

**A fast solver for a bivariate polynomial  
vector homogeneous interpolation  
problem.**

*Raf Vandebril , Marc Van Barel*

*Report TW 342, July 30, 2002*



**Katholieke Universiteit Leuven**  
Department of Computer Science  
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

# A fast solver for a bivariate polynomial vector homogeneous interpolation problem.

*Raf Vandebril , Marc Van Barel*

*Report TW 342, July 30, 2002*

Department of Computer Science, K.U.Leuven

## Abstract

In this paper we present a fast method for solving the following bivariate interpolation problem: Given the interpolation points  $(\omega_i, \xi_j)$  for  $i \in I$  and  $j \in J$  and the corresponding weights  $\Phi_{i,j}$  and  $\Psi_{i,j}$ , we look for a polynomial vector  $[p(x, y), q(x, y)]$  satisfying the following equation:

$$p(\omega_i, \xi_j)\Phi_{i,j} + q(\omega_i, \xi_j)\Psi_{i,j} = 0$$

for  $i \in I$  and  $j \in J$ . We solve the problem by solving smaller univariate interpolation problems, which we solve with the fast interpolation solver of Van Barel and Bultheel [?]. We rewrite the polynomials  $p(x, y)$  and  $q(x, y)$  in the following form:

$$\begin{aligned} p(x, y) &= p_0(y) + p_1(y)x + p_2(y)x^2 + \dots \\ q(x, y) &= q_0(y) + q_1(y)x + q_2(y)x^2 + \dots \end{aligned}$$

By substituting for  $y$  the values of  $\xi_j$  we get different univariate interpolation problems in  $x$ , which we solve with the fast univariate solver. This gives us a lot of univariate polynomials, which coefficients are polynomials evaluated in the interpolation points  $\xi_j$ . When we have enough values for these coefficients, we can also find the remaining unknown polynomials in  $y$ . The algorithm as presented here can be extended to multivariate problems, still being fast, and it is even possible to solve more general problems, with more unknown polynomials, but these are topics for future research. An application of this approach is the solving of a block Toeplitz matrix with circulant blocks, which can therefore be solved in a fast way.

The algorithm is implemented in matlab and results concerning the accuracy and efficiency are shown.

**Keywords :** Multivariate interpolation, univariate interpolation, fast solver, pivoting, bivariate interpolation problem, vector polynomial interpolation, block Toeplitz circulant block matrix

**AMS(MOS) Classification :** Primary : 65D05 Secondary : 41A05

# A fast solver for a bivariate polynomial vector homogeneous interpolation problem.\*

Raf Vandebril <sup>†</sup>, Marc Van Barel <sup>‡</sup>

30th July 2002

## Abstract

In this paper we present a fast method for solving the following bivariate interpolation problem: Given the interpolation points  $(\omega_i, \xi_j)$  for  $i \in I$  and  $j \in J$  and the corresponding weights  $\Phi_{i,j}$  and  $\Psi_{i,j}$ , we look for a polynomial vector  $[p(x, y), q(x, y)]$  satisfying the following equation:

$$p(\omega_i, \xi_j)\Phi_{i,j} + q(\omega_i, \xi_j)\Psi_{i,j} = 0$$

for  $i \in I$  and  $j \in J$ . We solve the problem by solving smaller univariate interpolation problems, which we solve with the fast interpolation solver of Van Barel and Bultheel [13]. We rewrite the polynomials  $p(x, y)$  and  $q(x, y)$  in the following form:

$$\begin{aligned} p(x, y) &= p_0(y) + p_1(y)x + p_2(y)x^2 + \dots \\ q(x, y) &= q_0(y) + q_1(y)x + q_2(y)x^2 + \dots \end{aligned}$$

By substituting for  $y$  the values of  $\xi_j$  we get different univariate interpolation problems in  $x$ , which we solve with the fast univariate solver. This gives

---

\*This research was partially supported by the Fund for Scientific Research–Flanders (FWO–V), project “SMA: Structured Matrices and their Applications” grant #G.0078.01, by the K.U.Leuven (Bijzonder Onderzoeksfonds), project “SLAP: Structured Linear Algebra Package,” grant #OT/00/16 and by the Belgian Programme on Interuniversity Poles of Attraction, initiated by the Belgian State, Prime Minister’s Office for Science, Technology and Culture. The scientific responsibility rests with the authors

<sup>†</sup>E-mail: raf.vandebril@cs.kuleuven.ac.be

<sup>‡</sup>E-mail: marc.vanbarel@cs.kuleuven.ac.be

us a lot of univariate polynomials, which coefficients are polynomials evaluated in the interpolation points  $\xi_j$ . When we have enough values for these coefficients, we can also find the remaining unknown polynomials in  $y$ . The algorithm as presented here can be extended to multivariate problems, still being fast, and it is even possible to solve more general problems, with more unknown polynomials, but these are topics for future research. An application of this approach is the solving of a block Toeplitz matrix with circulant blocks, which can therefore be solved in a fast way.

The algorithm is implemented in matlab and results concerning the accuracy and efficiency are shown.

**Keywords: Multivariate interpolation, univariate interpolation, fast solver, pivoting, bivariate interpolation problem, vector polynomial interpolation, block Toeplitz circulant block matrix**

## 1 Introduction

The multivariate polynomial interpolation problem is a very interesting, but not so simple problem. Not so simple, because very useful properties from the univariate case cannot be used anymore, and also the growing number of variables makes the problem more complicated than the univariate one. Already a lot of research has been done on this topic. Annie Cuyt performed a lot of research in the field of multivariate Padé approximations [6, 5, 4], and made a comparison between different types of multivariate solvers in the Padé case [3]. Carl De Boor also performed some interesting research in the field of multivariate polynomials [9, 8, 7]. An historical overview of the multivariate interpolation problem can be found in a paper by Sauer and Gasca [10].

The previous paper we presented [19] on this topic, was a straightforward extension of an algorithm of Van Barel and Bultheel [14], for solving univariate interpolation problems. Although this algorithm was slow, it gave a nice alternative and another point of view compared to the existing multivariate interpolation tools. Whereas our previous paper dealt with a quite specific interpolation problem (even though it could be generalized), the problem we solve here is more general, but limited to the bivariate case. More general in the sense, that it can easily be extended or adapted to fit other problems.

The speed of our algorithm follows from the fact that we split everything up into smaller, univariate interpolation problems, which can be solved fast (or even superfast). We fix the value of  $y$  (not symbolically but numerically, i.e., fill in one of the interpolation points) and we solve all the existing univariate interpolation

problems arising in  $x$ . With all the corresponding coefficients of the solutions in  $x$ , we can then solve the remaining univariate problems in  $y$ . The advantage of this approach is that the multivariate interpolation problem is reduced to several univariate interpolation problems, which can be solved by an arbitrary solver for these kind of problems, e.g. the fast solver of Van Barel and Bultheel [14].

The technique we present here in fact resembles a lot the one presented in the papers of Guillaume [11] and Chaffy [2], but the approach they use is a symbolic approach. They fix the value of  $y$ , and then symbolically solve the problem in  $x$ . What we do is significantly different because we solve the problem for specific values of  $y$ . We do not work symbolically, but numerically. This other approach gains us a lot in speed.

Just like in the univariate case, our algorithm could be a very useful tool, also for other topics. For example in the univariate case we refer to the papers by Bultheel, Van Barel, Kravanja and Heinig: superfast Toeplitz solvers [17], fast Hankel and Loewner matrix solvers [12, 18], connections with rational interpolation techniques [15]. All these papers are just a few cases where the univariate interpolation problem is a very useful and helpful tool. We are sure that our solver can also be adapted to solve several problems which can be translated into bivariate interpolation problems, e.g., signal processing problems and block Toeplitz Toeplitz block systems. More detailed information about the interpolation solver we use can be found in the book of Van Barel and Bultheel [1]. In chapter 7 a theoretical framework about interpolation can be found and also interesting links to papers and books concerning this topic.

Our paper is limited to the bivariate problem for several reasons. First of all because of the readability, dealing with more variables, increases the notational cost and decreases the understandability of the algorithm. The second reason is the possibility to extend this algorithm. Because it is a very compact and general problem, limited to two variables, and two unknown polynomials, it is easily extended to larger scale problems with more variables.

Our paper is divided in the following sections. In the first section we will briefly introduce the problem and some notations. The second section will deal with effective problem solving. An algorithmic description will be given in the third section, the next section shows the different test results for the algorithm, showing that it is efficient and accurate. In the final section we explain an application of this algorithm, we transform a block Toeplitz matrix with circulant blocks into an interpolation problem of the form described above, and therefore we are able to solve it in a fast and accurate way.

## 2 The interpolation problem

In this first explanatory section, we will situate the problem we will solve, and we will also introduce all the notations we will use throughout this paper. The polynomials we consider are bivariate polynomials denoted with  $p(x, y)$  and  $q(x, y)$ . The problem we will solve, is more general then the coordinate problem we solved in [19]. Here we will search for the polynomials  $p(x, y)$  and  $q(x, y)$  satisfying the following conditions:

$$p(\omega_i, \xi_j)\Phi_{i,j} + q(\omega_i, \xi_j)\Psi_{i,j} = 0 \quad (1)$$

for  $i \in I$  and  $j \in J$ , where the  $\omega_i$  and  $\xi_j$  denote the interpolation points, the  $\Phi_{i,j}$  and  $\Psi_{i,j}$  are just scalars depending on  $i, j$ . When we assume that the degree in  $x$  is limited by  $d_1$  and the degree in  $y$  by  $d_2$ , we can make more assumptions about the number of interpolation points and the sizes of  $\Phi = [\Phi_{i,j}]_{i \in I, j \in J}$  and  $\Psi = [\Psi_{i,j}]_{i \in I, j \in J}$ . First of all  $I$  and  $J$  will respectively equal  $1, 2, \dots, 2d_1 + 1$  and  $1, 2, \dots, d_2 + 1$  (this will become clear in the next section). The matrices  $\Phi$  and  $\Psi$  will be of size  $(2d_1 + 1) \times (d_2 + 1)$  and we have that  $\Omega = (\omega_1, \omega_2, \dots, \omega_{2d_1+1})$  and  $\Xi = (\xi_1, \xi_2, \dots, \xi_{d_2+1})$ .

As mentioned before we will rewrite the polynomials  $p(x, y)$  and  $q(x, y)$  in the following form:

$$p(x, y) = p_0(y) + p_1(y)x + p_2(y)x^2 + \dots + p_{d_1-1}(y)x^{d_1-1} + p_{d_1}(y)x^{d_1} \quad (2)$$

$$q(x, y) = q_0(y) + q_1(y)x + q_2(y)x^2 + \dots + q_{d_1-1}(y)x^{d_1-1} + q_{d_1}(y)x^{d_1} \quad (3)$$

with all the  $p_i(y)$  and  $q_i(y)$  polynomials in  $y$  of maximal degree  $d_2$ .

Counting the number of unknown parameters in the two polynomials  $p(x, y)$  and  $q(x, y)$  (2,3) gives us  $2(d_1 + 1)(d_2 + 1)$ , compared to the number of interpolation conditions  $(2d_1 + 1)(d_2 + 1)$  we get a difference of  $(d_2 + 1)$ , this means that there are still  $(d_2 + 1)$  parameters which are not determined by the interpolation-conditions. We still have to add conditions to get a unique solution to our problem. We will more clearly situate this problem in the next section.

## 3 Solving the bivariate interpolation problem

The speed of our algorithm is determined by the speed of the method to solve the different univariate interpolation problems. These univariate problems can be solved using the fast algorithm of Van Barel and Bultheel [13] or even with their

superfast solver [14]. In the implementation in this paper we choose for the fast solver because even with this fast solver, our algorithm will be very efficient and accurate, as can be seen in the section with the results.

As mentioned before we will consider a grid of interpolation points, of dimensions  $(2d_1 + 1) * (d_2 + 2)$ , with  $\Omega$  denoting the interpolation points corresponding to  $x$  and  $\Xi$  the interpolation points corresponding to  $y$ .

It is easily seen that for every different choice of  $y$  equal to a certain  $\xi_k$  we get a univariate interpolation problem in  $x$ . This is in fact how we will solve the problem. For every different value of  $y$  equal to a certain  $\xi_k$  we solve the corresponding univariate interpolation problem in  $x$ ,  $\forall k \in \{1, \dots, d_2 + 1\}$  we get a univariate interpolation problem of the following form:

$$p(\omega_i, \xi_k)\Phi_{i,k} + q(\omega_i, \xi_k)\Psi_{i,k} = 0 \quad (4)$$

with  $i$  varying in the range  $1, \dots, 2d_1 + 1$ . We only consider  $(2d_1 + 1)$  interpolation points, but we search two polynomials of degree  $d_1$ , we have one condition less, because we are solving a homogeneous interpolation problem, this means that the solution we will find here is not uniquely determined, i.e. when  $[p(x, \xi_k), q(x, \xi_k)]$  is a vector solution to the homogeneous interpolation problem (4) then also the vector  $[\alpha_k p(x, \xi_k), \alpha_k q(x, \xi_k)]$ ,  $\forall \alpha_k$  is a solution to the homogeneous problem (4). This brings us back to the problem mentioned at the end of the previous section.

These unknown parameters  $\alpha_k$ , there are  $d_2 + 1$  of them, can be determined in several ways. We shall mention some of the possibilities.

- A possible solution, is to make the polynomials monic in  $x$ , because this determines the factors  $\alpha_k$ , and so we get  $(d_2 + 1)$  extra conditions.
- You can also determine one of the polynomials  $p_i(y)$  or  $q_i(y)$ , because they are both of degree  $d_2$  we get the extra  $(d_2 + 1)$  conditions.
- Take a new interpolation point, lets say  $x_0$ , and place the following conditions on the polynomials:

$$\begin{cases} p(x_0, \xi_1)\Phi_{0,1} + q(x_0, \xi_1)\Psi_{0,1} & = b_1 \\ \vdots & \\ p(x_0, \xi_{d_2})\Phi_{0,d_2} + q(x_0, \xi_{d_2})\Psi_{0,d_2} & = b_{d_2} \\ p(x_0, \xi_{d_2+1})\Phi_{0,d_2+1} + q(x_0, \xi_{d_2+1})\Psi_{0,d_2+1} & = b_{d_2+1} \end{cases}$$

Where  $\forall i \in \{1, \dots, d_2 + 1\}$  the values  $b_i, \Phi_{0,i}, \Psi_{0,i}$  can be chosen freely (with the  $b_i$ 's different from 0). This gives us again  $(d_2 + 1)$  conditions needed for the unicity of the solution.

- You can take a look at the problem as a sort of Lagrange interpolation, by rewriting it as

$$\sum_{j=1}^{d_2+1} \left( p(x, \xi_j) \Phi_j + q(x, \xi_j) \Psi_j \right) \left( \prod_{\substack{i=1 \\ i \neq j}}^{d_2+1} (y - \xi_i) \right)$$

(In this final formula we are a little inconsistent, with regard to  $\Phi$  and  $\Psi$ , because they are linked to the interpolation points  $\Omega$ .) This again gives possibilities for placing  $(d_2 + 1)$  extra conditions on the problem.

- It is even not always necessary to place  $d_2 + 1$  conditions on the problem, when one wants a homogeneous solution to the problem  $d_2$  conditions is enough, you can then define all the  $(\alpha_1, \alpha_2, \dots, \alpha_{d_2+1})$  up to a constant factor  $\alpha$ . This  $\alpha$  can then be seen as a factor corresponding to the final solution, i.e., when the final vector solution equals  $[p(x, y), q(x, y)]$  then  $[\alpha p(x, y), \alpha q(x, y)]$  is also a solution to the original problem.

When having the intermediate results (the solutions of the problem towards  $x$ ), the other  $(d_2 + 1)$  free parameters have to be determined, this can be done by placing one of the conditions above on these parameters.

After having solved  $d_2 + 1$  interpolation problems, with extra conditions, we have  $2(d_2 + 1)$  unique univariate polynomials in  $x$ . These polynomials corresponding to  $p(x, \xi_k)$  and  $q(x, \xi_k)$  can be written in the following form:

$$\left\{ \begin{array}{l} p(x, \xi_1) = p_0(\xi_1) + p_1(\xi_1)x + p_2(\xi_1)x^2 + \dots + p_{d_1}(\xi_1)x^{d_1} \\ p(x, \xi_2) = p_0(\xi_2) + p_1(\xi_2)x + p_2(\xi_2)x^2 + \dots + p_{d_1}(\xi_2)x^{d_1} \\ \vdots \\ p(x, \xi_{d_2+1}) = p_0(\xi_{d_2+1}) + p_1(\xi_{d_2+1})x + \dots + p_{d_1}(\xi_{d_2+1})x^{d_1} \end{array} \right.$$

and the following  $d_2 + 1$  polynomials for  $q$ :

$$\left\{ \begin{array}{l} q(x, \xi_0) = q_0(\xi_0) + q_1(\xi_0)x + q_2(\xi_0)x^2 + \dots + q_{d_1}(\xi_0)x^{d_1} \\ q(x, \xi_1) = q_0(\xi_1) + q_1(\xi_1)x + q_2(\xi_1)x^2 + \dots + q_{d_1}(\xi_1)x^{d_1} \\ \vdots \\ q(x, \xi_{d_2+1}) = q_0(\xi_{d_2+1}) + q_1(\xi_{d_2+1})x + \dots + q_{d_1}(\xi_{d_2+1})x^{d_1} \end{array} \right.$$

We still have  $2(d_1 + 1)$  unknown polynomials  $p_i(y)$  and  $q_i(y)$  to solve, however the equations above give us enough information to determine these remaining unknown polynomials in  $y$ , because for every polynomial  $p_i(y)$  (as well as for the polynomials  $q_i(y)$ ) we have the values of these polynomials corresponding to the interpolation points  $\Xi = (\xi_1, \xi_2, \dots, \xi_{d_1+1})$  namely:  $(p_i(\xi_0), p_i(\xi_1), \dots, p_i(\xi_{d_1+1}))$  (the same for the polynomials  $q_i(y)$ ). Again these final problems can easily be solved using the same interpolation solver from Van Barel and Bultheel [13]. We have not further discussed the unicity of the solution. The first problem we solved was not uniquely determined because we were searching for the solution of a homogeneous problem, when however we place one of the conditions mentioned above on these solutions, we can find a unique solution to these homogeneous interpolation problem, this means that our first part is solved in a unique way. For the second part we search for polynomials in  $y$ , which at the interpolation points  $\Xi$  attain certain values determined by the coefficients of the polynomials in  $x$ , because the interpolation points are all different we get linear independent conditions and we get a unique solution. This means that we have found a unique solution to our problem.

With this approach we have solved the bivariate interpolation problem by using only univariate interpolation techniques. The complexity of this algorithm and the accuracy are topics of section 5.

## 4 The algorithm

In this section we will describe the algorithm in detail using matlab style notation. We will briefly repeat the problem and notations used. We search for the bivariate polynomials  $p(x, y)$  and  $q(x, y)$ , satisfying the following equation:

$$p(\omega_i, \xi_j)\Phi_{i,j} + q(\omega_i, \xi_j)\Psi_{i,j} = 0,$$

for  $i \in \{1, \dots, 2d_1 + 1\}$  and  $j \in \{1, \dots, d_2 + 1\}$ . This corresponds with the degree of  $x$  limited to  $d_1$  and the degree of  $y$  limited to  $d_2$ .

**Definition 1** *Let us first briefly review the notations we will use when describing the algorithm*

- *The variables  $\Omega, \Xi, \Phi$  and  $\Psi$  are defined in the same way as before.*
- *$\Phi_i$  and  $\Psi_i$  denote the  $i$ th column of the matrix  $\Phi$  respectively  $\Psi$ .*

- **solpx** and **solqx**, in these matrices we will store the intermediate univariate polynomials in  $x$ , with each row corresponding to another value of  $y$ .

The multivariate polynomials will be stored in a matrix, using the following form:

$$q(x, y) = q_0(y) + q_1(y)x + q_2(y)x^2 + \cdots + q_{d_1-1}(y)x^{d_1-1} + q_{d_1}(y)x^{d_1}$$

so that the first column corresponds to  $q_0(y)$  the second column to  $q_1(y)$  and the  $i$ th column to  $q_i(y)$ . The first component of every column corresponds to the coefficient of degree 0, the second to the coefficient of degree 1 and so on. The polynomial  $q(x, y)$  from above will thus be stored in a matrix in the following way:

$$\begin{cases} 1 \rightarrow \\ y \rightarrow \\ y^2 \rightarrow \\ \vdots \rightarrow \\ y^{d_2} \rightarrow \end{cases} \begin{pmatrix} \overbrace{1 \quad x \quad x^2 \quad \cdots \quad x^{d_1}} \\ \downarrow \quad \downarrow \quad \downarrow \quad \quad \downarrow \\ q_{0,0} \quad q_{1,0} \quad q_{2,0} \quad \cdots \quad q_{d_1,0} \\ q_{0,1} \quad q_{1,1} \quad \ddots \quad \quad q_{d_1,1} \\ q_{0,2} \quad \ddots \quad \ddots \quad \quad \vdots \\ \vdots \\ q_{0,d_2} \quad q_{1,d_2} \quad \cdots \quad q_{d_1,d_2} \end{pmatrix}$$

**algorithm 1 (The polvecint procedure)** *The **polvecint** procedure solves a one dimensional interpolation problem of the following form: Consider  $p(x)$  and  $q(x)$  as univariate polynomials in  $x$ ,  $\Phi$  and  $\Psi$  are vectors filled with scalars and  $\Omega = (\omega_1, \omega_2, \dots, \omega_{n_1+n_2+1})$  corresponding to the interpolation points. Then the **polvecint** procedure returns us the polynomials  $p(x)$  and  $q(x)$  of degree respectively  $n_1$  and  $n_2$  satisfying the following equation:*

$$p(\omega_i)\Phi_i + q(\omega_i)\Psi_i = 0. \quad (5)$$

We denote this procedure as follows:

**procedure**  $[\mathbf{p}, \mathbf{q}] := \mathbf{polvecint}(\Omega, \Pi, \Psi, n_1, n_2)$ .

This **polvecint** procedure does not return a unique solution, (however unique up to a constant). For a description of how to deal with this we refer to section 3.

We are now ready to describe the main algorithm for solving the bivariate interpolation problem. The algorithm consists of three large parts. First of all, we solve interpolation problems in  $x$  for different values of  $\xi_j$  and then in the next two parts we solve the problems towards  $y$ .

**algorithm 2 (The main algorithm)** *We start with the first loop to construct the univariate polynomials in  $x$*

*# Solving the problem towards  $x$ .*

*for  $i = 1 : d_2 + 1$*

**[ $\mathbf{p}, \mathbf{q}$ ] := polvecint( $\Omega, \Phi_i, \Psi_i, d_1, d_1$ );**

**solpx( $i, :$ ) =  $\mathbf{p}$ ;**

**solqx( $i, :$ ) =  $\mathbf{q}$ ;**

*end*

- *Determining the remaining  $(d_2 + 1)$  unknown parameters.*

*# Solving the problem towards  $y$  for  $p$ .*

*for  $i = 1 : d_1 + 1$*

**[ $\mathbf{p}, \mathbf{q}$ ] := polvecint( $\Xi, 1, (-1) * \text{solpx}(:, i), d_2, 0$ );**

**$\mathbf{P}(:, i) = \mathbf{p}'$ ;**

*end*

*# Solving the problem towards  $y$  for  $q$*

*for  $i = 1 : d_1 + 1$*

**[ $\mathbf{p}, \mathbf{q}$ ] := polvecint( $\Xi, 1, (-1) * \text{solqx}(:, i), d_2, 0$ );**

**$\mathbf{Q}(:, i) = \mathbf{p}'$ ;**

*end*

**RETURN  $\mathbf{P}$  and  $\mathbf{Q}$**

*end of procedure*

In the next section it will be shown that the algorithm is efficient and stable.

## 5 Timings and stability issues

We made an implementation of the algorithm in matlab <sup>1</sup>. For the interpolation points we tried roots of unity, equal spaced and Chebyshev interpolation points. As can be seen in the next figures, roots of unity give us the most accurate results.

Instead of using the superfast version of the rational interpolation solver we used the  $O(n^2)$  algorithm, however, even when we use this complexity our algorithm is even faster than  $O(n\sqrt{n})$ , and can therefore be considered to be superfast. The reason why we prefer to use the  $O(n^2)$  univariate interpolation solver instead of the  $O(n\log^2(n))$  version, is because  $O(n\log^2(n)) \approx \alpha n\log^2(n) + O(n)$  and  $\alpha$  is a quite large constant. Therefor the  $O(n^2)$  version will in fact be faster then the  $O(n\log^2(n))$  version for not too large values of  $n$ . To find a unique solution we had to insert some more conditions on the polynomials, we chose to find polynomials which are monic with regard to the variable  $x$  in  $p(x, y)$ .

The data of the tests we performed were randomly generated, i.e., the matrices  $\Phi$  and  $\Psi$  are randomly generated within  $[0, 1]$ . For the first figure the interpolation points are the roots of unity.

Figure 1 shows us the 1-norm of the residual generated by the algorithm. For a fixed size always five problems of the same dimension were considered. You can see that for problems up to the size of two million the error stays within the range of the acceptable. The residual that we calculated here is an absolute residual, and is the 1-norm of the matrix  $M$  which consists of the following values:

$$M_{i,j} = (p(\omega_i, \xi_j)\Phi_{i,j} + q(\omega_i, \xi_j)\Psi_{i,j})_{i \in I, j \in J}$$

When dealing with other kinds of interpolation points, it is very important to choose a good representation of the polynomials. When using a representation in an orthogonal basis satisfying a three terms recurrence, the complexity of the algorithm does not change (only the constant factor may increase a bit (This factor corresponds with  $\beta$  in 6)), but the accuracy can increase dramatically. We performed tests on equal spaced interpolation points, with the standard basis and the Chebyshev basis, the second basis gave a solution which was more accurate, up to a factor 2, i.e., when the first only had 4 accurate digits the second had about 8 accurate digits. This means that it is extremely important to take a good basis representation, to get accurate results.

Another obvious example that states our assumptions here above, is the following. We consider the same interpolation problem, which we try to solve now

---

<sup>1</sup>matlab is a registrered trademark of the Mathworks Inc.

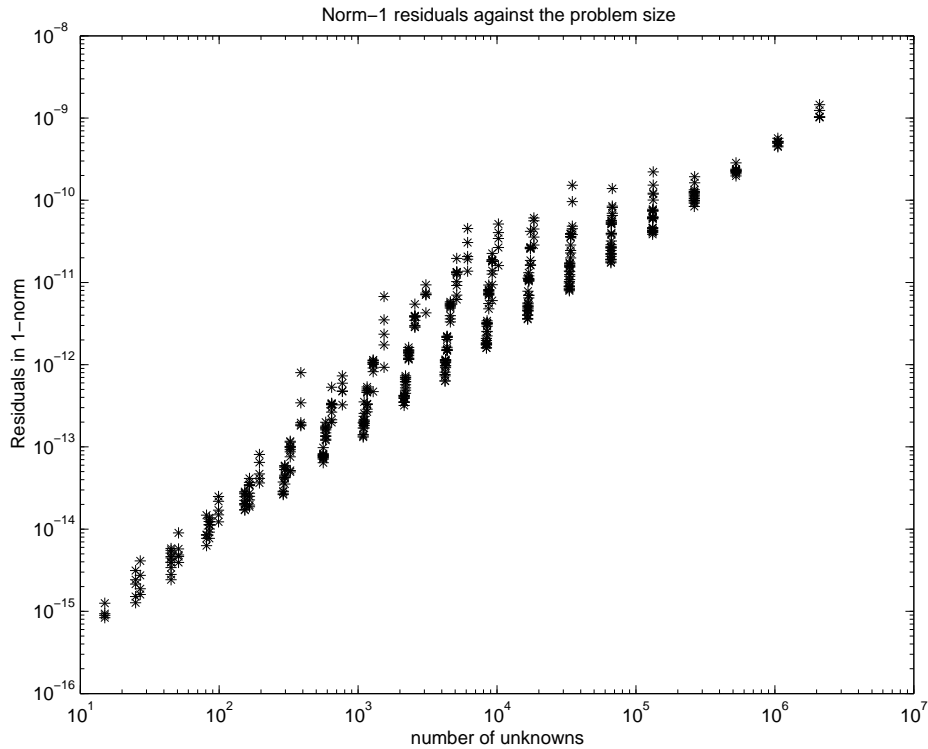


Figure 1: Norm-1 of the absolute residual for roots of unity.

in Chebyshev interpolation points generated in the interval  $[0, 1]$ . Figure 2 corresponds to the solution of this problem when using the standard basis representation. The residual that is displayed is again the absolute residual.

As can immediately be seen in the graph, very quick all the significant digits are lost, and the error keeps growing, in this case, with this polynomial representation the algorithm does not work at all. In Figure 3 the same problems are solved, but now, instead of presenting the polynomials in the standard basis we present them in the Chebyshev basis, this adaptation in the algorithm, does not increase the complexity, but dramatically increases the numerical stability, as the figure clearly shows.

These final examples show once more that it is very important to choose a good basis representation for the polynomials, in order to get accurate results.

We will now briefly deduce the asymptotic complexity of the algorithm that we use (when using the superfast rational interpolation approach, the complexity

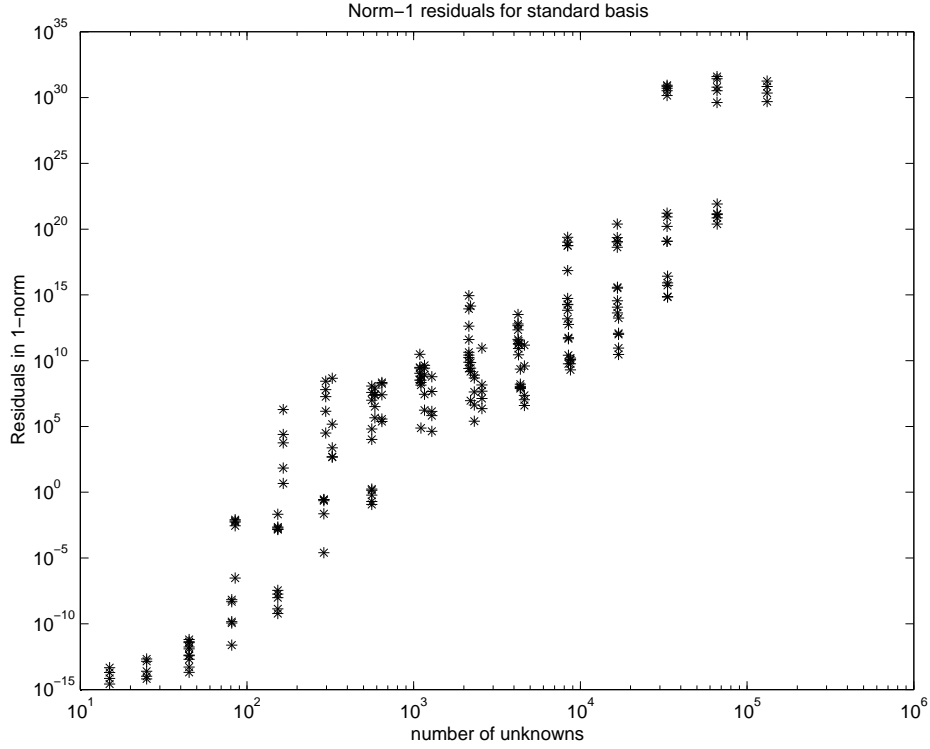


Figure 2: Norm-1 for Chebyshev interpolation points, and the standard polynomial representation.

is of course even smaller). We assume that the rational interpolation solver is of order  $O(n^2)$ . The first step consists of solving  $(d_2 + 1)$  interpolation problems of order  $(2d_1 + 1)$ . The next step consists of solving  $2(d_1 + 1)$  interpolation problems of order  $(d_2 + 1)$ , this gives us as complexity, when we take

$$\approx \beta n^2 + O(n) \tag{6}$$

(and we neglect the term  $O(n)$ ):

$$\begin{aligned} & \beta(d_2 + 1) ((2d_1 + 1)^2) + 2(d_1 + 1) ((d_2 + 1)^2) \\ \leq & \beta(d_2 + 1) 4 ((d_1 + 1)^2) + 4(d_1 + 1) ((d_2 + 1)^2) \\ \leq & 4\beta(d_2 + 1)(d_1 + 1) (d_1 + 1 + d_2 + 1) \end{aligned}$$

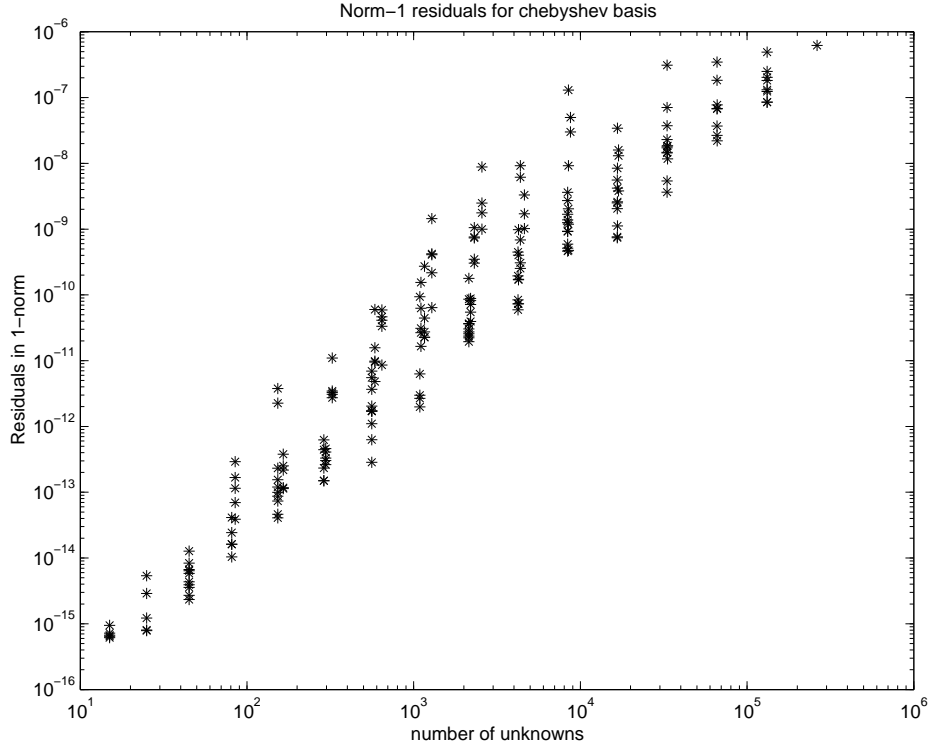


Figure 3: Norm-1 for Chebyshev interpolation points, and the Chebyshev polynomial representation.

The number of unknown parameters in the system we solve equals  $2(d_2 + 1)(d_1 + 1)$ , when we denote this number with  $n$ , we clearly get:

$$\begin{aligned}
 & 4\beta(d_2 + 1)(d_1 + 1)(d_1 + 1 + d_2 + 1) \\
 &= 2n\beta(d_1 + 1 + d_2 + 1) \\
 &\leq O(n^2).
 \end{aligned}$$

This means that our algorithm is faster than  $n^2$  and therefore superfast (while we have not even used the superfast solver for the intermediate interpolation problems).

When using the superfast solver we get the following complexity:

$$O(n(\log^2(d_1 + 1) + \log^2(d_2 + 1))).$$

And when we consider a special case namely  $d_1 = d_2$  then the fast univariate solver leads to a complexity a little smaller than  $O(n\sqrt{n})$ .

The next two figures are comparisons between the timed speed and the theoretical speed we deduced. You can see in Figure 4, which only shows the results for  $d_1 = d_2$  that the algorithm is faster than  $O(n\sqrt{n})$ . There is also a comparison between our solver and Gaussian elimination, you can see that the crossover appears to be lying around 1000, so our algorithm is not only theoretically faster, but even in practical applications, our algorithm performs better than Gaussian elimination.

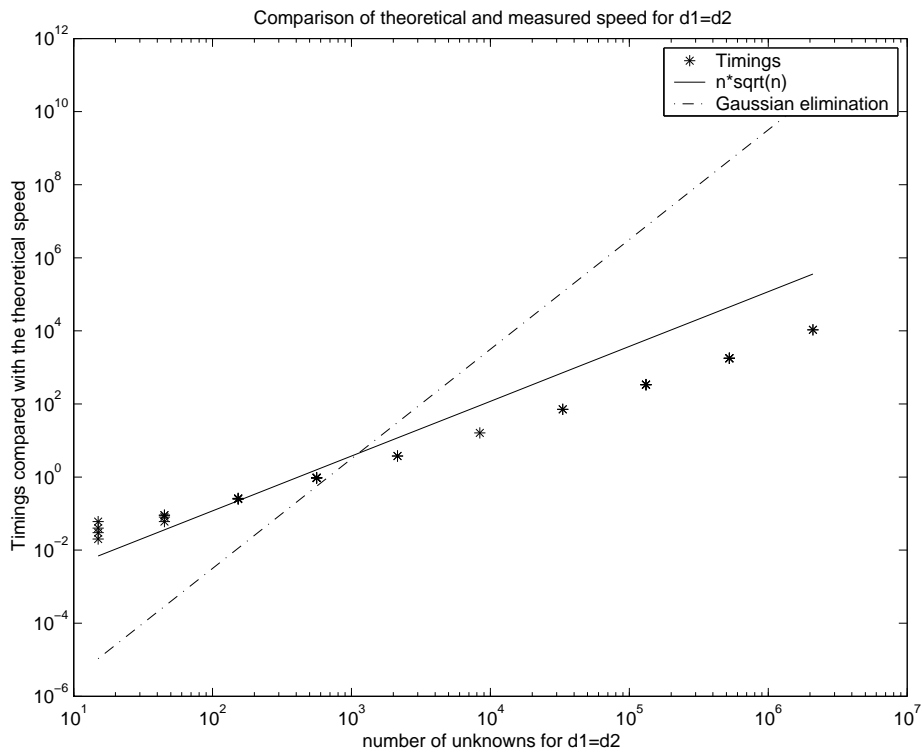


Figure 4: Timings for  $d_1 = d_2$  compared with  $O(n\sqrt{n})$  and Gaussian-elimination.

The last figure makes a comparison between the timed results and Gaussian elimination. In this figure also measurements for  $d_1 \neq d_2$  are shown.

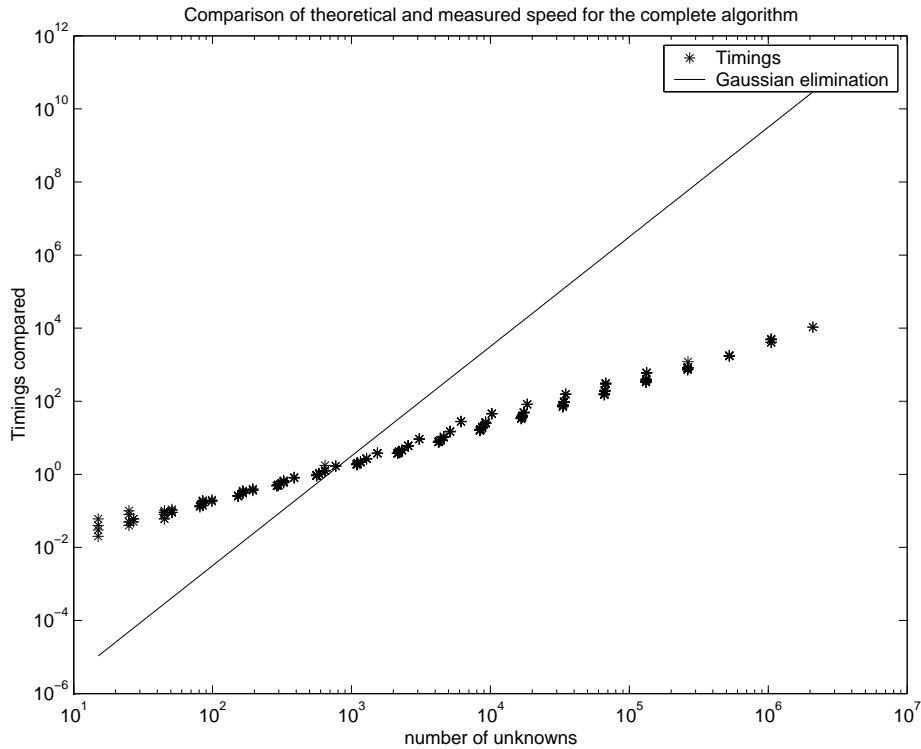


Figure 5: Timings compared with Gaussian elimination.

## 6 An application: Solving a block Toeplitz matrix with circulant blocks in a fast and accurate way

This final section deals with the block Toeplitz problem with circulant blocks. We will transform this problem, because of the block structure into a bivariate interpolation problem. All the conditions to solve this bivariate interpolation problem with our specific technique will be satisfied and therefore we will be able to solve this problem in a fast and accurate way.

Before starting the transformation of the BTCB problem we will introduce all the new notations, for dealing with this problem. Suppose we have the following matrix problem, for a certain natural number  $n_2$ .

$$\begin{pmatrix} C_0 & C_{-1} & \cdots & C_{-n_2} \\ C_1 & C_0 & & \vdots \\ \vdots & & \ddots & \\ C_{n_2} & \cdots & & C_0 \end{pmatrix} \begin{pmatrix} P_0 \\ P_1 \\ \vdots \\ P_{n_2} \end{pmatrix} = \begin{pmatrix} B_0 \\ B_1 \\ \vdots \\ B_{n_2} \end{pmatrix}$$

Where the  $C_i$  are circulant matrices of dimension  $n_1$  and the  $P_i$  and  $B_i$  are vectors of dimension  $n_1$  of the following form (with the  $B_i$  constructed in a similar way as the  $P_i$ )

$$C_i = \begin{pmatrix} c_0^{(i)} & c_{n_1}^{(i)} & \cdots & c_1^{(i)} \\ c_1^{(i)} & c_0^{(i)} & & c_2^{(i)} \\ \vdots & & \ddots & \vdots \\ c_{n_1}^{(i)} & \cdots & & c_0^{(i)} \end{pmatrix}, P_i = \begin{pmatrix} p_0^{(i)} \\ p_1^{(i)} \\ \vdots \\ p_{n_1}^{(i)} \end{pmatrix}$$

The first step in transforming the matrix problem into an interpolation problem, is rewriting the block system into a matrix-vector interpolation problem. This is an extension of the univariate transformation of a Toeplitz matrix into an interpolation problem, see for example the paper of Van Barel, Heinig and Kravanja [16], which exploits this transformation.

We introduce the following polynomials in  $x$ ,

$$\begin{aligned} C(x) &= \sum_{i=-n_2}^{n_2} C_i x^i \\ P(x) &= \sum_{i=0}^{n_2} P_i x^i \\ B(x) &= \sum_{i=0}^{n_2} B_i x^i. \end{aligned}$$

It is an easy calculation to show that the BTCB problem is equivalent to the following polynomial representation:

$$\prod_{x=0}^{n_2} C(x)P(x) = B(x)$$

$\prod_{x=0}^{n_2}$  denotes the projection of the polynomials between the degrees 0 and  $n_2$ . When dropping the projection we have to add two more polynomials, coming

from the degrees lower and higher than respectively 0 and  $n_2$ . Call these polynomials  $A(x)$  and  $E(x)$ . Remark that also the polynomials  $A(x)$  and  $E(x)$  are vector polynomials. This gives us the next equation:

$$C(x)P(x) = B(x) + x^{-n_2}A(x) + x^{n_2+1}E(x) \quad (7)$$

Where the two new polynomials  $A(x)$  and  $C(x)$  are both of degree  $n_2 - 1$ .

To continue with our polynomial reduction, we have to use a well known property of circulant matrices, namely they can be diagonalized very cheaply by means of FFT's. However here we are working with matrix polynomials where all the matrices are circulant matrices, therefore applying FFT's on these matrix polynomials gives us a diagonal matrix, where the diagonals are polynomials:

$$F * C(x) * F^H = D(x).$$

As mentioned above,  $D(x)$  denotes a diagonal matrix where the diagonals are polynomials. Substituting this into Equation (7) reduces our problem to:

$$\begin{aligned} & D(x) * F * P(x) \\ = & F * B(x) + x^{-n_2}F * A(x) + x^{n_2+1}F * E(x) \end{aligned}$$

Remark now that for  $q(y) = \sum_{i=0}^{n_1} q_i y^i$

$$\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_1 & \omega_1^2 & \cdots & \omega_1^{n_1} \\ 1 & \omega_2 & \omega_2^2 & \cdots & \omega_2^{n_1} \\ \vdots & \ddots & & & \\ 1 & \omega_{n_1} & \omega_{n_1}^2 & \cdots & \omega_{n_1}^{n_1} \end{pmatrix} * \begin{pmatrix} q_0 \\ q_1 \\ \vdots \\ q_{n_1} \end{pmatrix} = \begin{pmatrix} q(\omega_0) \\ q(\omega_1) \\ \vdots \\ q(\omega_{n_1}) \end{pmatrix}.$$

This means that we can interpret the multiplication  $F * P(x)$  with  $P_i(y) = \sum_{k=0}^{n_1} P_{i,k} y^k$  as:

$$F * P(x) = \begin{pmatrix} \sum_{i=0}^{n_2} P_i(\omega_0) x^i \\ \sum_{i=0}^{n_2} P_i(\omega_1) x^i \\ \vdots \\ \sum_{i=0}^{n_2} P_i(\omega_{n_1+1}) x^i \end{pmatrix}$$

Exactly the same can be done for  $A(x)$ ,  $E(x)$  and  $B(x)$ . We can now construct the multivariate polynomial  $P(x, y)$  in the following way:

$$P(x, y) = \sum_{i=0}^{n_2} P_i(y) x^i.$$

After similar transformations for the other vector polynomials, we can rewrite the main problem in the following way:

$$D(x) \begin{pmatrix} P(x, \omega_0) \\ P(x, \omega_1) \\ \vdots \\ P(x, \omega_{n_1}) \end{pmatrix} = \begin{pmatrix} B(x, \omega_0) \\ B(x, \omega_1) \\ \vdots \\ B(x, \omega_{n_1}) \end{pmatrix} \\ + x^{-n_2} \begin{pmatrix} A(x, \omega_0) \\ A(x, \omega_1) \\ \vdots \\ A(x, \omega_{n_1}) \end{pmatrix} + x^{n_2+1} \begin{pmatrix} E(x, \omega_0) \\ E(x, \omega_1) \\ \vdots \\ E(x, \omega_{n_1}) \end{pmatrix}$$

When we take a closer look at this equation, we see that the  $n_1 + 1$  rows can be transformed into an interpolation problem of the desired structure. However there are a few small differences. Instead of two components ( $p, q$  from the interpolation problem) we have here four components namely ( $P, B, A, E$ ). Where in the previous problem we could freely choose our interpolation points both for  $x$  and  $y$ , here the interpolation points for  $y$  are already fixed. The final difference is that in the interpolation problem from section 1 the  $\Phi$  and  $\Psi$  are independent from the choice of the interpolation points. Here however the  $\Phi$  and  $\Psi$  are built from  $D(x), x^{-n_2}$  and  $x^{n_2+1}$  and therefore dependent on the interpolation points. However none of these small differences changes the approach, explained in the first sections.

The complexity of solving this interpolation problem does not increase, because we can use FFT's. A quick calculation gives us the following complexity. First we have to perform  $n_2$  FFT's of order  $n_1$ , this means  $O(n_1 n_2 \log(n_1))$  operations, next we solve  $n_1$  problems with  $O(n_2)$  interpolation points. This gives us  $O(n_1 n_2 \log^2(n_1))$  operations, when working with the superfast univariate interpolation solver. Then the final step consists of constructing  $P(x, y)$  out of the  $P(x, \omega_i)$ 's, this is a simple interpolation problem which will cost us  $O(n_1 \log^2(n_1))$ . A summation of the operation cost of the different steps gives us a complexity of

$$O(n_1 n_2 \log(n_1) + n_2 n_1 \log^2(n_2) + n_1 \log^2(n_1)) \quad (8)$$

Because the size of the matrix is  $n_1 * n_2$  this algorithm is faster than an algorithm with quadratic complexity and can therefore be considered to be superfast.

## **7 Further research**

The algorithm for solving the bivariate interpolation problem can be generalized and applied in different fields. First of all we can extend this method to multivariate interpolation techniques. As can be seen in the structure of the algorithm, it can be parallellized, in a very easy and cheap way. We assume that it is also possible to solve coordinate problems as they arise in algebraic geometry, and even search in these coordinate problems, for minimal degree solutions. We are performing research on applying this algorithm to solve certain problems in signal processing. And we want to use this algorithm for the superfast solving of block Toeplitz, Toeplitz block systems.

## References

- [1] A. Bultheel and M. Van Barel. *Linear algebra, rational approximation and orthogonal polynomials*. Studies in computational mathematics. North-Holland, 1997.
- [2] C. Chaffy. (Padé)<sub>x</sub> of (Padé)<sub>y</sub> approximants of  $f(x,y)$ . In A. Cuyt, editor, *Nonlinear numerical methods and rational approximation*, volume 1 of *Mathematics and its applications*, pages 155–166. D. Reidel Publishing company, 1988.
- [3] A. Cuyt. A comparison of some multivariate Padé approximants. *SIAM Journal on Mathematical Analysis*, 14(1):195–202, January 1983.
- [4] A. Cuyt. A recursive computation scheme for multivariate rational interpolants. *SIAM Journal on Numerical Analysis*, 24(1):228–239, February 1987.
- [5] A. Cuyt. How well can the concept of Padé approximant be generalized to the multivariate case? *Journal of Computational and Applied Mathematics*, 105(1-2):25–50, 1999.
- [6] A. Cuyt and D. Lubinsky. On the convergence of multivariate Padé approximants. *Bulletin of the Belgian Mathematical Society Simon Stevin*, 1996.
- [7] C. De Boor. On the error in multivariate polynomial interpolation.
- [8] C. de Boor. Polynomial interpolation in several variables. Studies in Computer Science (J. R. Rice and R. A. DeMillo, eds.), Plenum Press, New York, 1994.
- [9] C. De Boor and A. Ron. Computational aspects of multivariate polynomial interpolation in several variables. *Math. Comp.*, 58:705–727, 1992.
- [10] M. Gasca and T. Sauer. Polynomial interpolation in several variables. *Adv. Comput. Math.*, 12(4):377–410, 2000.
- [11] Ph. Guillaume. Nested multivariate Padé approximants. *Journal of Computational and Applied Mathematics*, 82:149–158, 1997.

- [12] P. Kravanja and M. Van Barel. A fast Hankel solver based on an inversion formula for Loewner matrices. *Linear Algebra and Its Applications*, 282(1–3):275–295, September 1998.
- [13] M. Van Barel and A. Bultheel. A new approach to the rational interpolation problem. *Journal of Computational and Applied Mathematics*, 32(1-2):281–289, 1990.
- [14] M. Van Barel and A. Bultheel. A new approach to the rational interpolation problem: the vector case. *Journal of Computational and Applied Mathematics*, 33(3):331–346, 1990.
- [15] M. Van Barel and A. Bultheel. A new formal approach to the rational interpolation problem. *Numerische Mathematik*, 62:87–122, 1992.
- [16] M. Van Barel, G. Heinig, and P. Kravanja. An algorithm based on orthogonal polynomial vectors for Toeplitz Least Squares Problems. *Lecture Notes in Computerscience*, 1988:27–34, 2001.
- [17] M. Van Barel, G. Heinig, and P. Kravanja. A stabilized superfast solver for nonsymmetric toeplitz systems. *SIAM Journal on Matrix Analysis and its Applications*, 23(2):494–510, 2001.
- [18] M. Van Barel and P. Kravanja. A stabilized superfast solver for indefinite Hankel systems. *Linear Algebra and Its Applications*, 284(1–3):335–355, 1998.
- [19] R. Vandebril, M. Van Barel, O. Ruatta, and B. Mourrain. A new algorithm for multivariate interpolation problems. unpublished, 2002.