algorithm does $O(n^4)$ arithmetic operations in the ring $\mathbb{Z}[x, y]$ with no divisions. For our matrices, it is much faster than the $O(n^3)$ fraction-free method, over 10 times faster for the 16 by 16 case. The reason is that the intermediate polynomials in the Bareiss fraction free method grow in size and the multiplications and divisions are more expensive than the multiplications in the Berkowitz algorithm.

2.1 Hessenberg

For matrix entries over a field F, the Hessenberg algorithm [4] does $O(n^3)$ arithmetic operations in F to find $C(\lambda)$ in $F[\lambda]$. This algorithm builds up the characteristic polynomial from submatrices, and is the core of our modular algorithm. The first stage transforms the matrix A into a matrix H in Hessenberg form while preserving the characteristic polynomial. The Hessenberg form of H is given below.

$$H = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} & \dots & h_{1,n} \\ k_2 & h_{2,2} & h_{2,3} & \dots & h_{2,n} \\ 0 & k_3 & h_{3,3} & \dots & h_{3,n} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & k_n & h_{n,n} \end{bmatrix}$$

The second stage computes the characteristic polynomial of H using a recurrence. Let $C_m(\lambda)$ represent the characteristic polynomial of the top left submatrix formed by the first m rows and columns. We have the following recurrence relation starting with $C_0(\lambda) = 1$,

$$C_m(\lambda) = (\lambda - h_{m,m})C_{m-1}(\lambda) - \sum_{i=1}^{m-1} \left(h_{i,m} C_{i-1}(\lambda) \prod_{j=i+1}^{m} k_j \right).$$

2.2 The Modular Algorithm

We compute the image of the characteristic polynomial for a sequence of primes $p_1, p_2, \ldots p_m$. Then to recover the solution over the integers we simply use Chinese remainder algorithm. Below is the outline, followed by the homomorphism diagram in Figure 1 for the modular algorithm.

1. For each prime p in $p_1, p_2, \ldots p_m$, do the following:
   (a) Evaluate the matrix entries at $x = \alpha_i$ and $y = \beta_j$ modulo p.
   (b) Apply the Hessenberg algorithm to compute $C(\lambda, \alpha_i, \beta_j)$ the characteristic polynomial of the evaluated matrix $A(\alpha_i, \beta_j)$ modulo p.
   (c) Interpolate the coefficients of λ in y and x for $C(\lambda, x, y)$ modulo p.
2. Recover the integer coefficients of $C(\lambda, x, y)$ using the Chinese remainder algorithm (CRA).

$$\begin{tikzcd}
\mathbb{Z}^{n\times n}[x,y] \arrow[rr, "\text{Characteristic Polynomial}"] \arrow[d, "\text{Mod}"', "p \in \{p_1, p_2, \ldots, p_m\}"] & & \mathbb{Z}[\lambda, x, y] \\
\mathbb{Z}_p^{n\times n}[x,y] \arrow[d, "\text{Evaluate}"', "x = \alpha_i,\ y = \beta_j"] & & \mathbb{Z}_p[\lambda, x, y] \arrow[u, "\text{CRA}"'] \\
\mathbb{Z}_p^{n\times n} \arrow[rr, "\text{Hessenberg Algorithm}", "O(n^3) \text{ arithmetic operations in } \mathbb{Z}_p"'] & & \mathbb{Z}_p[\lambda] \arrow[u, "\text{Interpolate } y, x"']
\end{tikzcd}$$

Fig. 1. Modular algorithm homomorphism diagram.

2.3 Degree Bounds

To interpolate x and y in $C(\lambda, x, y)$ the modular algorithm needs $1 + \deg_x C$ points for x and $1 + \deg_y C$ for y. The degrees in x, y of the characteristic polynomial are bounded by the following:

$$\deg_x C(\lambda, x, y) \leq D_x = \min\left(\sum_{i=1}^{n} \max_{j=1}^{n} \deg_x A_{ij}, \sum_{j=1}^{n} \max_{i=1}^{n} \deg_x A_{ij}\right)$$

$$\deg_y C(\lambda, x, y) \leq D_y = \min\left(\sum_{i=1}^{n} \max_{j=1}^{n} \deg_y A_{ij}, \sum_{j=1}^{n} \max_{i=1}^{n} \deg_y A_{ij}\right)$$

Each sum within the two equations above adds the largest degree in each row and column respectively. Then we take the minimum of the two to obtain the best degree bound in that variable.

Kronecker Substitution

There is no doubt that the problem will be simpler if the matrices of interest only consist of one variable. This can be achieved by a Kronecker substitution. To ensure a reversible substitution, let $b > \deg_x C(\lambda, x, y)$, and apply it to $A(x, y = x^b)$. Then $C(\lambda, x, y)$ can be recovered from the characteristic polynomial of $A(x, y = x^b)$.

We use $b = D_x + 1$, as this is the smallest possible value for an invertible mapping. As expected, the degrees of the polynomial entries in $A(x, y = x^b)$ become quite large. Now the problem has effectively become solving for

$$\det(A(x, x^b) - \lambda I_n) = C(\lambda, x, x^b).$$

Note that in our benchmarks section, no Kronecker substitution was involved.

Coefficient Bound

The classical Hadamard inequality for n by n integer matrix $M = (m_{ij})$ asserts

$$|\det(M)| \leq H(M) = \left(\prod_{i=1}^{n} \sum_{j=1}^{n} |m_{ij}|^2 \right)^{1/2}.$$

A similar bound [11] exists for matrices with polynomial entries. Let $M(x) = (m_{ij})$ with m_{ij} a polynomial in $\mathbb{Z}[x]$. Let $s_0, s_1, \ldots$ be the coefficients of the polynomial representation of $\det(M(x))$. Let $T = (t_{ij})$ be the n by n matrix obtained from M as follows. Let t_{ij} be the sum of the absolute values of the coefficients of $m_{ij}(x)$. Then the equivalent bound is given by

$$|| \det(M)||_2 = \left(\sum |s_i|^2 \right)^{1/2} \leq H(T) = \left(\prod_{i=1}^{n} \sum_{j=1}^{n} |t_{ij}|^2 \right)^{1/2}.$$

This bound generalizes to matrices of multivariate polynomials with integer coefficients by using a Kronecker substitution. The entries of our matrix $A(x, y)$ are unit monomials. Thus for $M = A - \lambda I_n$, $t_{ii} = 2$ and $t_{ij} = 1$ for $i \neq j$. The height of $C(\lambda, x, y)$, denoted $||C(\lambda, x, y)||_\infty$, is bounded by

$$||C(\lambda, x, y)||_\infty \leq ||C(\lambda, x, y)||_2 \leq \left(\prod^{n} (n+3) \right)^{1/2} = (n+3)^{n/2}.$$

This bound tells us how many primes are needed at most to recover the integer coefficients of $C(\lambda, x, y)$. Namely, we need $\prod_{i=1}^{m} p_i > 2(n+3)^{n/2}$. Note that, with optimizations to be mentioned, the integer coefficients can be recovered with fewer primes.

Let $C(\lambda) = \sum_{i=0}^{n} c_i \lambda^i$ be the characteristic polynomial of A. Because $c_0 = C(0) = \det(A)$ we suspected that $||C(\lambda)||_\infty \leq H(T)$. Thus for our matrices, we thought we could replace the above bound $(n+3)^{n/2}$ with the Hadamard bound $n^{n/2}$ on $\det(T)$. But in [5], an example of integer matrix with entries ± 1 is given where $|c_1| > n^{n/2} > |c_0|$.

3 Naive First Approach

For starters, we will proceed with a Kronecker substitution. With that, the degrees of the matrix entries are much larger, so we decided to use fast evaluation and interpolation. The fast algorithms used are based on the FFT (Fast Fourier Transform), which have been optimized. Since the transform itself is not the main concern, we will take advantage of the transforms decimated in time and frequency, as the authors did in [10].

3.1 Structures Found

Here we identify the foundation and justification for the following sections/phases, as there is much room for optimization. Let $C(\lambda, x, y) = \sum_{i=0}^{n} c_i(x, y) \lambda^i$. Then the coefficients of λ^i can be written in the form

$$c_i(x, y) = f_i(x, y) x^{g_i} y^{h_i} (x^2 - 1)^{k_i}$$

where the f_i are bivariate polynomials with even degrees in x. See Appendix A3 for $c_0(x, y)$ and $c_1(x, y)$ for 16 by 16 matrix. The exponent values $g_i, h_i, k_i \in \mathbb{N} \cup \{0\}$, for $0 \leq i < n$, represent factors. Table 1 below contains the values for these parameters for the $n = 16$ matrix (see Appendix A1) and also other information about $f_i(x, y)$. The columns $\deg_x$ are the degrees of f_i in x and columns $\deg_y$ are the degrees of f_i in y. Notice that the largest integers in $c_i(x, y)$ in magnitude (see columns $||c_i||_\infty$) decrease as i increases but those in f_i (see columns $||f_i||_\infty$) increase and decrease.

i	g_i	h_i	k_i	$\deg_x$	$\deg_y$	$\|\|f_i\|\|_\infty$	$\|\|c_i\|\|_\infty$	i	g_i	h_i	k_i	$\deg_x$	$\deg_y$	$\|\|f_i\|\|_\infty$	$\|\|c_i\|\|_\infty$
0	32	32	32	0	0	1	601080390	8	12	10	10	20	12	4730	35264
1	32	28	28	4	4	4	160466400	9	14	8	8	16	12	3740	10876
2	24	25	25	14	6	31	28428920	10	8	6	6	20	12	2116	3242
3	26	22	22	12	8	128	16535016	11	10	4	4	16	12	806	1556
4	20	19	19	18	10	382	3868248	12	4	3	3	18	10	454	322
5	22	16	16	16	12	684	946816	13	6	2	2	12	8	142	108
6	16	14	14	20	12	1948	183648	14	0	1	1	14	6	31	22
7	18	12	12	16	12	3738	82492	15	4	0	0	4	4	4	4

Table 1. Data for the coefficients of $C(\lambda, x, y)$ for $n = 16$.

3.2 Method of Approach

As the structure suggests, the most complicated factors to recover are the bivariate polynomials $f_i(x, y)$. In the next two sections, we present two phases to recover $C(\lambda, x, y)$. The first phase is to find g_i, h_i, k_i for $0 \leq i < n$. The second phase is to compute the "cofactors" $f_i(x, y)$.

4 Phase 1 - Query

The factors that need to be found for each λ coefficient are namely

$$x^{g_i} y^{h_i} (x^2 - 1)^{k_i}.$$

We can find the lowest degrees g_i and h_i by making two queries mod a prime, one for each variable. In each query, we evaluate one variable of the matrix A to obtain an image of the characteristic polynomial in the other variable. Let p be the prime and γ be chosen at random from $\mathbb{Z}_p$. We evaluate the matrix A to obtain the two univariate matrices with integer coefficients modulo p

$$A(x, \gamma) \mod p \text{ and } A(\gamma, y) \mod p.$$

Their respective characteristic polynomials are

$$C(\lambda, x, y = \gamma) \text{ and } C(\lambda, x = \gamma, y).$$

This is a much simpler problem, as the entries have much smaller degree and no Kronecker substitution is necessary. Even for our large matrices, the characteristic polynomial of univariate matrices can solved within minutes. The query phase concludes by finding the necessary parameters for phase 2, which are the factor degrees g_i, h_i, k_i for $0 \leq i < n$, and hence the minimal number of evaluation points e_x and e_y.

Due to two random evaluations, there is a possibility of failure in this phase. But we will show that the failure probability is small.

4.1 Lowest Degree Factors

After making the two queries, we have images $C(\lambda, x, y = \gamma)$ and $C(\lambda, x = \gamma, y)$. For each λ coefficient, simply search for first and last non-zero coefficient in x or y. The lowest degrees correspond to g_i, h_i for $0 \leq i < n$. Now also let $\bar{g}_i, \bar{h}_i$ be the largest degrees with non-zero coefficient in x, y respectively. To see it in perspective, the coefficient of λ^i in $C(\lambda, x, y = \gamma)$ has the form

$$\bullet x^{\bar{g}_i} + \cdots + \bullet x^{g_i}$$

where the symbol $\bullet$ represents integers modulo p. Similarly for $C(\lambda, x = \gamma, y)$,

$$\bullet y^{\bar{h}_i} + \cdots + \bullet y^{h_i}$$

and the key values $\bar{g}_i, g_i, \bar{h}_i, h_i$ can found easily by searching for first non-zeroes.

Non-Zero Factors

Most coefficients of λ have a factor of $(x^2-1)^{k_i}$ with k_i large. Removing this reduces the number of evaluation points needed and the integer coefficient size of $c_i(x,y)/(x^2-1)^{k_i}$, hence also the number of primes needed. To determine k_i, we pick $0<\gamma<p$ at random and divide $c_i(x,\gamma)$ by (x^2-1) modulo p repeatedly. For our matrices it happens that $k_i=h_i$.

In general, to determine if $(ax+b)$ is a factor of a coefficient of $c_i(x,y)$, for small integers $a>0,b$, we could compute the roots of $c_i(x,\gamma)$, a polynomial in $\mathbb{Z}_p[x]$, using Rabin's algorithm [12]. From each root, we try to reconstruct a small fraction $-\frac{b}{a}$ using rational number reconstruction (section 5.10 of [6]). We have not implemented this.

4.2 Required Points

Now to find the minimal points required to recover all the $f_i(x,y)$, for $0\le i<n$. Since we have already computed the largest, smallest and factor degrees, we can know the maximal degrees of $f_i(x,y)$ in both variables. The minimal number of evaluation points in x,y needed to interpolate x and y are given respectively by

$$e_x := \max\frac{1}{2}\{\bar{g}_i-g_i-2k_i\}+1,\ \text{for } 0\le i<n$$

$$e_y := \max\{\bar{h}_i-h_i\}+1,\ \text{for } 0\le i<n$$

The scalar of half in e_x is due an optimization to be mentioned later (see section 5.3). The subtraction of $2k_i$ corresponds to the factor $(x^2-1)^{k_i}$.

4.3 Unlucky Evaluations

As mentioned earlier, random evaluations may cause the algorithm to return an incorrect answer, as we will explain here. Without loss of generality, consider the query of randomly evaluating at $y=\gamma$. Let $d_i=\deg_x f_i(x,y)$, then

$$f_i(x,y)=\sum_{j=0}^{d_i} f_{ij}(y)x^j,\ \text{where } f_{ij}\in\mathbb{Z}[y].$$

When $f_{i0}(\gamma)=0$, then the lowest degree is strictly greater than g_i, which is incorrect. If the algorithm continues, the target for interpolation is compromised, and the final answer is incorrect.

If $f_{id_i}(\gamma)=0$, then the largest degree becomes less than $\bar{g}_i$. This may affect e_x, the required number of evaluation points, which takes the maximum of a set (see section 4.2). The final answer will still be correct as long as e_x is correct.

Definition 1 *Let p be a prime, and $0 \leq \gamma < p$. Then γ is an unlucky evaluation if for any $0 \leq i < n$,*

$$f_{i0}(\gamma) = 0 \ (mod\ p) \ or \ f_{id_i}(\gamma) = 0 \ (mod\ p)$$

where $D_y \geq \deg_y C(\lambda, x, y)$ (from section 2.3).

Theorem 1 *The probability that γ is unlucky is at most $\frac{2nD_y}{p}$.*

Proof. Since $D_y \geq \deg_y C(\lambda, x, y)$, so $\deg_y f_{ij} \leq D_y$ for all i, j. For each $0 \leq i < n$, there are at most $2D_y$ points where

$$f_{i0}(\gamma) = 0 \pmod{p} \text{ or } f_{id_i}(\gamma) = 0 \pmod{p}.$$

There are n of these cases, giving a total of $2nD_y$ unlucky evaluations. Therefore the probability of an unlucky evaluation (for $0 \leq i < n$) is given by

$$\Pr\left[f_{i0}(\gamma) = 0 \pmod{p} \text{ or } f_{id_i}(\gamma) = 0 \pmod{p}\right] \leq \frac{2nD_y}{p}.$$

We use a 31 bit prime $p = 2^{27} \times 15 + 1$ in the query phase. For our largest matrix, the parameters are $n = 256$, $D_x = 3072$ and $D_y = 1024$. Our algorithm makes two queries, one for each variable, so the probability of an unlucky evaluation is less than 0.105%.

5 Phase 2 - Optimizations

The structure for each λ coefficient is already known, so in this section we will use a specific matrix and its characteristic polynomial. We have implemented each of the following optimizations with Newton interpolation. We note that these optimizations apply for fast interpolation (FFT) too.

We will illustrate the optimizations on the λ^6 coefficient from the 16 by 16 matrix. The coefficient of λ^6 is

$$c_6(x, y) = f_6(x, y) x^{16} y^{14} \left(x^2 - 1\right)^{14}$$

where $f_6(x, y)$ (see appendix A2) has 91 terms and is irreducible over $\mathbb{Z}$. The parameters for the 16 by 16 matrix include $g_6 = 16, h_6 = 14 = k_6$, and $\bar{g}_6 = 64, \bar{h}_6 = 26$.

If the Kronecker substitution were to be used, it will use the substitution $y = x^{97}$, implying Fourier transform size of $s = 4096 > 64 + 26(97) = 2586$. If we work on one variable at a time, it will require $(64+1)(26+1) = 1755$ points. Keep in mind that the total number of evaluation points is the same as the number of calls to the Hessenberg algorithm, which is the bottleneck of the whole algorithm. Our implementation of the FFT involves the staircase increments as well, but can be eliminated if the truncated FFT (see [8, 1]) were to be used.

Consider the step of interpolating x after y is interpolated. Let $E = \{\alpha_1, \alpha_2, \dots\}$ be the evaluation points, and $V_i = \{c_i(\alpha_1, y), c_i(\alpha_2, y), \dots\}$ be the values. By the end of this section, we will only require $(10+1)(12+1) = 143$ points, which gives a gain of more than a factor of 12. Note that the gain is greater for larger matrices.

5.1 Lowest Degree

Since the lowest degree is known, that is $g_6 = 16$, we need to interpolate $c_6(x, y)/x^{16}$. For each $\alpha_j \in E$, divide V_i by $\alpha_j^{g_6}$ point wise. Then regular interpolation will give

$$c_6(x, y)/x^{16} = f_6(x, y)y^{14}(x^2 - 1)^{14}.$$

In this example there is a saving of $g_6 = 16$ points. This optimization also applies to the other variable y, as $h_i = 14$. When both variables are taken into account, the total number of evaluation points is reduced from 1775 to $(64 - 16 + 1)(26 - 14 + 1) = 637$.

5.2 Even Degree

All the terms in $f_i(x, y)$ have even degrees in x. So if we interpolate $f_i(x^{1/2}, y)$ instead of $f_i(x, y)$, the degree of the target is halved, and the number of evaluation points is also (approximately) halved. To do so, simply square each value in E, and proceed with interpolation as usual. The polynomial recovered will have half of the true degree, then double each exponent to recover $f_i(x, y)$.

The even degrees structure for our matrices only applies to variable x. With this optimization the number of evaluation points decreases from $(64+1)$ to $(32+1)$.

5.3 Non-zero Factors

Here we have multiplicity of $k_6 = 14$. This optimization is similar to that of the lowest degree. For each $\alpha_j \in E$, we divide each V_i value by $(\alpha_j^2 - 1)^{h_i}$. Then regular interpolation will return

$$c_i(x, y)/(x^2 - 1)^{14} = f_i(x, y)x^{16}y^{14}.$$

E cannot contain ± 1, because there will be divisions by zero. This optimization is only applicable to variable x, and it alone decreases the number of evaluation points from $(64 + 1)$ to $(36 + 1)$.

Since k_i is large, $||f_i(x, y)||_\infty$ will be much smaller than $||c_i(x, y)||_\infty$. Therefore the algorithm needs fewer primes to recover $C(\lambda, x, y)$. The largest coefficient of

$(x^2-1)^{k_i}$ is $\binom{k_i}{\lceil k_i/2 \rceil}$. For the 16 by 16 case, the coefficient bound for $C(\lambda, x, y)$ is $(16+3)^8$ a 34 bit integer. The actual height $||C(\lambda, x, y)||_\infty$ is a 30 bit integer (see Table 1) and $\max ||f_i(x, y)||_\infty = 4730 = ||f_8(x, y)||_\infty$ is a 13 bit integer. For the 64 by 64 case, $||C(\lambda, x, y)||_\infty$ is 188 bits and $\max ||f_i(x, y)||_\infty$ is 72 bits.

Due to this loose bound, the problem has effectively become much smaller in terms of integer coefficient size. The target is $\max_{0 \le i < n} ||f_i(x, y)||_\infty$, instead of the much larger $\max_{0 \le i < n} ||c_i(x, y)||_\infty$. So $C(\lambda, x, y)$ can be recovered with fewer primes. We give more details in section 5.5.

5.4 Combined

If all three improvements in sections 5.1, 5.2 and 5.3 are combined together, the number of evaluation points required for x is $e_x = (64 - 16 - 2 \times 14)/2 + 1 = 11$. Likewise for y, $e_y = (26 - 14) + 1 = 13$. So the total the number of evaluations and hence Hessenberg calls decreases from 1755 to $11 \times 13 = 143$. Since the degrees in y are dense with the lowest degree optimization, we evaluate x first then y and interpolate in reverse order.

5.5 Chinese Remainder Algorithm

The last step of the modular algorithm is to recover the solution over the integers. For m primes and each non-zero coefficient $0 \le c_i < p_i$, we have to solve the system of congruences

$$u \equiv c_i \pmod{p_i} \quad \text{for } 1 \le i \le m.$$

To build up the final coefficients u over $\mathbb{Z}$, we use the mixed radix representation for the integer u, namely, we compute integers $v_1, v_2, \dots, v_m$ such that

$$u = v_1 + v_2 p_1 + v_3 p_1 p_2 + \cdots + v_m p_1 p_2 \dots p_{m-1}.$$

See page 206 of [7]. To account for negative coefficients in $C(\lambda)$ we solve for v_i in the symmetric range $-\frac{p_i}{2} < v_i < \frac{p_i}{2}$ to obtain u in the range $-\frac{M}{2} < u < \frac{M}{2}$ where $M = \prod_{i=1}^{m} p_i$. From the non-zero factors optimization, the coefficient size bound on $||f_i(x, y)||_\infty$ becomes a very loose one. As stated previously for the 64 by 64 matrix, the bound is is 188 bits, but we can recover $C(\lambda, x, y)$ with only 72 bits. So after computing each image of $C(\lambda, x, y) \bmod p_1, p_2, \dots$, we build the solution in mixed radix form until it stabilizes, that is when $v_k = 0$.

6 Parallelization

The modular algorithm was originally chosen since the computation for each prime can be done in parallel. Each evaluation, Hessenberg call and interpolation may also be computed in parallel. We chose to run each prime sequentially and look to parallelize within each prime for two reasons. First, we don't know how many primes are necessary because of the loose bound from section 5.3. Second, memory may become an issue for computers with low RAM.

Our implementation in Cilk C does the x evaluations in parallel, and it suits the incremental method more. With each prime, the algorithm starts by evaluating x, and this evaluation is trivial since the matrix has unit monomials. In the previous section we have seen that $e_x = 11$ is required for the smallest case (16 by 16). But this matrix is too small to see any significant speed up. The 64 by 64 matrix on a computer with 4 cores shows a good speed up close to the theoretical maximum. Please see the benchmarks section for more details on timings.

7 Benchmarks

Table 2 consists of timings of our modular algorithm. Column min is is the minimum number of 30 bit primes needed to recover the integer coefficients in $C(\lambda, x, y)$. Column bnd is the number of primes needed using the bound for $||C(\lambda, x, y)||_\infty$. The number of calls to the Hessenberg algorithm is $(xy)(\min + k)$ assuming we require k check primes. Our implementation uses $k = 1$ check prime. Table 3 includes data for Maple 2016 and Magma V2.22-2. The names and details of machines we ran our software on are given below and they all run Fedora 22.

- sarah: Intel Core i5-4590 quad core at 3.3 GHz, 8 GB RAM
- mark: Intel Core i5-4670 quad core at 3.4 GHz, 16 GB RAM
- luke: AMD FX8350 eight core at 4.2 GHz, 32 GB RAM
- ant: Intel Core i7-3930K six core at 3.2 GHz, 64 GB RAM

Size	#points	#primes	sarah		mark		luke		ant	
n	x, y	min,bnd	1 core	4 cores	1 core	4 cores	1 core	8 cores	1 core	6 cores
16	11,13	1, 2	0.04s	0.01s	0.04s	0.01s	0.04s	0.01s	0.04s	0.01s
32	28,31	1, 3	0.48s	0.10s	0.58s	0.10s	0.60s	0.10s	0.64s	0.14s
64	67,61	3, 7	19.12s	5.19s	18.64s	5.08s	32.93s	4.84s	21.52s	4.16s
128	131,141	6, 16	14.72m	3.92m	14.30m	4.41m	30.36m	4.41m	16.85m	2.87m
256	261,281	12, 35	14.42h	3.75h	14.23h	3.77h	31.29h	4.28h	16.53h	2.74h

Table 2. Modular algorithm timings in seconds (s), minutes(m) or hours (h)

	Maple								Magma
Size	sarah		mark		luke		ant		ant
n	real	cpu	real	cpu	real	cpu	real	cpu	cpu
16	0.28s	0.30s	0.21s	0.23s	0.46s	0.53s	0.32s	0.36s	0.32s
32	34.1s	45.2s	30.2s	41.1s	50.3s	83.7s	32.7s	46.3s	99.7s
64	19.9h	32.6h	12.1h	23.2h	3.63h	5.42h	2.86h	3.91h	15.1h
128,256	Not attempted								

Table 3. Maple and Magma timings in seconds (s), minutes(m) or hours (h)

8 Appendix

A1: 16 x 16 matrix

$$\begin{bmatrix}
x^8 & x^5y & x^5y & x^4y^2 & x^5y & x^2y^2 & x^4y^2 & x^3y^3 & x^5y & x^4y^2 & x^2y^2 & x^3y^3 & x^4y^2 & x^3y^3 & x^3y^3 & x^4y^4 \\
x^7 & x^6y & x^4y & x^5y^2 & x^4y & x^3y^2 & x^3y^2 & x^4y^3 & x^4y & x^5y^2 & xy^2 & x^4y^3 & x^3y^2 & x^4y^3 & x^2y^3 & x^5y^4 \\
x^7 & x^4y & x^6y & x^5y^2 & x^4y & xy^2 & x^5y^2 & x^4y^3 & x^4y & x^3y^2 & x^3y^2 & x^4y^3 & x^3y^2 & x^2y^3 & x^4y^3 & x^5y^4 \\
x^6 & x^5y & x^5y & x^6y^2 & x^3y & x^2y^2 & x^4y^2 & x^5y^3 & x^3y & x^4y^2 & x^2y^2 & x^5y^3 & x^2y^2 & x^3y^3 & x^3y^3 & x^6y^4 \\
x^7 & x^4y & x^4y & x^3y^2 & x^6y & x^3y^2 & x^5y^2 & x^4y^3 & x^4y & x^3y^2 & xy^2 & x^2y^3 & x^5y^2 & x^4y^3 & x^4y^3 & x^5y^4 \\
x^6 & x^5y & x^3y & x^4y^2 & x^5y & x^4y^2 & x^4y^2 & x^5y^3 & x^3y & x^4y^2 & y^2 & x^3y^3 & x^4y^2 & x^5y^3 & x^3y^3 & x^6y^4 \\
x^6 & x^3y & x^5y & x^4y^2 & x^5y & x^2y^2 & x^6y^2 & x^5y^3 & x^3y & x^2y^2 & x^2y^2 & x^3y^3 & x^4y^2 & x^3y^3 & x^5y^3 & x^6y^4 \\
x^5 & x^4y & x^4y & x^5y^2 & x^4y & x^3y^2 & x^5y^2 & x^6y^3 & x^2y & x^3y^2 & xy^2 & x^4y^3 & x^3y^2 & x^4y^3 & x^4y^3 & x^7y^4 \\
x^7 & x^4y & x^4y & x^3y^2 & x^4y & xy^2 & x^3y^2 & x^2y^3 & x^6y & x^5y^2 & x^3y^2 & x^4y^3 & x^5y^2 & x^4y^3 & x^4y^3 & x^5y^4 \\
x^6 & x^5y & x^3y & x^4y^2 & x^3y & x^2y^2 & x^2y^2 & x^3y^3 & x^5y & x^6y^2 & x^2y^2 & x^5y^3 & x^4y^2 & x^5y^3 & x^3y^3 & x^6y^4 \\
x^6 & x^3y & x^5y & x^4y^2 & x^3y & y^2 & x^4y^2 & x^3y^3 & x^5y & x^4y^2 & x^4y^2 & x^5y^3 & x^4y^2 & x^3y^3 & x^5y^3 & x^6y^4 \\
x^5 & x^4y & x^4y & x^5y^2 & x^2y & xy^2 & x^3y^2 & x^4y^3 & x^4y & x^5y^2 & x^3y^2 & x^6y^3 & x^3y^2 & x^4y^3 & x^4y^3 & x^7y^4 \\
x^6 & x^3y & x^3y & x^2y^2 & x^5y & x^2y^2 & x^4y^2 & x^3y^3 & x^5y & x^4y^2 & x^2y^2 & x^3y^3 & x^6y^2 & x^5y^3 & x^5y^3 & x^6y^4 \\
x^5 & x^4y & x^2y & x^3y^2 & x^4y & x^3y^2 & x^3y^2 & x^4y^3 & x^4y & x^5y^2 & xy^2 & x^4y^3 & x^5y^2 & x^6y^3 & x^4y^3 & x^7y^4 \\
x^5 & x^2y & x^4y & x^3y^2 & x^4y & xy^2 & x^5y^2 & x^4y^3 & x^4y & x^3y^2 & x^3y^2 & x^4y^3 & x^5y^2 & x^4y^3 & x^6y^3 & x^7y^4 \\
x^4 & x^3y & x^3y & x^4y^2 & x^3y & x^2y^2 & x^4y^2 & x^5y^3 & x^3y & x^4y^2 & x^2y^2 & x^5y^3 & x^4y^2 & x^5y^3 & x^5y^3 & x^8y^4
\end{bmatrix}$$

A2: $f_6(x, y)$ of 16 x 16 matrix

$$\begin{gathered}
(2x^{16} + 4x^{14})y^{12} + (4x^{18} + 32x^{16} + 28x^{14} + 8x^{12})y^{11} + \\
(x^{20} + 34x^{18} + 149x^{16} + 188x^{14} + 43x^{12} - 22x^{10} + 3x^8)y^{10} + \\
(16x^{20} + 128x^{18} + 452x^{16} + 568x^{14} + 268x^{12} - 32x^{10} - 72x^8 - 8x^6)y^9 + \\
(52x^{20} + 348x^{18} + 910x^{16} + 1172x^{14} + 704x^{12} + 68x^{10} - 136x^8 - 120x^6 - 28x^4)y^8 + \\
(112x^{20} + 596x^{18} + 1344x^{16} + 1788x^{14} + 1224x^{12} + 216x^{10} - 220x^8 - 184x^6 - 92x^4 - 32x^2)y^7 + \\
(133x^{20} + 734x^{18} + 1551x^{16} + 1948x^{14} + 1476x^{12} + 428x^{10} - 320x^8 - 276x^6 - 81x^4 - 34x^2 - 15)y^6 + \\
(112x^{20} + 596x^{18} + 1344x^{16} + 1788x^{14} + 1224x^{12} + 216x^{10} - 220x^8 - 184x^6 - 92x^4 - 32x^2)y^5 + \\
(52x^{20} + 348x^{18} + 910x^{16} + 1172x^{14} + 704x^{12} + 68x^{10} - 136x^8 - 120x^6 - 28x^4)y^4 + \\
(16x^{20} + 128x^{18} + 452x^{16} + 568x^{14} + 268x^{12} - 32x^{10} - 72x^8 - 8x^6)y^3 + \\
(x^{20} + 34x^{18} + 149x^{16} + 188x^{14} + 43x^{12} - 22x^{10} + 3x^8)y^2 + \\
(4x^{18} + 32x^{16} + 28x^{14} + 8x^{12})y + (2x^{16} + 4x^{14})y^0
\end{gathered}$$

A3: First two coefficients of $C(\lambda, x, y)$ for 16 by 16 matrix

$$c_0(x,y) = x^{32}y^{32}\left(x^2-1\right)^{32}$$
$$c_1(x,y) = -x^{32}y^{28}\left(x^2-1\right)^{28}\left(2\,x^4y^2+4\,x^2y^3+4\,x^2y^2+y^4+4\,x^2y+1\right)$$

B1: Time 16 by 16 on Maple

```
A := Matrix(16, 16, [ [x^8, x^5*y, ...], ... ]);
with(LinearAlgebra):
C := CodeTools[Usage]( CharacteristicPolynomial(A, lambda) ):
```

B2: Time 16 by 16 on Magma

```
P<x,y> := PolynomialRing( IntegerRing(), 2);
A := Matrix(P, 16, 16, [ [x^8, x^5*y, ...], ... ]);
time C := CharacteristicPolynomial(A);
```

References

1. Arnold, A.: A New Truncated Fourier Transform Algorithm. In: Proceedings of ISSAC 13, pp. 15–22. ACM Press (2013)
2. Bareiss, E.H.: Sylvester's Identity and Multistep Integer-Preserving Gaussian Elimination. Mathematics of Computation **22**(103), pp. 565–578. (1968)
3. Berkowitz, S.J.: On Computing The Determinant in Small Parallel time using a Small Number of Processors. Inf. Processing Letters **18**(3), pp. 147–150. (1984)
4. Cohen, H.: A Course in Computational Algebraic Number Theory. pp. 138. Springer-Verlag (1995)
5. Dumas, J.G.: Bounds on the coefficients of the characteristic and minimal polynomials. J. of Inequalities in Pure and Applied Mathematics **8**(2), art. 31. pp. 1–6. (2007)
6. von zur Gathen, J., Gerhard, J.: Modern Computer Algebra. Cambridge University Press (2003)
7. Geddes, K.O., Czapor, S.R., Labahn, G.: Algorithms for Computer Algebra. Kluwer Academic Publishers, Boston, Massachusetts (1992)
8. van der Hoeven, J.: The truncated fourier transform and applications. In: Proceedings of ISSAC 2004, pp. 290–296. ACM Press (2004)
9. Kauers, M.: Personal Communication.
10. Law, M., Monagan, M: A parallel implementation for polynomial multiplication modulo a prime. In: Proceedings of PASCO 2015, pp. 78–86. ACM Press (2015)
11. Lossers, O.P.: A Hadamard-Type Bound on the Coefficients of a Determinant of Polynomials. SIAM Rev. **16**(3), pp. 394–395, solution to an exercise by A. Jay Goldstein and Ronald L. Graham. (2006)
12. Rabin, M.: Probabilistic Algorithms in Finite Fields. SIAM J. Comput. **9**(2), pp. 273–280. (1979)