Inverses of Multivariable Polynomial Matrices by Discrete Fourier Transforms

S. Vologiannidis and N. Karampetakis
(svol, karampet)@math.auth.gr
Department of Mathematics,
Aristotle University of Thessaloniki,
Greece

Abstract. The problem of the fast computation of the Moore-Penrose and Drazin inverse of a multivariable polynomial matrix is addressed. The algorithms proposed, use evaluation-interpolation techniques and the Fast Fourier transform. They proved to be faster than other known algorithms. The efficiency of the algorithms is illustrated via random generated examples.

Keywords: generalized inverse, drazin inverse, discrete fourier transform, multivariable polynomial matrix

AMS codes: Primary 15A09, 65T50; Secondary 93B40, 93B50, 65F30, 65F05

1. Introduction

The Moore-Penrose inverse has originally been defined by Penrose [21], while later Decell [4] proposed a Leverrier-Faddeev algorithm for its computation. An extension of this algorithm to the one and two-variable polynomial matrices has been proposed by [12], [16], [17], [18]. A Leverrier-Faddeev algorithm has also been proposed by Grevile [9] for the computation of the Drazin inverse of square constant matrices with extensions to the one-variable polynomial matrices by [15], [23].

The Leverrier algorithms have the advantage that are easily implemented in symbolic programming languages like Mathematica, Maple etc. However, their main disadvantage, is their complexity. In order to overcome these difficulties we may use other techniques such as interpolation methods. Schuster and Hippe [22] for example, use interpolation techniques in order to find the inverse of a polynomial matrix. The speed of interpolation algorithms can be increased by using Discrete Fourier Transforms (DFT) techniques or better Fast Fourier Transforms (FFT). Some of the advantages of the DFT based algorithms are that there are very efficient algorithms available both in software and hardware and that they are greatly benefitted by the existence of a parallel environment (through symmetric multiprocessing or other techniques).

During the past two decades there has been extensive use of DFT - based algorithms, due to their computational speed and accuracy. Some remarkable examples, but not the only ones of the use of DFT in linear algebra problems, are the calculation of the determinantal polynomial by [20], the computation of the transfer function of generalized n-dimensional systems by [13] and the solutions of polynomial matrix Diophantine equations by [11].

Interest in these two specific inverses stems from their application in inverse systems, solution of AutoRegressive Moving Average representations [10], solution of Diophantine equations which gives rise to numerous applications to the field of control system synthesis (see for example [16] and its references) and in the study of multidimensional filters which find numerous applications in image processing, electrical networks with variable elements etc. Note that in case of square and nonsingular matrices, both inverses coincide with the known inverse of the matrix. Therefore fast computation of these special inverses is also useful to applications where the usual inverse of a matrix is required such as the computation of the transfer function of a matrix ([1], [13]).

The main purpose of this work is to present a DFT-algorithm for the evaluation of the Moore-Penrose and the Drazin inverse of a multivariable polynomial matrix. More specifically in section 2 we attempt a brief review of the notions of multidimensional DFT and interpolation, while later in section 3 and 4 we propose two new DFT algorithms for the evaluation of the Moore-Penrose and Drazin inverse respectively of a polynomial matrix in n indeterminates. In the last section the efficiency of the algorithms is illustrated via random generated examples.

2. Preliminaries

Let $\mathbb{R}$ be the set of real numbers, $\mathbb{R}^{p\times m}$ be the set of $p\times m$ real matrices, $\mathbb{R}[z_1, z_2, ..., z_n]$ (respectively. $\mathbb{R}(z_1, z_2, ..., z_n)$) denotes the polynomials (resp. rational expressions) with real coefficients in the n indeterminates $z_1, z_2, ..., z_n$. The $p \times m$ matrices with elements in $\mathbb{R}[z_1, z_2, ..., z_n]$ (resp. $\mathbb{R}(z_1, z_2, ..., z_n)$) are denoted by $\mathbb{R}[z_1, z_2, ..., z_n]^{p\times m}$ (resp. $\mathbb{R}(z_1, z_2, ..., z_n)^{p\times m}$). By I_p we denote the identity matrix of dimensions p, and by $0_{p,m}$ the $p\times m$ zero matrix. Consider the polynomial matrix with real coefficients in the n indeterminates $z_1, z_2, ..., z_n$ (called nD polynomial matrix)

$$A(z_1,\ldots,z_n)=\sum_{k_1=0}^{M_1}\sum_{k_2=0}^{M_2}\cdots\sum_{k_n=0}^{M_n}(A_{k_1\ldots k_n})\times\left(z_1^{k_1}\ldots z_n^{k_n}\right)\in\mathbb{R}\left[z_1,\ldots,z_n\right]^{p\times m} \tag{1}$$

with $A_{k_1 \ldots k_n} \in \mathbb{R}^{p \times m}$. For brevity and when the variables are obvious, we will use the notation $\bar{z}$ instead of $(z_1, z_2, ..., z_n)$. By $A(z_1, z_2, ..., z_n)^D$ (resp. $A(z_1, z_2, ..., z_n)^+$) we denote the Drazin (resp. Moore-Penrose) inverse of $A(z_1, z_2, ..., z_n)$. For the sake of completeness, the next subsections will provide a brief description of interpolation and DFT.

2.1. nD POLYNOMIAL INTERPOLATION

Evaluation-interpolation techniques proved to be a valuable tool for several algebraic computations especially when handling polynomials and polynomial matrices. Consider an nD polynomial matrix $A(\bar{z})$ as in (1). Assume that we need to compute the value of $f(A)$ where f in our case is a polynomial function. The evaluation-interpolation method of computing $f(A(\bar{z}))$ is based on the following steps.

- **Step 1** Evaluation of the polynomial (matrix) at a set of $R = \prod_{i=1}^{n} (\deg_{z_i} (f(A(\bar{z}))) + 1)$ suitably chosen points $\bar{z}_i, i = 1, 2, ..., R$. This step results in a set of constant matrices $A(\bar{z}_i), i = 1, 2, ..., R$.
- **Step 2** Application of the function f on the set of constant matrices $A(\bar{z}_i), i = 1, 2, ..., R$ derived from step 1.
- **Step 3** Computation of the coefficients of the polynomial matrix $f(A(\bar{z}))$ by using one of the known interpolation methods such as : a) the direct approach using Vandermonde's matrix, b) Newton's interpolation, c) Lagrange's interpolation.

The efficiency of the above procedure in terms of speed and storage can be vastly improved by choosing appropriate algorithms in each step.

EXAMPLE 1. *Consider the one variable polynomial matrix*

$$A(z) = \begin{bmatrix} z & 1 \\ 1 & z \end{bmatrix}$$

We are interested to compute the value of $f(A(z))$ *where*

$$f(x) = x^2$$

Step 1. *Since the degree of* $f(A(z))$ *is at most equal to* 2*, we have that* $R = (2 + 1) = 3$*. Therefore we evaluate the polynomial matrix* $A(z)$ *at a set of* 3 *points ie* $z_0 = 0, z_1 = 1, z_2 = 2$*.*

$$A(z_0) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}; A(z_1) = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}; A(z_2) = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

Step 2. *Applying the function f on the above set of constant matrices we get*

$$f\left(A(z_0)\right) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; f\left(A(z_1)\right) = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}; f\left(A(z_2)\right) = \begin{bmatrix} 5 & 4 \\ 4 & 5 \end{bmatrix}$$

Step 3. *We compute the coefficients of the polynomial matrix $f(A(z_1))$ by using the Lagrange interpolation method.*

$$f(A(z)) = \left\{ \begin{bmatrix} L_0(z) & L_1(z) & L_2(z) \end{bmatrix} \otimes I_2 \right\} \begin{bmatrix} f\left(A(z_0)\right) \\ f\left(A(z_1)\right) \\ f\left(A(z_2)\right) \end{bmatrix}$$

where

$$L_i(z) = \frac{\prod\limits_{\substack{k=0 \\ k \neq i}}^{2} (z - z_k)}{\prod\limits_{\substack{k=0 \\ k \neq i}} (z_i - z_k)}, i = 0, 1, 2$$

In our case

$$L_0(z) = \frac{1}{2}(z-1)(z-2); L_1(z) = -z(z-2); L_2(z) = \frac{1}{2}z(z-1)$$

and

$$f(A(z)) = \begin{bmatrix} z^2+1 & 2z \\ 2z & z^2+1 \end{bmatrix} = A(z)^2$$

2.2. nD DFT

Multidimensional Fourier transforms arise very frequently in many scientific fields such as image processing, statistics etc. Let us now present the strict definition of a DFT pair. Consider the finite sequence $X(k_1, \ldots, k_n)$ and $\tilde{X}(r_1, \ldots, r_n)$, $k_i, r_i = 0, 1, ..., M_i$. In order for the sequence $X(k_1, \ldots, k_n)$ and $\tilde{X}(r_1, \ldots, r_n)$ to constitute a DFT pair the following relations should hold [6] :

$$\tilde{X}(r_1, \ldots, r_n) = \sum_{k_1=0}^{M_1} \sum_{k_2=0}^{M_2} \cdots \sum_{k_n=0}^{M_n} X(k_1, \ldots, k_n) W_1^{-k_1 r_1} \cdots W_1^{-k_n r_n} \tag{2}$$

$$X(k_1, \ldots, k_n) = \frac{1}{R} \sum_{r_1=0}^{M_1} \sum_{r_2=0}^{M_2} \cdots \sum_{r_n=0}^{M_n} \tilde{X}(r_1, \ldots, r_n) W_1^{k_1 r_1} \cdots W_1^{k_n r_n} \tag{3}$$

where

$$W_i = e^{\frac{2\pi j}{M_i+1}} \ \forall i = 1, 2, 3, ..., n \tag{4}$$

$$R = \prod_{i=1}^{n} (M_i + 1) \tag{5}$$

and $X, \tilde{X}$ are discrete argument matrix-valued functions, with dimensions $p \times m$. The complex numbers W_i defined in (4) are otherwise called Fourier points. Relation (2) is the forward Fourier transform of $X(k_1, \ldots, k_n)$ while (3) the inverse Fourier transform of $\tilde{X}(r_1, \ldots, r_n)$.

2.3. Computational complexity and FFT

The FFT appeared in the literature in [3], although Gauss had actually described FFT's critical steps as early as 1805. The great advantage of FFT methods is their reduced complexity. The complexity of $1D$ DFT on a matrix $M \in \mathbb{R}^{1\times R}$ is an $\mathcal{O}(R^2)$, while the FFT has a complexity of $\mathcal{O}(RlogR)$. Similarly, the complexity of the DFT of a multidimensional matrix $M \in \mathbb{R}^{m_1\times\cdots\times m_n}$ is $\mathcal{O}\left(\prod_{i=1}^{n} m_i^2\right)$ which using FFT reduces to $\mathcal{O}\left(\left(\prod_{i=1}^{n} m_i\right)\left(\sum_{i=1}^{n} \log m_i\right)\right)$. The inverse DFT is of the same complexity as the forward one. A very fast free implementation of FFT can be found in [7]. An analysis of the floating point errors in the determination of the Fourier coefficients can be found in [2], [14]. Also due to the great popularity of the FFT, there are several implementations that take into account the particular features of the host machine such as cache, storage, pipelining etc.

2.4. Polynomial interpolation and DFT

Let us now focus on the main problem, which is the connection of interpolation techniques and DFT. Let us now investigate the efficiency of steps 1 to 3 of the evaluation-interpolation algorithm presented in section 2.1. In step 2 a suitable algorithm computing $f(A(\bar{z}_i))$ where $A(\bar{z}_i)$ are constant matrices, must be chosen. The critical points of the algorithm are steps 1 and 3. By using as evaluation points in step 1 Fourier points, the evaluation of the polynomial matrix is equivalent to the DFT of the multidimensional matrix of its coefficients. If we need to evaluate the polynomial matrix in more than R points, we should pad the coefficient matrix by zeros and afterwords apply DFT. Then step 3 becomes an inverse DFT problem. By using DFT techniques of better FFT techniques, the speed of the algorithm improves dramatically.

The importance of FFT in the computational efficiency of algebraic algorithms is stressed in [2], [19].

3. Moore-Penrose Inverse of a Multivariable Polynomial Matrix

The Moore-Penrose generalized inverse of a constant matrix was defined by Penrose in [21].

DEFINITION 1. *[21] For every matrix $A \in \mathbb{R}^{p\times m}$, a unique matrix $A^+ \in \mathbb{R}^{m\times p}$, which is called (Moore-Penrose) generalized inverse, exists satisfying*

(i) $AA^+A = A$

(ii) $A^+AA^+ = A^+$

(iii) $(AA^+)^T = AA^+$

(iv) $(A^+A)^T = A^+A$

where A^T denotes the transpose of A. In the special case that the matrix A is square nonsingular matrix, the Moore-Penrose generalized inverse of A is simply its inverse i.e. $A^+ = A^{-1}$.

Consider a possibly non square nD polynomial matrix with real coefficients as in (1). In an analogous way we define the Moore-Penrose generalized inverse $A(z_1,\ldots,z_n)^+ \in \mathbb{R}\,(z_1,\ldots,z_n)^{m\times p}$ of the polynomial matrix $A(z_1,\ldots,z_n) \in \mathbb{R}\,[z_1,\ldots,z_n]^{p\times m}$ defined in (1) as the matrix which satisfies the properties (i)-(iv) of Definition (1). Following the steps of [17] we have

THEOREM 1. *Let $A(\bar{z}) = A(z_1,\ldots,z_n) \in \mathbb{R}\,[z_1,\ldots,z_n]^{p\times m}$ as in (1) and*

$$\begin{aligned} a(s,z_1,\ldots,z_n) &= \det\left[sI_p - A(\bar{z})A(\bar{z})^T\right] \\ &= (a_0(\bar{z})s^p + \cdots + a_{p-1}(\bar{z})s + a_p(\bar{z})) \end{aligned} \tag{6}$$

$a_0(\bar{z}) = 1$, be the characteristic polynomial of $A(\bar{z}) \times A(\bar{z})^T$. Let k such that it satisfies $a_p(\bar{z}) \equiv 0, ..., a_{k+1}(\bar{z}) \equiv 0$ while $a_k(\bar{z}) \neq 0$, and define $\Lambda := \{(\bar{z}) \in \mathbb{C}^n : a_k(\bar{z}) = 0\}$. Then the Moore-Penrose generalized inverse $A(\bar{z})^+$ of $A(\bar{z})$ for $\bar{z} \in \mathbb{C}^n - \Lambda$ is given by

$$A(\bar{z})^+ = -\frac{1}{a_k(\bar{z})}A(\bar{z})^T B_{k-1}(\bar{z}) \tag{7}$$

$$B_{k-1}(\bar{z}) = a_0(\bar{z})\left[A(\bar{z})A(\bar{z})^T\right]^{k-1} + \cdots + a_{k-1}(\bar{z})I_p \tag{8}$$

If $k = 0$ is the largest integer such that $a_k(\bar{z}) \neq 0$, then $A(\bar{z})^+ = 0$. For those $\bar{z} \in \Lambda$ we can use the same algorithm again.

Proof. The theorem is can be proved following the same steps as in the proof of the corresponding theorem about constant matrices in [4]. ■

REMARK 1. *The algorithm described in Theorem 1 is efficient using symbolic programming languages. Its main advantages are*
1) It consists of simple recursions.
2) No matrix inversion is required.
3)In the case that $p > m$ we can compute the transpose $B(\bar{z}) = A(\bar{z})^T$ and then compute $A(\bar{z})^+ = [B(\bar{z})^+]^T$ in order to reduce the complexity of the algorithm.

In the following we will propose a new algorithm for the calculation of the Moore-Penrose generalized inverse using interpolation and discrete Fourier transforms. It is easily seen from (6), that the greatest powers of the $(n+1)$ variables in $a(s, z_1, \ldots, z_n)$ are

$$\begin{array}{c} \deg_s \left(a(s, z_1, \ldots, z_n)\right) = p := b_0 \\ \deg_{z_1} \left(a(s, z_1, \ldots, z_n)\right) \leq 2pM_1 := b_1 \\ \vdots \\ \deg_{z_n} \left(a(s, z_1, \ldots, z_n)\right) \leq 2pM_n := b_n \end{array} \tag{9}$$

Thus, the polynomial $a(s, z_1, \ldots, z_n)$ can be written as

$$a(s, z_1, \ldots, z_n) = \sum_{k_0=0}^{b_0} \sum_{k_1=0}^{b_1} \ldots \sum_{k_n=0}^{b_n} \left(a_{k_0 k_1 \ldots k_n}\right) \left(s^{k_0} z_1^{k_1} \ldots z_n^{k_n}\right) \tag{10}$$

and can be numerically computed via interpolation using the following R_1 points

$$\begin{array}{c} u_i(r_j) = W_i^{-r_j}; i = 0, \ldots, n \text{ and } r_j = 0, 1, ..., b_i \\ W_i = e^{\frac{2\pi j}{b_i+1}} \end{array} \tag{11}$$

where

$$R_1 = \prod_{i=0}^{n} (b_i + 1) \tag{12}$$

In order, to evaluate the coefficients $a_{k_0 k_1 \ldots k_n}$ define

$$\tilde{a}_{r_0 r_1 \ldots r_n} = \det \left[u_0(r_0) I_p - A(u_1(r_1), \ldots, u_n(r_n)) \left[A(u_1(r_1), \ldots, u_n(r_n))\right]^T\right] \tag{13}$$

From (10), (11), (13) we get

$$\tilde{a}_{r_0 r_1 \ldots r_n} = \sum_{l_0=0}^{b_0} \sum_{l_1=0}^{b_1} \ldots \sum_{l_n=0}^{b_n} \left(a_{l_0 l_1 \ldots l_n}\right) \left(W_0^{-r_0 l_0} \ldots W_n^{-r_n l_n}\right) \tag{14}$$

Notice that $[a_{l_0 l_1 \ldots l_n}]$ and $[\tilde{a}_{r_0 r_1 \ldots r_n}]$ form a DFT pair and thus using (3) we derive the coefficients of (10) i.e.

$$a_{l_0 l_1 \ldots l_n} = \frac{1}{R_1} \sum_{r_0=0}^{b_0} \sum_{r_1=0}^{b_1} \cdots \sum_{r_n=0}^{b_n} \tilde{a}_{r_0 r_1 \ldots r_n} W_0^{r_0 l_0} \ldots W_n^{r_n l_n}$$

where $l_i = 0, \ldots, b_i$. After evaluating k as in theorem 1, $C(\bar{z})$ has to be computed. The greatest powers of z_i in

$$\begin{gathered} C(\bar{z}) = A(\bar{z})^T B_{k-1}(\bar{z}) = \\ = A(\bar{z})^T \left(a_0(\bar{z}) \left[A(\bar{z}) A(\bar{z})^T \right]^{k-1} + \cdots + a_{k-1}(\bar{z}) I_p \right) \end{gathered} \tag{15}$$

is

$$n_i = \max \{2(k-1)M_i + M_i, k = 1, \ldots, p\} = (2p-1)M_i \tag{16}$$

for $i = 1, ..., n$. Then $C(\bar{z})$ can be written as

$$C(\bar{z}) = \sum_{l_1=0}^{n_1} \ldots \sum_{l_n=0}^{n_n} C_{l_0 \cdots l_n} \left(z_1^{l_1} \cdots z_n^{l_n} \right) \tag{17}$$

We compute $C(\bar{z})$ via interpolation using the following R_2 points

$$\begin{gathered} u_i(r_j) = W_i^{-r_j}; i = 1, \ldots, n \text{ and } r_j = 0, 1, ..., n_i \\ W_i = e^{\frac{2\pi j}{n_i+1}} \end{gathered} \tag{18}$$

where

$$R_2 = \prod_{i=1}^{n} \{(2p-1)M_i + 1\} \tag{19}$$

To evaluate the coefficients $C_{l_0 \cdots l_n}$ define

$$\tilde{C}_{r_1 \cdots r_n} = C(u_1(r_1), \ldots u_n(r_n)) \tag{20}$$

Using (18) and (19), equation number (20) becomes

$$\tilde{C}_{r_1 \cdots r_n} = \sum_{l_1=0}^{n_1} \ldots \sum_{l_n=0}^{n_n} C_{l_0 \cdots l_n} W_1^{-r_1 l_1} \cdots W_n^{-r_n l_n}$$

which through (3)

$$C_{l_0 \cdots l_n} = \frac{1}{R_2} \sum_{l_1=0}^{n_1} \cdots \sum_{l_n=0}^{n_n} \tilde{C}_{r_1 \cdots r_n} W_1^{r_1 l_1} \cdots W_n^{r_n l_n}$$

where $l_i = 0, \ldots, n_i$. Finally the Moore-Penrose generalized inverse is computed via (7). Having in mind the above theoretical considerations, we will continue by describing the algorithm as a pure guide for computation. The multivariable polynomial matrix $A(\bar{z})$ in (1) is assumed to be internally represented by a multidimensional matrix $\mathcal{A} \in \mathbb{R}^{(M_1+1)\times\cdots\times(M_n+1)\times p\times m}$ whose elements $\mathcal{A}(k_1, \ldots, k_n, *) \in \mathbb{R}^{p\times m}$ are the coefficient matrices of $z_1^{k_1} \ldots z_n^{k_n}$ and $0 \leq k_i \leq M_i$. In the following we will refer to such matrices using the terminology "$\mathcal{A} \in \mathbb{R}^{(M_1+1)\times\cdots\times(M_n+1)}$ with elements $\mathcal{A}(k_1, \ldots, k_n) \in \mathbb{R}^{p\times m}$". The above multidimensional matrix is given as the input to the algorithm. Note that for the sake of clarity, in the algorithm we use different symbols for every matrix computed.

Algorithm 1. (Evaluation of the Moore-Penrose inverse)

1. **Evaluate (10)**

   a) Calculate the number of interpolation points b_i by (9)

   b) Define $\mathcal{B} \in \mathbb{R}^{2\times(M_1+1)\times\cdots\times(M_n+1)}$ by

$$\mathcal{B}(i, k_1, \ldots, k_n) = \begin{cases} A(k_1, \ldots, k_n) & i = 0 \\ 0_{p\times m} & i = 1 \end{cases}$$

   where

$$\mathcal{B}(i, k_1, \ldots, k_n) \in \mathbb{R}^{p\times m}$$

   The above procedure adds another dimension in $\mathcal{A}$ corresponding to the new variable s. Pad $\mathcal{B}$ by zeros until it reaches dimensions $(b_0 + 1) \times (b_1 + 1) \times \cdots \times (b_n + 1)$.

   c) Compute the multidimensional FFT transform of $\mathcal{B}$ and store it in $\mathcal{B}_F$.

   d) Define $\mathcal{C}_F \in \mathbb{C}^{(b_0+1)\times(b_1+1)\times\cdots\times(b_n+1)}$ by

$$\mathcal{C}_F(i, k_1, \ldots, k_n) = \mathcal{B}_F(i, k_1, \ldots, k_n)\left(\mathcal{B}_F(i, k_1, \ldots, k_n)\right)^T \in \mathbb{C}^{p\times p}$$

   e) Define $\mathcal{S} \in \mathbb{R}^{(b_0+1)\times(b_1+1)\times\cdots\times(b_n+1)}$ as

$$\mathcal{S}(i, k_1, \ldots, k_n) = \begin{cases} I_p & (i, k_1, ..., k_n) = (1, 0, ..., 0) \\ 0_{p\times p} & (i, k_1, ..., k_n) \neq (1, 0, ..., 0) \end{cases}$$

   where

$$\mathcal{S}(i, k_1, \ldots, k_n) \in \mathbb{R}^{p\times p}$$

   f) Compute the multidimensional FFT transform of $\mathcal{S}$ and store it in $\mathcal{S}_F$.

g) Compute $\mathcal{D}_F = \mathcal{C}_F - \mathcal{S}_F$.

h) Calculate the determinants of $\mathcal{D}_F\,(i, k_1, \ldots, k_n)$, which result in a multidimensional matrix $\mathcal{E}_F \in \mathbb{C}^{(b_0+1)\times(b_1+1)\times\cdots\times(b_n+1)}$ with elements in $\mathbb{C}$.

i) Use the IFFT on $\mathcal{E}_F$ resulting in the matrix $\mathcal{E}$ which corresponds to the coefficients of (10).

2. **Evaluate** $a_k(\bar{z})$

a) Since $\mathcal{E}(i, k_1, \ldots, k_n)$ correspond to the coefficients of $s^i z_1^{k_1} ... z_n^{k_n}$, find (using a for loop) k st. $\mathcal{E}(k, *) \neq 0$ and $\mathcal{E}(i, *) = 0$, $0 \leq i < k$.

3. **Evaluate** $C(\bar{z}) = A(\bar{z})^T B_{k-1}(\bar{z})$

a) Calculate the number of interpolation points n_i by (16).

b) Define $\mathcal{G}$ by padding $\mathcal{A}$ by zeros until it reaches dimensions $(n_1 + 1) \times \cdots \times (n_n + 1)$.

c) Compute the multidimensional FFT transform of $\mathcal{G}$ and store it in $\mathcal{G}_F$.

d) Define $\mathcal{H}$ by padding $\mathcal{E}$ with zeros until it reaches dimensions $(n_1 + 1) \times \cdots \times (n_n + 1)$.

e) Compute the multidimensional FFT transform of $\mathcal{H}$ and store it in $\mathcal{H}_F$.

f) Compute (15) for each constant matrix in $\mathcal{G}_F$. The result is $\mathcal{J}_F \in \mathbb{C}^{(n_1+1)\times\cdots\times(n_n+1)}$. The elements of $\mathcal{J}_F$ are in $\mathbb{C}^{m\times p}$. This step can be accomplished using the Horner's rule, which results in faster and more stable calculations.

g) Use the IFFT on $\mathcal{J}_F$ to compute the coefficients of (15) and store them in $\mathcal{J}$.

4. **Return numerator and denominator of the Moore-Penrose inverse**

a) Return $\mathcal{J}_F$ and $\mathcal{E}(k, *)$.

In the following the complexity of each step of the algorithm that is considered CPU intensive is presented. The symbols R_1 and R_2 are defined respectively in (12) and (19). We introduce the following notation

$$L_1 = \sum_{i=0}^{n} \log(b_i + 1), \;\; L_2 = \sum_{i=1}^{n} \log(n_i + 1)$$

Step No	Complexity	Step No	Complexity
1c	$\mathcal{O}(pmR_1L_1)$	1i	$\mathcal{O}(R_1L_1)$
1d	$\mathcal{O}(p^2(2m-1)R_1)$	3c	$\mathcal{O}(pmR_2L_2)$
1f	$\mathcal{O}(p^2R_1L_1)$	3e	$\mathcal{O}(R_2L_2)$
1g	$\mathcal{O}(p^2R_1)$	3f	$\mathcal{O}(kp^3R_2)$
1h	$\mathcal{O}(p^3R_1)$	3g	$\mathcal{O}(pmR_2L_2)$

Denote by $R = \max\{R_1, R_2\}$ and by $L = \max\{L_1, L_2\}$. Having in mind that $p < m$, since otherwise we would use the transpose of the matrix $A(\bar{z})$, (see remark 1) and that $k = p$ (worst case scenario), the total complexity of the algorithm is of the order $\mathcal{O}(mp^3RL)$.

4. Drazin Inverse of a Multivariable Polynomial Matrix

The Drazin inverse of a constant matrix was defined by Drazin in [5].

DEFINITION 2. *For every matrix $A \in \mathbb{R}^{m\times m}$, there exists a unique matrix $A^D \in \mathbb{R}^{m\times m}$, which is called Drazin inverse, satisfying*

(i) $A^DA^{k+1} = A^k$ *for* $k = ind(A) = \min(k \in N : rank\left(A^k\right) = rank\left(A^{k+1}\right))$

(ii) $A^DAA^D = A^D$

(iii) $AA^D = A^DA$

In the special case that the matrix A is square and nonsingular matrix, the Drazin inverse of A is simply its inverse i.e. $A^D = A^{-1}$.

In an analogous way we define the Drazin inverse of polynomial matrix $A(z_1, \ldots, z_n) \in \mathbb{R}[z_1, \ldots, z_n]^{m\times m}$ defined in (1) as the matrix which satisfies the properties of Definition (2). The following theorem proposes a new algorithm for the computation of the Drazin inverse of an nD polynomial matrix, which generalizes the results in [23].

THEOREM 2. *Consider a nonregular nD polynomial matrix $A(\bar{z})$. Assume that*

$$\begin{aligned} a(s, z_1, \ldots, z_n) &= \det[sI_m - A(\bar{z})] \\ &= (a_0(\bar{z})s^m + \cdots + a_{m-1}(\bar{z})s + a_m(\bar{z})) \end{aligned} \tag{21}$$

where

$$a_0(\bar{z}) \equiv 1, z \in \mathbb{C}$$

is the characteristic polynomial of $A(\bar{z})$*. Also, consider the following sequence of* $m \times m$ *polynomial matrices*

$$\begin{array}{c} B_j(\bar{z}) = a_0(\bar{z})A(\bar{z})^j + \cdots a_{j-1}(\bar{z})A(\bar{z}) + a_j(\bar{z})I_m, \\ a_0(\bar{z}) = 1, j = 0, \ldots, m \end{array} \tag{22}$$

Let

$$a_m(\bar{z}) \equiv 0, \ldots, a_{t+1}(\bar{z}) \equiv 0, a_t(\bar{z}) \neq 0. \tag{23}$$

Define the following set:

$$\Lambda = \{\bar{z}_i \in \mathbb{C}^n : a_t(\bar{z}_i) = 0\}$$

Also, assume that

$$B_m(\bar{z}),...,B_r(\bar{z}) = 0, B_{r-1}(\bar{z}) \neq 0 \tag{24}$$

and $k = r - t$*. In the case* $\bar{z} \in \mathbb{C}^n - \Lambda$ *and* $k > 0$*, the Drazin inverse of* $A(\bar{z})$ *is given by*

$$\begin{array}{c} A(\bar{z})^D = \dfrac{A(\bar{z})^k B_{t-1}(\bar{z})^{k+1}}{a_t(\bar{z})^{k+1}} \\ B_{t-1}(\bar{z}) = a_0(\bar{z})A(\bar{z})^{t-1} + \cdots + a_{t-2}(\bar{z})A(s) + a_{t-1}(\bar{z})I_m \end{array} \tag{25}$$

In the case $\bar{z} \in \mathbb{C}^n - \Lambda$ *and* $k = 0$*, we get* $A(\bar{z})^D = O$*.*
For $\bar{z}_i \in \Lambda$ *we can use the same algorithm again.*

Proof. The proof uses the same logic as the one in [23]. For the sake of brevity the interested reader is advised to study [23]. ■

A useful test to check whether a multivariable polynomial matrix is zero, is described by the following lemma.

LEMMA 3. *A polynomial matrix*

$$B(z_1, \ldots, z_n) \in \mathbb{R}[z_1, \ldots, z_n]^{m \times m} \tag{26}$$

of degrees q_i *in respect with variables* $z_i, i = 1, ..., n$*, is the zero polynomial matrix iff its values at* $R = \prod_{i=0}^{n}(q_i + 1)$ *distinct points are the zero matrix. Note that in practice it is necessary to check whether the values at those distinct points are all smaller in absolute value than* ε*, where* ε *should be close to the machine numerical precision.*

In the following we suggest a computationally attractive algorithm for the calculation of Drazin inverses based on interpolation techniques

and DFT. It is easily seen from (21), that the greatest powers of the $(n+1)$ variables in $a(s, z_1, \ldots, z_n)$ are

$$\begin{array}{c} \deg_s\left(a(s, z_1, \ldots, z_n)\right) = m := b_0 \\ \deg_{z_1}\left(a(s, z_1, \ldots, z_n)\right) \leq mM_1 := b_1 \\ \vdots \\ \deg_{z_n}\left(a(s, z_1, \ldots, z_n)\right) \leq mM_n := b_n \end{array} \tag{27}$$

So the polynomial $a(s, z_1, \ldots, z_n)$ can be written as

$$a(s, z_1, \ldots, z_n) = \sum_{k_0=0}^{b_0} \sum_{k_1=0}^{b_1} \cdots \sum_{k_n=0}^{b_n} \left(a_{k_0 k_1 \ldots k_n}\right) \left(s^{k_0} z_1^{k_1} \ldots z_n^{k_n}\right) \tag{28}$$

and can be numerically computed via interpolation using the following R_1 points

$$\begin{array}{c} u_i(r_j) = W_i^{-r_j}; i = 0, \ldots, n \text{ and } r_j = 0, 1, ..., b_i \\ W_i = e^{\frac{2\pi j}{b_i+1}} \end{array} \tag{29}$$

where

$$R_1 = \prod_{i=0}^{n} (b_i + 1) \tag{30}$$

To evaluate the coefficients $a_{k_0 k_1 \ldots k_n}$ define

$$\tilde{a}_{r_0 r_1 \ldots r_n} = \det\left[u_0(r_0) I_p - A(u_1(r_1), \ldots, u_n(r_n))\right] \tag{31}$$

From (28), (29), (31) we get

$$\tilde{a}_{r_0 r_1 \ldots r_n} = \sum_{l_0=0}^{b_0} \sum_{l_1=0}^{b_1} \cdots \sum_{l_n=0}^{b_n} \left(a_{l_0 l_1 \ldots l_n}\right) \left(W_0^{-r_0 l_0} \ldots W_n^{-r_n l_n}\right) \tag{32}$$

Notice that $[a_{l_0 l_1 \ldots l_n}]$ and $[\tilde{a}_{r_0 r_1 \ldots r_n}]$ form a DFT pair and thus using the above equation and (3) the coefficients of (28) are

$$a_{l_0 l_1 \ldots l_n} = \frac{1}{R_1} \sum_{r_0=0}^{b_0} \sum_{r_1=0}^{b_1} \cdots \sum_{r_n=0}^{b_n} \tilde{a}_{r_0 r_1 \ldots r_n} W_0^{r_0 l_0} \ldots W_n^{r_n l_n} \tag{33}$$

where $l_i = 0, \ldots, b_i$. Following the lines of theorem 2, the evaluation of $a_t(\bar{z})$ as in (23) is straightforward. Consider the polynomial matrix $B_i(\bar{z})$ in (22). We need to evaluate r such that (24) holds. To check whether $B_i(\bar{z})$ is the zero matrix using lemma (3),

$$\hat{R}_i = \prod_{j=0}^{n} \underbrace{(iM_j + 1)}_{m_{ij}}, i = r-1, \ldots, m \tag{34}$$

Fourier interpolation points are needed. After a short loop we determine the value of r. Afterwards the computation of the numerator of (25)

$$C(\bar{z}) = A(\bar{z})^k B_{t-1}(\bar{z})^{k+1} \tag{35}$$

is in order. The greatest power of z_i in (35) is

$$n_i = (t-1)(k+1)M_i \tag{36}$$

Using (36), $C(\bar{z})$ can be written as

$$C(\bar{z}) = \sum_{l_1=0}^{n_1} \cdots \sum_{l_n=0}^{n_n} C_{l_0 \cdots l_n} \left(z_1^{l_1} \cdots z_n^{l_n} \right)$$

We compute $C(\bar{z})$ via interpolation using the following R_2 points

$$\begin{gathered} u_i(r_j) = W_i^{-r_j}; i = 1, \ldots, n \text{ and } r_j = 0, 1, ..., n_i \\ W_i = e^{\frac{2\pi j}{n_i+1}} \end{gathered} \tag{37}$$

where

$$R_2 = \prod_{i=1}^{n} \{(n_i + 1\} \tag{38}$$

To evaluate the coefficients $C_{l_0 \cdots l_n}$ define

$$\tilde{C}_{r_1 \cdots r_n} = C(u_1(r_1), \ldots u_n(r_n)) \tag{39}$$

Using (36), (37), (39) becomes

$$\tilde{C}_{r_1 \cdots r_n} = \sum_{l_1=0}^{n_1} \cdots \sum_{l_n=0}^{n_n} C_{l_0 \cdots l_n} W_1^{-r_1 l_1} \cdots W_n^{-r_n l_n}$$

which through (3) becomes

$$C_{l_0 \cdots l_n} = \frac{1}{R_2} \sum_{l_1=0}^{n_1} \cdots \sum_{l_n=0}^{n_n} \tilde{C}_{r_1 \cdots r_n} W_1^{r_1 l_1} \cdots W_n^{r_n l_n}$$

where $l_i = 0, \ldots, n_i$. Similarly the greatest power of z_i appearing in $c(\bar{z}) = a_t(\bar{z})^{k+1}$ is

$$d_i = tM_i(k+1) \tag{40}$$

so $c(\bar{z})$ can be written as

$$c(\bar{z}) = \sum_{k_1=0}^{d_1} \cdots \sum_{k_n=0}^{d_n} \left(c_{k_1 \ldots k_n} \right) \left(z_1^{k_1} \ldots z_n^{k_n} \right)$$

and can be numerically computed via interpolation using the following R_3 points

$$\begin{gathered} u_i(r_j) = W_i^{-r_j}; i = 1, \ldots, n \text{ and } r_j = 0, 1, ..., d_i \\ W_i = e^{\frac{2\pi j}{d_i+1}} \end{gathered} \tag{41}$$

where

$$R_3 = \prod_{i=1}^{n} (d_i + 1) \tag{42}$$

Define

$$\tilde{c}_{r_1 \ldots r_n} = \sum_{l_1=0}^{d_1} \cdots \sum_{l_n=0}^{d_n} (c_{k_1 \ldots k_n}) \left(W_1^{-r_1 l_1} \ldots W_n^{-r_n l_n} \right) \tag{43}$$

Using the above equation and (3) we have

$$c_{l_1 \ldots l_n} = \frac{1}{R_3} \sum_{r_1=0}^{d_1} \cdots \sum_{r_n=0}^{d_n} \tilde{c}_{r_1 \ldots r_n} W_1^{r_1 l_1} \ldots W_n^{r_n l_n}$$

where $l_i = 0, \ldots, d_i$. After the above, the Drazin inverse can be computed via (25) as

$$A(\bar{z})^D = \frac{C(\bar{z})}{c(\bar{z})}$$

The main algorithm of the calculation of the Drazin inverse follows. As in Algorithm 1, we assume that the input is a multidimensional matrix $\mathcal{A} \in \mathbb{R}^{(M_1+1)\times\cdots\times(M_n+1)}$ whose elements $A(k_1, \ldots, k_n) \in \mathbb{R}^{m\times m}$ are the coefficient matrices of $z_1^{i_1} \ldots z_n^{i_n}$ and $0 \leq k_i \leq M_i$.

Algorithm 2. (Evaluation of the Drazin inverse)

1. **Evaluate (21)**

   a) Calculate the number of interpolation points b_i using (27).

   b) Define $\mathcal{B} \in \mathbb{R}^{2\times(M_1+1)\times\cdots\times(M_n+1)}$ by

$$\mathcal{B}(i, k_1, \ldots, k_n) = \begin{cases} A(k_1, \ldots, k_n) & i = 0 \\ 0_{m\times m} & i = 1 \end{cases}$$

   where

$$\mathcal{B}(i, k_1, \ldots, k_n) \in \mathbb{R}^{m\times m}$$

   The above procedure adds another dimension in $\mathcal{A}$ corresponding to the new variable s. Pad $\mathcal{B}$ by zeros until it reaches dimensions $(b_0 + 1) \times (b_1 + 1) \times \cdots \times (b_n + 1)$.

c) Compute the multidimensional FFT transform of $\mathcal{B}$ and store it in $\mathcal{B}_F$.

d) Define $\mathcal{S} \in \mathbb{R}^{(b_0+1)\times(b_1+1)\times\cdots\times(b_n+1)}$ as

$$\mathcal{S}\left(i,k_1,\ldots,k_n\right) = \begin{cases} I_m & (i,k_1,...,k_n) = (1,0,...,0) \\ 0_{m\times m} & (i,k_1,...,k_n) \neq (1,0,...,0) \end{cases}$$

where

$$\mathcal{S}\left(i,k_1,\ldots,k_n\right) \in \mathbb{R}^{m\times m}$$

e) Compute the multidimensional FFT transform of $\mathcal{S}$ and store it in $\mathcal{S}_F$.

f) Compute $\mathcal{D}_F = \mathcal{B}_F - \mathcal{S}_F$.

g) Calculate the determinants of $\mathcal{D}_F\left(i,k_1,\ldots,k_n\right)$, which results in a multidimensional matrix $\mathcal{E}_F \in \mathbb{C}^{(b_0+1)\times(b_1+1)\times\cdots\times(b_n+1)}$ with elements in $\mathbb{C}$.

h) Use the IFFT on $\mathcal{E}_F$ resulting in the matrix $\mathcal{E}$ which corresponds to the coefficients of (21).

2. **Evaluate** $a_t(\bar{z})$

a) Since $\mathcal{E}(i,k_1,\ldots,k_n)$ correspond to the coefficients of $s^i z_1^{k_1}...z_n^{k_n}$, find (using a for loop) t st.$\mathcal{E}(t,*) \neq 0$ and $\mathcal{E}(i,*) = 0$, $0 \leq i < t$

3. **Find** $B_{r-1} \neq 0$

- $i = m$
  Do WHILE $(\mathcal{E}_i < \varepsilon)$
- Calculate the number of interpolation points m_{ij} as in (34).
- Define $\mathcal{C}_i$ by padding $\mathcal{A}$ by zeros until it reaches dimensions $(m_{i1}+1)\times\cdots\times(m_{in}+1)$.
- Compute the multidimensional FFT transform of $\mathcal{C}_i$ and store it in $\mathcal{C}_i^F$.
- Define $\mathcal{D}_i$ by padding $\mathcal{E}$ with zeros until it reaches dimensions $(n_1+1)\times\cdots\times(n_n+1)$.
- Compute the multidimensional FFT transform of $\mathcal{D}_i$ and store it in $\mathcal{D}_i^F$.
- Compute (22) and store it in $\mathcal{E}_i$.
- $i = i-1$
- END DO.
  $r = i$

4. **Evaluate (35)**

   a) Calculate the number of interpolation points n_i as in (36).

   b) Define $\mathcal{G}$ by padding $\mathcal{A}$ by zeros until it reaches dimensions $(n_1+1)\times\cdots\times(n_n+1)$.

   c) Compute the multidimensional FFT transform of $\mathcal{G}$ and store it in $\mathcal{G}_F$.

   d) Define $\mathcal{H}$ by padding $\mathcal{E}$ with zeros until it reaches dimensions $(n_1+1)\times\cdots\times(n_n+1)$.

   e) Compute the multidimensional FFT transform of $\mathcal{H}$ and store it in $\mathcal{H}_F$.

   f) Compute (35) using the constant matrices in $\mathcal{G}_F$ and $\mathcal{H}_F$. The result is $\mathcal{J}_F \in \mathbb{C}^{(n_1+1)\times\cdots\times(n_n+1)}$. The elements of $\mathcal{J}_F$ are in $\mathbb{C}^{m\times m}$.

   g) Use the IFFT on $\mathcal{J}_F$ to compute the coefficients of (15) and store it in $\mathcal{F}$.

5. **Evaluate** $a_t(\bar{z})^{k+1}$

   a) The coefficients of $a_t(\bar{z})$ correspond to the multidimensional matrix $E(t,*)$. Create $\mathcal{H}$ by padding $E(t,*)$ with zeros up to dimension $(d_1+1)\times\cdots\times(d_n+1)$ where d_i are defined in (40).

   b) Compute the multidimensional FFT transform of $\mathcal{H}$ and store it in $\mathcal{H}_F$.

   c) Compute the $(k+1)$ power of each element in $\mathcal{H}_F$, storing the results in $\mathcal{I}_F$.

   d) Use the IFFT on $\mathcal{I}_F$ to compute the coefficients of $a_t(\bar{z})^{k+1}$.

6. **Return numerator and denominator of Drazin inverse**

   a) Return $\mathcal{F}$ and $\mathcal{I}$.

In the following the complexity of each step of the algorithm that is considered CPU intensive is presented. The symbols R_1, R_2, R_3 and $\hat{R}_i$ are defined respectively in (30),(38),(42) and (34). We introduce the following notation

$$L_1 = \sum_{i=0}^{n}\log(b_i+1), L_2 = \sum_{i=1}^{n}\log(n_i+1), L_3 = \sum_{i=1}^{n}\log(d_i+1)$$

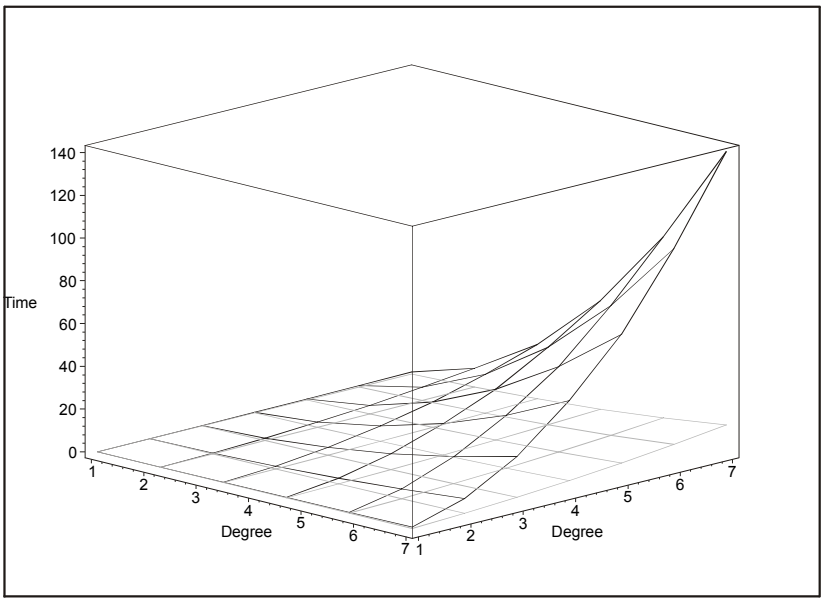
$$\hat{L}_i = \sum_{i=0}^{n}\log(m_{ij}+1), i = r-1,...,m$$

Step No	Complexity	Step No	Complexity
1c	$\mathcal{O}\left(m^2 R_1 L_1\right)$	3	$\mathcal{O}\left(\sum_{i=r-1}^{m} m^3 \hat{R}_i \hat{L}_i\right)$
1e	$\mathcal{O}\left(m^2 R_1 L_1\right)$	4c	$\mathcal{O}\left(m^2 R_2 L_2\right)$
1f	$\mathcal{O}\left(m^2 R_1\right)$	4d	$\mathcal{O}\left(R_2 L_2\right)$
1g	$\mathcal{O}\left(m^3 R_1\right)$	4f	$\mathcal{O}\left(k m^3 R_2\right)$
1h	$\mathcal{O}\left(R_1 L_1\right)$	5b,5c,5d	$\mathcal{O}\left(R_3 L_3\right)$

Denote by $R = \max\left\{R_1, R_2, R_3, \hat{R}_i\right\}$ and by $L = \max\left\{L_1, L_2, L_3, \hat{L}_i\right\}$. Assuming that the worst case (most expensive in computations) is when $k = m$, the total complexity of the algorithm is bounded by $\mathcal{O}\left(m^4 RL\right)$.

5. Numerical aspects of the algorithms

In this section some experimental results about the efficiency of the algorithm on the computation of the Moore-Penrose generalized inverse are presented. Similar results hold for the computation of the Drazin inverse. The algorithms were implemented using Mathematica 4.1. We run some benchmarks on a Pentium III 700Mhz with 128Mb of RAM using the default floating point accuracy of Mathematica and the results clearly showed the advantages of the DFT based approach. In section 3, we have shown that the complexity of the algorithm depends on the dimensions and the degree (corresponding to each variable) of the matrix. Since it is not viable to present in the same graph the CPU time as a function of three or more variables, we chose to create random $2D$ polynomial matrices with constant dimensions 3×4 and degrees up to 7, and plot the CPU time as a function of the degrees. The grey wireframe corresponds to the FFT method, while the black one to the symbolic one. Note that the polynomials that appeared as elements in the matrices had almost all their coefficients nonzero.



Comparison between the CPU time needed for the computation of the Moore-Penrose generalized inverse using the proposed algorithm (grey) and the one in Theorem 1 (black)

The benefits of the FFT based algorithm are obvious as the degrees of the matrices increase. Similar results appear if we choose matrices of different dimensions or by plotting the CPU time as a function of the dimensions while keeping the degrees constant. In order to highlight the complicacy of the Moore-Penrose inverse, note the simple fact that the denominator polynomial of the Moore-Penrose inverse of a general 3×4 $2D$ polynomial matrix with polynomial degrees 3 is of degree 18 having 361 terms, while each of the elements of the numerator is of degree 15 having 256 terms.

A perturbation analysis of the algorithm follows. Let $A(\bar{z})$ be a random multidimensional polynomial matrix with integer coefficients. Denote by $\delta_A(\bar{z}, c)$ a polynomial matrix of the same dimensions and degrees as $A(\bar{z})$ with coefficients belonging to $[-10^{-c}, 10^{-c}]$. We check the relationship between the relative error $\varepsilon_c = \frac{\left\| (A(\bar{z})+\delta_A(\bar{z},c))^+ - A^+(\bar{z}) \right\|_2}{\|A^+(\bar{z})\|_2}$ and the relative error in the data $\delta_c = \frac{\|\delta_A(\bar{z},c)\|_2}{\|A(\bar{z})\|_2}$, where $\|A(\bar{z})\|_2$ denotes the 2-norm of the coefficient matrix $[A_{0\ldots0} \ \cdots \ A_{M_1\ldots M_n}]$of $A(\bar{z})$. In order for the algorithm to be well conditioned, ε_c must be close to δ_c. The following graphs show in natural logarithmic scale the relationship between ε_c and δ_c for several values of c between $[3, 16]$.

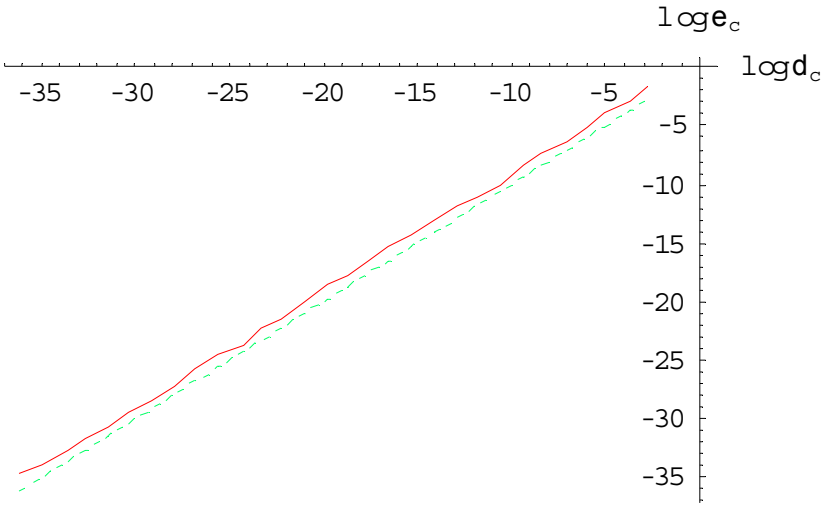


Figure 2. Perturbation analysis of the denominator polynomial of the Moore-Penrose g.i.

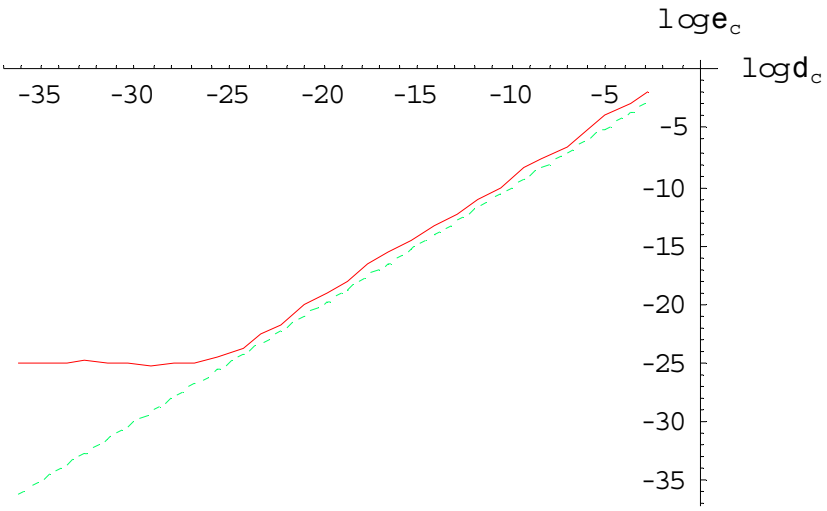


Figure 3. Perturbation analysis of the numerator of the Moore-Penrose g.i.

In order for the algorithm to be well conditioned the relative error represented by the red line must be as close as possible to the bisector (dashed line). The algorithm is very well behaved regarding the computation of the denominator polynomial of the Moore-Penrose inverse. In the case of the numerator matrix there seem to be some roundoff errors when the original matrix is perturbed by $\delta_A(\bar{z}, c)$ where $c \geq 12$. This is due to the matrix powers appearing in (15). The application

of special algorithms such as Horner's rule for its evaluation, results in less roundoff errors.

The above procedure was applied for the algorithm concerning the evaluation of the Drazin inverse with similar results.

6. Conclusions

In this paper two algorithms have been presented for determining the Moore-Penrose and Drazin inverse of nD polynomial matrices. The algorithms are based on the fast Fourier transform and therefore have the main advantages of speed in contrast to other known algorithms. The theoretical work is accompanied by a discussion of the efficiency of the algorithm under small perturbations. Possible applications of the theory introduced include model matching, the solution of multivariable Diophantine equations and its application to control system synthesis problems, the computation of the transfer function matrix of multidimensional systems, the solution of multidimensional AutoRegressive representations etc. The above mentioned algorithms may be easily extended in order to determine other kind of inverses such as {2}, {1,2}, {1,2,3} and {1,2,4} inverses of multivariable polynomial matrices by using the Leverrier-Faddeev algorithms presented in [24].

Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable comments.

References

1. Antoniou, G. E., G. O. A. Glentis, S. J. Varoufakis, and D. A. Karras: 1989, 'Transfer function determination of singular systems using the DFT'. *IEEE Trans. Circuits and Systems* **36**(8), 1140–1142.
2. Bini, D. and V. Y. Pan: 1994, *Polynomial and matrix computations. Vol. 1*, Progress in Theoretical Computer Science. Boston, MA: Birkhäuser Boston Inc. Fundamental algorithms.
3. Cooley, J. W. and J. W. Tukey: 1965, 'An algorithm for the machine calculation of complex Fourier series'. *Math. Comp.* **19**, 297–301.
4. Decell, Jr., H. P.: 1965, 'An application of the Cayley-Hamilton theorem to generalized matrix inversion'. *SIAM Rev.* **7**, 526–528. (extended in [25]).
5. Drazin, M. P.: 1958, 'Pseudo inverses in associative rings and semigroups'. *Amer. Math. Monthly* **65**, 506–514.

6. Dudgeon, D. and R. Mersereau: 1984, *Multidimensional Digital Signal Processing*. Prentice Hall.
7. Frigo, M. and S. Johnson: 1998, 'FFTW: An Adaptive Software Architecture for the FFT'. In: *ICASSP conference proceedings (vol. 3, pp. 1381-1384)*.
8. Golub, G. H. and C. F. Van Loan: 1996, *Matrix computations*, Johns Hopkins Studies in the Mathematical Sciences. Baltimore, MD: Johns Hopkins University Press, third edition.
9. Greville, T. N. E.: 1973, 'The Souriau-Frame algorithm and the Drazin pseudoinverse'. *Linear Algebra and Appl.* **6**, 205–208.
10. Güyer, T., O. Kı ymaz, G. Bilgici, and Ş. Mirasyedioğlu: 2001, 'A new method for computing the solutions of differential equation systems using generalized inverse via Maple'. *Appl. Math. Comput.* **121**(2-3), 291–299.
11. Hromcik, M. and M. Sebek: 2001, 'Fast Fourier transform and linear polynomial matrix equations'. In: *Proceedings of the 1rst IFAC Workshop on Systems Structure and Control, Prague, Czech Republic*.
12. Jones, J., N. P. Karampetakis, and A. C. Pugh: 1998, 'The computation and application of the generalized inverse via Maple'. *J. Symbolic Comput.* **25**(1), 99–124.
13. Kaczorek, T.: 2001, 'Transfer function computation for generalized n-dimensional systems'. *Journal of the Franklin Institute* **338**, 83–90.
14. Kaneko, T. and B. Liu: 1970, 'Accumulation of Round-Off Error in Fast Fourier Transforms'. *Journal of the ACM (JACM)* **17**(4), 637–654.
15. Karampetakis, N. and P. Stanimirovic: 2001, 'On the computation of the Drazin inverse of a polynomial matrix'. In: *1rst IFAC Symposium on Systems Structure and Control, Prague, Czech Republic*.
16. Karampetakis, N. P.: 1997a, 'Computation of the generalized inverse of a polynomial matrix and applications'. *Linear Algebra Appl.* **252**, 35–60.
17. Karampetakis, N. P.: 1997b, 'Generalized inverses of two-variable polynomial matrices and applications'. *Circuits Systems Signal Process.* **16**(4), 439–453.
18. Karampetakis, N. P. and P. Tzekis: 2001, 'On the computation of the generalized inverse of a polynomial matrix'. *IMA J. Math. Control Inform.* **18**(1), 83–97.
19. Lipson, J. D.: 1976, 'The fast Fourier transform its role as an algebraic algorithm'. In: *Proceedings of the annual conference of the ACM*. pp. 436–441.
20. Paccagnella, L. E. and G. L. Pierobon: 1976, 'FFT calculation of a determinantal polynomial'. *IEEE Trans. on Automatic Control* pp. 401–402.
21. Penrose, R.: 1955, 'A Generalized Inverse for Matrices'. *Proceedings of the Cambridge Philosophical Society* **51**, 406–413.
22. Schuster, A. and P. Hippe: 1992, 'Inversion of polynomial matrices by interpolation'. *IEEE Trans. Automat. Control* **37**(3), 363–365.
23. Stanimirovic, P. and N. Karampetakis: 2000, 'Symbolic implementation of Leverrier-Faddeev algorithm and applications.'. In: *8th IEEE Medit. Conference on Control and Automation, Patras, Greece.*
24. Stanimirovic, P. S.: 2002, 'A finite algorithm for generalized inverses of polynomial and rational matrices'. *Applied Mathematics and Computation.*
25. Wang, G.: 1987, 'A finite algorithm for computing the weighted Moore-Penrose inverse A^+_{MN}'. *Appl. Math. Comput.* **23**(4), 277–289.