An Improved Algorithm for Multivariate Polynomial Interpolation

Min Tang and Guoqiang Deng

ABSTRACT

In this paper, we present an improved method based on Zippel’s algorithm for multivariate polynomial interpolation, and a testing point technique is proposed to verify the recovered polynomial. Compared with Zippel’s algorithm, the new method not only returns an exact target polynomial but takes less computation time as well. The experiments show the effectiveness of the improved method.

INTRODUCTION

Polynomial interpolation is an old topic, but it is an indispensable part of numerical computation because it provides an effective way to present data which comes from the function of a certain scientific problem. In practice work, cubic splines and Bezier splines are developed for the shipbuilding and aircraft industries. The most well-known application of splines is computing typesetting. Bezier splines were a simple and valid way to adapt the same mathematical curves to fonts. Steve Jobs paid much attention to this technique because he hopes to find a way to control laser printer. A more flexible format called PDF use Bézier splines to represent printed characters in arbitrary fonts[1]. In short, interpolation is the reverse process of evaluation: Given some points, compute a function that can generate them.

Although it is unrealistic to expect all functions are polynomials, they may be close enough to replace other complicated functions and solve real life problems. More important, for digital computers polynomial is the key and fundamental function. Central processing units can implement polynomials evaluation very fast. Another important aspect, in the real world, physical quantities usually depend on two or more variables. So we focus on the multivariate polynomial interpolation in this paper. Dense interpolations have exponential behavior since they need as many as

1Min Tang
School of Mathematics and Computing Science, Guangxi Key Laboratory of Cryptography and Information Security, Guilin University of Electronic Technology, Guilin, Guangxi, China, 541004. Shanghai Key Lab of Trustworthy Computing, East China Normal University, Shanghai, China, 200062.

Guoqiang Deng
School of Mathematics and Computing Science, Guangxi Colleges and Universities Key Laboratory of Data Analysis and Computation, Guilin

The work was supported by Guangxi Key Laboratory of Cryptography and Information Security (No. GCIS201615), the National Natural Science Foundation of China (No.11561015), the Natural Science Foundation of Guangxi Province (No. 2016GXNSFFA380009).
(d+1)^n independent evaluations, where d is degree of each variable and n is the number of variables. Some early approaches, such as modular, have an exponential behavior. But the current algorithms can avoid exponential worst case and take polynomial time (including Zippel’s algorithm and Tiwari’s algorithm).

If reconstructed polynomial is univariate case, then Lagrange and Newton have already given us excellent solving methods. These methods are also needed and used to our multivariate polynomial interpolation algorithm. Consider multivariate cases, we have to refer to two important algorithms. One is a probabilistic algorithm presented by Richard Zippel[3]. Another is a deterministic algorithm introduced by Michael Ben-or and Prasoon Tiwari[2]. These two algorithms are both for sparse polynomials, which is similar to our study object. Zippel’s interpolation proceeds one variable at a time and interpolates densely. But this is clearly not exponential in the number of variables. The dominant behavior is $O(t^3)$, assuming $t >> d$ or $n$. In contrast, Tiwari’s algorithm is a deterministic algorithm. If the exact number of terms in target polynomial was known then we only need $2t$ evaluations. Zippel’s algorithm requires knowledge of an upper bound of the degree of each variable and Tiwari’s algorithm requires an upper bound of the number of terms.

To make these algorithms more efficient early termination techniques were applied to Zippel’s and Tiwari’s algorithms by Erich Kaltofen [8-9]. Other some references to multivariate polynomial interpolation usually include [4-7].

Except for multivariate polynomial cases rational functions have been considered for recent years. For details, see [10-13].

Our method is a variation of Zippel’s algorithm. If we apply Zippel’s algorithm to interpolation the degree bound of each variable appearing in $P$ must be given. However, it is not necessary to know the indispensable information (degree bound) in our method, which is in contrast to Zippel’s. The main theorem of polynomial interpolation is the key to understanding our ideas. Our method is a probabilistic algorithm which is similar to Zippel’s. In most cases, especially given degree bound is too high so that it will take plenty of time by Zippel’s, whereas our method can use very few probes to complete interpolation. Numerical experiments show that the new algorithm is efficient.

The rest of the paper is organized as follows. In Section 2, we formulate the multivariate polynomial interpolation problems. In Section 3, we introduce Zippel’s algorithm and analyze it. An improved algorithm is proposed in Section 4. We illustrate the effectiveness of our new technique through some examples in Section 5. Some concluding remarks are made in Section 6.

PROBLEM FORMULATION

Suppose that we have known some points (vectors) $x_i=(x_1, x_2, \ldots, x_m)$ and $y_i=f(x_i)$, that is, discrete information is given. Can we recover the function $y=f(x)$ which pass through given vectors? That is interpolation problem.

Assume there exists a black-box which contains a multivariate polynomial $P(x)$ in the variables $x_1, x_2, \ldots, x_n$ with real coefficients. If we input a specific value of $x=(a_1, a_2, \ldots, a_n)$, then the box will tell us the output $P(a_1, a_2, \ldots, a_n)$. Furthermore, there is no limit on the probes, i.e., it can evaluate $P(x)$ for any independent variable $x$ (when it is defined).
We need to determine all the coefficients of $P(x)$. To verify that recovered $P(x)$ is an exact polynomial in the box, one can take some points (excluding probes we have used), substitute these values into the $P(x)$ and the black-box respectively, then compare their outputs. If the results via $P(x)$ agrees with the box, then $P(x)$ is the goal polynomial.

In this paper, we propose an improved method to multivariate polynomial interpolation. The effectiveness of the new method is demonstrated by the experiments in Section 5. For the first group experiments the number of variables in goal polynomial is changed while other arguments (the number of terms, the total degree) are fixed. For the second and the third one, the number of terms and the total degree of polynomials are changed respectively (fixing other two parameters).

**ZIPPEL’S ALGORITHM**

Since our algorithm is a variation of Zippel’s algorithm. We believe that it is necessary to sum up this method, including idea and time consumption.

**Overview of Zippel’s Algorithm**

In this section, let us recall the key idea and implementation process of Zippel’s probabilistic algorithm.

Suppose that $P(x_1, x_2, ..., x_n)$ is the goal polynomial. This algorithm produce a sequence of polynomials,

$$
P_1 = P(x_1, x_{20}, ..., x_{n0}),
$$

$$
P_2 = P(x_1, x_{21}, ..., x_{n0}),
$$

$$\vdots$$

$$
P_n = P(x_1, x_{2n}, ..., x_{n0}),
$$

where $P_1$ is a univariate polynomial in $x_1$ and other variables $x_{2}, ..., x_{d}$ are substituted by values $x_{20}, ..., x_{n0}$. $P_2$ is two variables polynomial in $x_1, x_2$, and so on. $P_n$ is multivariate polynomial in $x_1, x_2, ..., x_n$, which is goal polynomial $P$. Point $(x_{10}, x_{20}, ..., x_{n0})$ is chosen randomly in space $R^n$. If one want to implement this algorithm, there is an essential information which is given in advance. That is the degree bound of each variable appearing in $P$ (suppose that the largest one is no more than $d$).

To obtain the first polynomial $P_1$ one can pick $x_{10}, x_{11}, ..., x_{1d}$ and examine the values of $P$ at the points $(x_{10}, x_{20}, ..., x_{n0}), (x_{1d}, x_{20}, ..., x_{n0}).$ For univariate polynomial interpolation the above information is sufficient to determine $P_1$, which can be written in the form

$$f_1(1, ..., l)x_1^{d-1} + \cdots + f_{d-l}(1, ..., l)x_1^{d-2} + f_{d-l}(1, ..., l).$$

According to the number of terms in $P_1$ one can set up a equation system in $t$ unknown coefficients. The values we need are $P(x_{10}, x_{21}, ..., x_{n0}), ..., P(x_{1l}, x_{21}, ..., x_{n0}).$ So that $P(x, x_{21}, ..., x_{n0})$ is obtained via these values. Because we have known $P(x, x_{20}, ..., x_{n0})$ and $P(x, x_{21}, ..., x_{n0})$, by setting and solving the resting $d-1$ linear
equation systems we will get enough knowledge to obtain $P_2$. This procedure may
be repeated until we have determined the final polynomial $P_n$.

Algorithm Analysis

We can compute the number of probes, i.e., the number of polynomial evaluations. Completing $P(x_1)$ interpolation needs $d+1$ probes. Suppose that the number of terms in $P$ is no more than $t$, then the size of coefficient matrix in linear equation system is no more than $t \times t$ for variables $x_2, \ldots, x_n$. And recover each polynomial $P_i (i=2,3, \ldots, n)$ need $d$ equations. In general, we need $dt$ probes at most when considering variables $x_2, \ldots, x_n$. The total probes is $(d+1)+dt(n-1)$ at most and no more than $dtm$.

Except evaluating the polynomials we need solve a set of linear systems of equations. The number of systems we construct is $dn$ at most.

A NEW ALGORITHM FOR INTERPOLATION

The Zippel’s interpolation method, as described in the previous section, is a probabilistic way to construct the polynomial in the black-box. Our algorithm does not require a priori knowledge of the degree $d$. The method results in more manageable and less computationally complexity.

Algorithm Description

Unlike Zippel’s algorithm, the degree bound $d$ is not required. In this case how do we know the number of probes is sufficient to reconstruct polynomial $p_i$? First let’s recall a theorem which is the main theorem of polynomial interpolation.

Let $(x_1, y_1), \ldots, (x_m, y_m)$ be $m$ points in the plane with distinct $x_i$. Then there exists one and only one polynomial $P$ of degree $m-1$ or less that satisfies $P(x_i)=y_i$ for $i=1, \ldots, m$.

From the theorem above suppose that the degree of variable $x_i$ is $d_i$, then we need $d_i+1$ probes exactly. If the number of probes is less than $d_i+1$, then we cannot recover $P_i$. Otherwise, the redundant points are useless to interpolate $P_i$.

Our algorithm begins by choosing a testing point $(x_{10}, x_{20}, \ldots, x_{n0})$ for the verification and a starting point $(x_{11}, x_{21}, \ldots, x_{n1})$ for the interpolation.

Firstly we want to recover the univariate polynomial $P_1$. Put the following three operations into a loop and execute in order until the last operation satisfies the exit condition.

1. (Adding probes)
   Add a new point $(x_{1i}, x_{2i}, \ldots, x_{ni})$ (the initial value of $i$ is 2) and evaluate $P$ at this point.
2. (Interpolating)
   Complete the $P_1$ interpolation according to the information of previous obtained and the new probe.
3. (Testing)
   Substitute $(x_{10}, x_{20}, \ldots, x_{n0})$ into the current polynomial $P_1$ and let $(x_{10}, x_{20}, \ldots, x_{n0})$ as an input of $P$. Compare the evaluation $P_1(x_{10}, x_{20}, \ldots, x_{n0})$ and output $P(x_{10}, x_{20}, \ldots, x_{n0})$. If the results via $P_1$ agrees with the $P$, then $P_1$ is the exact polynomial. Termi-
nate the loop. If the results are different from each other, it means the probes is less than or equal to degree $d_j$. Return the first step to add a new point.

The same operations can be applied to variables $x_j (j=2, \ldots, n)$. By far we have obtained the exact degree of each variable. Suppose the degree of variables $x_j$ is $d_j$, for $j=1, 2, \ldots, n$.

From interpolating $P_1$, suppose we obtain a polynomial

$$P_1 = P(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{t_1} f_i(x_2, \ldots, x_n)x_1^{d_i}$$

where the number of terms of $P_1$ is $t_1$.

The form of undetermined polynomial is

$$P(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{t_1} f_i(x_2, \ldots, x_n)x_1^{d_i}.$$

In the following of this section, we describe in detail how to set up a system of linear equations, and then polynomial $P_2(x_1, x_2, \ldots, x_n)$ is determined by solving this system.

Denote the $i$-th prime integer by $p_i$. Evaluate the polynomial $P$ at the points given by

$$u_j = (p_i^{r_j}, x_2, x_3, \ldots, x_n), \text{ for } j = 0, 1, \ldots, t_1 - 1.$$

Let $v_j = P(u_j)$ and set up a sequence of equations

$$\sum_{i=1}^{t_1} f_i(x_2, \ldots, x_n)(p_i^{r_j})^{d_i} = v_j, \text{ for } j = 0, 1, \ldots, t_1 - 1.$$

Let $M$ be the $t_1 \times t_1$ matrix defined by $M_{ij} = (p_i^{r_j})^{d_i}$. We will determine the coefficients $f_i(x_2, \ldots, x_n)$, for $i = 1, \ldots, t_1$ by solving a linear system, since $M$ is a nonsingular Vandermonde matrix.

To obtain polynomial $P_2 = P(x_1, x_2, \ldots, x_n)$ some extra polynomials $P_2(x_1, x_2, \ldots, x_n)$, $P_2(x_1, x_2, \ldots, x_n)$, $P_2(x_1, x_2, \ldots, x_n)$ are needed.

Up to now we have had enough information to interpolate $P_2 = P(x_1, x_2, \ldots, x_n)$.

Repeating this procedure we will finally determine $P$.

**Complexity of the New Method**

The main factors which affect the efficiency our algorithm include:

1. Polynomial evaluation
2. Univariate polynomial interpolation algorithm
3. Linear systems solving

Let’s describe the methods which are used in the above three operations.

For the first operation at a cost of evaluating a polynomial depends on the black-box given. Thus the less probes are needed, the less evaluation time the algorithm takes.

For the second operation we use the Lagrange interpolation formula.

For the third operation we use elimination algorithm to solve all linear systems.

It is obvious that Zippel’s algorithm need more evaluations than ours. This fact directly leads to the quantity of linear systems should be more. Besides, the size of
these systems should be larger. For a square coefficient matrix, the elimination algorithm (row reduction) is used and measured in flops (or floating point numbers). For \( n \times (n+1) \) matrix, the reduction to echelon form can take approximately \( 2n^3/3 \) flops.

In our algorithm it is not neglect that comparing two polynomials at a fixed point (amounts to compare two real numbers) needs extra time. In floating-point arithmetic we have a fast way of comparing two signed real numbers\[14\].

Completing \( P(x_1) \) interpolation needs \( d_1+1 \) probes. Suppose again that the number of terms in \( P \) is no more than \( t \), then the size of coefficient matrix in linear equation system is no more than \( t \times t \) for variables \( x_2, \ldots, x_n \). And recover each polynomial \( P(i=2,3, \ldots,n) \) need \( d_i+1 \) equations. In general, we need \( (d_i+1)t \) probes at most when considering variables \( x_2, \ldots, x_n \). The total probes is

\[
d_1 + 1 + \sum_{i=2}^{n} (d_i + 1)t
\]

at most and no more than \( d_{\text{max}}tn \), where

\[
d_{\text{max}} = \max\{d_1, d_2, \ldots, d_n\}.
\]

Only in the worst case \( d_i = d_i = \ldots = d_n \) our algorithm is the same as Zippel’s (assume the degree bound \( d \) is exactly equal to \( d_i \)).

Example

To indicate how to implement our algorithm we work out all the details in the following simple example.

Assume the goal multivariate polynomial (that is, the function in the black-box) is

\[
P = 5x_1^2x_2 + x_1x_3 + 3.
\]

Firstly, we choose a testing point and a starting point. For instance, take point \((0,0,0)\) as a testing point and \((1,1,1)\) as a starting point. These two points have image 3 and 9 respectively.

Step 1: Generate univariate polynomial \( P_1(x_1,1,1) \).

Loop_1: compute the value of \( P \) at point \((2,1,1)\). For simplicity, we use the notation \( P(\cdot) \) to indicate the evaluation of \( P \) at some point. For instance, \( P(2,1,1)=25 \). Using the information \( P(1,1,1) \) and \( P(2,1,1) \) we can interpolation a polynomial which is \( 5x_1^2+7 \). Substitute \((0,0,0)\) into \( 5x_1^2+7 \) and get the result. It indicates we should enter the next loop to add new point because the result (-7) is not equal to 3.

Loop_2: compute \( P(3,1,1) \) and use the above information. We obtain the new polynomial \( 5x_1^2 + x_1 + 3 \). Next it’s time to verify this polynomial whether is the exact one. By substituting \((0,0,0)\) into \( 5x_1^2 + x_1 + 3 \) we get the value 3. Due to \( P(0,0,0)=3 \) the iteration can be terminated. This means we have already got the exact polynomial \( P_1(x_1,1,1) \).

From the above description one can understand why our algorithm doesn’t require the knowledge of degree bound \( d \). We should give the credit to the Theorem 1.

Step 2: Generate two variables polynomial \( P_2(x_1, x_2, 1) \).

According to the form of \( P_1 \) (i.e., \( 5x_1^2 + x_1 + 3 \)) we can construct an undetermined polynomial \( f_2x_1^2 + f_1x_1 + f_0 \). Three function values of \( P \) at points \((1,2,1)\), \((2,2,1)\), \((3,2,1)\) are required to set up a linear equation system,
Solving the system can get $P_1(x_1,2,1)$ which is $10x_1^2 + x_1 + 3$. In conjunction with $P_1(x_1,1,1)$ (i.e., $5x_1^2 + x_1 + 3$) interpolate each monomial in $x_2$. The result of this operation is $5x_1^2 + x_1 + 3$.

Next step we confirm the polynomial $5x_1^2x_2 + x_1 + 3$ is the polynomial $P_2(x_1, x_2,1)$ exactly. Replace three zeros with all variables of $5x_1^2 + x_1 + 3$. The computation result is 3, which indicates our statement is true.

Step 3: Generate polynomial $P_3(x_1, x_2, x_3)$.
The process of this step is analogue of step 2. According to the form of $P_2$ we can construct a polynomial $g_2x_1^2x_2 + g_1x_1 + g_0$. Three function values of $P$ at points $(1,1,2), (2,2,2), (3,3,2)$ are required to set up a linear equation system,

\[
\begin{align*}
g_2 + g_1 + g_0 &= 10 \\
g_2 + 2g_1 + g_0 &= 47 \\
27g_2 + 3g_1 + g_0 &= 144
\end{align*}
\]

Solving the system can get $P_2(x_1,x_2,2)$ which is $5x_1^2x_2 + 2x_1 + 3$. In conjunction with $P_2(x_1,x_2,1)$ (i.e., $5x_1^2 + x_1 + 3$) interpolate each monomial in $x_3$. The result of this operation is $5x_1^2x_2 + x_1x_3 + 3$. One can verify the result is exact.

Up to now we have got the goal polynomial $P$. Obviously this example is very simple, only containing three variables and tiny integer coefficients. Moreover, the small number of terms leads to the small size of linear equation system.

Therefore, the next section we are planning to explore different and various size applications.

**TIMINGS**

In this section, we apply the new method presented in Section 4 to multivariate polynomials interpolation. We focus attention to the timings of the algorithm. In the following tables $n$ denotes the number of variables, $t$ denotes the number of terms in $P$, $d$ denotes the number of the total degree of $P$. After a set of multivariate polynomials are interpolated, Table 1-3 report the timing taken by our algorithm (Time_1) and Zippel’s algorithm (Time_2). Note that we give a rigorous degree bound to Zippel’s algorithm.

Remark that all routines are run in Maple 15 on Intel(R) Core(TM) at 3.10GHZ CPU and 4GB memory under Windows.

**Experiment 1**

In the first group implementations, we mainly discuss how the change of the number of variables affects the efficiency of our algorithm.

In this case the number of terms $t$ and the total degree $d$ of $P$ are fixed (while $n$ is increased). We choose six sets of data to illustrate our method. Table 1 gives the numerical testing results.
Table 1. Comparing the running time in the case of changing $n$.

| Ex. | $n$ | $t$ | $d$ | Coeff. Range | Time$_1$(s) | Time$_2$(s) |
|-----|-----|-----|-----|--------------|-------------|-------------|
| 1   | 5   | 5   | 5   | [-100,100]   | 0.047       | 0.141       |
| 2   | 10  | 5   | 5   | [-100,100]   | 0.125       | 0.249       |
| 3   | 15  | 5   | 5   | [-100,100]   | 0.156       | 0.358       |
| 4   | 5   | 10  | 10  | [-100,100]   | 0.187       | 0.234       |
| 5   | 10  | 10  | 10  | [-100,100]   | 0.405       | 0.515       |
| 6   | 15  | 10  | 10  | [-100,100]   | 0.530       | 0.796       |

From Table 1, we observe that both algorithms cost more time with the increase of $n$. Because the main strategy of the algorithm is variable by variable, the more quantity of $n$, the more time we will spend.

Figure 1 shows the result of comparing the running time of Zippel’s algorithm and ours in the case of changing the number of variables $n$. Other arguments are chosen as follows: take $t=10$; $d=10$; the range of coefficients $D=[-100,100]$. Symbol diamond denotes the running time by Zippel’s algorithm and symbol cross denotes the time by ours.

![Figure 1. Comparing the running time in the case of $t=10$, $d=10$.](image)

Experiment 2

In these experiments we let $n$ and $d$ fix and let $t$ increase gradually. Table 2 illustrates the experiment results.

Table 2. Comparing the running time in the case of changing $t$.

| Ex. | $t$ | $n$ | $d$ | Coeff. Range | Time$_1$(s) | Time$_2$(s) |
|-----|-----|-----|-----|--------------|-------------|-------------|
| 1   | 10  | 5   | 5   | [-100,100]   | 0.062       | 0.094       |
| 2   | 15  | 5   | 5   | [-100,100]   | 0.141       | 0.203       |
| 3   | 30  | 5   | 5   | [-100,100]   | 0.655       | 0.842       |
| 4   | 5   | 10  | 10  | [-100,100]   | 0.093       | 0.219       |
| 5   | 15  | 10  | 10  | [-100,100]   | 0.546       | 1.186       |
| 6   | 30  | 10  | 10  | [-100,100]   | 3.712       | 5.414       |

From Table 2, we observe that with the increase of $t$ the algorithm costs more time because the size of equation system is dominated by the number of terms $t$. 

1338
The systems of linear equations are solved by a computer with programs that use floating point arithmetic. The experiments have been implemented in Maple which is a symbolic computation environment. It can avoid round off error. You can see from Figure 2 the difference between Zippel’s algorithm and ours.

![Figure 2. Comparing the running time in the case of n=10, d=10.](image)

**Experiment 3**

Similarly, in the last set of tests, we let n and t fix and let d increase gradually.

| Ex. | d  | n  | t  | Coeff. Range | Time_1(s) | Time_2(s) |
|-----|----|----|----|--------------|-----------|-----------|
| 1   | 10 | 5  | 5  | [-100,100]   | 0.062     | 0.094     |
| 2   | 15 | 5  | 5  | [-100,100]   | 0.094     | 0.140     |
| 3   | 30 | 5  | 5  | [-100,100]   | 0.187     | 0.250     |
| 4   | 15 | 10 | 10 | [-100,100]   | 1.482     | 2.262     |
| 5   | 20 | 10 | 10 | [-100,100]   | 1.997     | 3.978     |
| 6   | 30 | 10 | 10 | [-100,100]   | 3.026     | 4.663     |

As expected the algorithm ran in more time when the total degree d is bigger. Because the construction of linear systems need set up coefficients matrix, the bigger degree d will cause the bigger coefficients. Therefore, the system solving is getting more complicated. It is illustrated by Figure 3.
In general, our method takes less time since our method needs fewer probes than Zippel’s.

CONCLUSION AND FUTURE WORK

In this paper, we present a variation of Zippel’s algorithm. Compared with Zippel’s method, our algorithm needs fewer probes. Therefore, the computation time can be less. Our multivariate polynomial interpolation algorithm is suitable for the lack of knowledge of degree bound $d$. To conclude, we have tried to demonstrate the testing pointing technique can be used to increase the effectiveness of the interpolation algorithms.

ACKNOWLEDGEMENT

We acknowledge the anonymous referees for their constructive comments.

The work was supported by Guangxi Key Laboratory of Cryptography and Information Security (No GCIS201615), the National Natural Science Foundation of China (No.11561015), the Natural Science Foundation of Guangxi Province (No. 2016GXNSFFA380009)

REFERENCES

[1] Timothy Sauer. Numerical Analysis. Beijing: China Machine Press. 2012.
[2] Michael Ben-Or, Prasoon Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In Proceedings of the twentieth annual ACM symposium on Theory of Computing (New York, USA, 1988), STOC’88, ACM, pp. 301-309.
[3] R. Zippel, Probabilistic algorithms for sparse polynomials, in: Proc. EUROSAM ’79, in: Lect. Notes Comput. Sci., vol. 72, Springer Verlag, 1979, pp. 216-226.
[4] A. Cuyt, W.-s. Lee, A new algorithm for sparse interpolation of multivariate polynomials, Theoret. Comput. Sci. 409 (2) (2008) 180-185.
[5] M. Giesbrecht, G. Labahn, W.-s. Lee, Symbolic-numeric sparse interpolation of multivariate polynomials (extended abstract), in: J.-G. Dumas (Ed.), ISSAC MMVI Proc. 2006 Internat. Symp. Symbolic Algebraic Comput., ACM Press, 2006, pp. 116-123.
[6] M. Giesbrecht, G. Labahn, W.-s. Lee, Symbolic-numeric sparse interpolation of multivariate polynomials, J. Symbolic Comput. 44 (8) (2009) 943–959.
[7] D.Y. Grigoriev, M. Karpinski, M.F. Singer, Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields, SIAM J. Comput. 19 (6) (1990) 1059-1063.

Figure 3. Comparing the running time in the case of $n=15$, $n=15$. 
[8] E. Kaltofen, W.-s. Lee, Early termination in sparse interpolation algorithms, J. Symbolic Comput. 36 (3-4) (2003) 365-400.

[9] E. Kaltofen, W.-s. Lee, A.A. Lobo, Early termination in Ben-Or/Tiwari sparse interpolation and a hybrid of Zippel's algorithm, in: C. Traverso (Ed.), Proc. 2000 Intern. Symp. Symbolic Algebraic Comput., ISSAC’00, ACM Press, 2000, pp. 192-201.

[10] E. Kaltofen, Z. Yang, On exact and approximate interpolation of sparse rational functions, in: C.W. Brown (Ed.), ISSAC’07: Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation, ACM, 2007, pp. 203-210.

[11] S. Khodadad, M. Monagan, Fast rational function reconstruction, in: J.-G. Dumas (Ed.), ISSAC MMVI Proc. 2006 Internat. Symp. Symbolic Algebraic Comput., ACM Press, 2006, pp. 184-190.

[12] Z. Olesh, A. Storjohann, The vector rational function reconstruction problem, in: Proceedings of the Waterloo Workshop: Computer Algebra 2006, Latest Advances in Symbolic Algorithms, World Scientific, 2007, pp. 137-149.

[13] A. Cuyt, W. Lee. Sparse interpolation of multivariate rational functions. Theoretical Computer Science. 412, 16 (2011), pp. 1445-1456.

[14] Jean-Michel Muller et al. Handbook of Floating-Point Arithmetic. Birkhauser Berlin. 2010.