# On the Newton Multivariate Polynomial Interpolation with Applications

Dimitris N. Varsamis and Nicholas P. Karampetakis

*Abstract*— **The main purpose of this work is to provide an optimized version of the Newton divided-difference algorithm presented by [11] for the polynomial interpolation problem. This optimized algorithm is used for the computation of the determinant of a two-variable polynomial matrix.**

## I. Introduction

Interolation, is the problem of approximating a function *f*, with another function *g* more usable, when its values at distinct points are known. When the function *g* is a polynomial we call the method *polynomial interpolation*. Polynomial interpolation of single variable functions is a rather old method in applied mathematics [6]. However, polynomial interpolation in several variables is a relatively new topic that has received not deep but constant attention from one part of the mathematical community and is today a basic subject of Approximation Theory and Numerical Analysis with applications to many mathematical problems.

The problem of the multivariate interpolation has been the concern of many scientists [2], [5], [11], [12]. Some of the most known multivariate interpolation methods are: a) the use of a multivariate Vandermode matrix and its LU factorization [12], b) the Lagrange interpolation [14], [15] c) the Hermite-Birkhoff interpolation [4] and d) the Newton-form interpolation [11], [14].

Although the one-variable interpolation has always solution for given *distinct* points, the multivariate interpolation problem through arbitrary given points may or may not have a solution when the number of unknown polynomial coefficients agree with the number of points. An interpolation problem is defined to be *poised* if it has a unique solution. Unlike the one-variable interpolation problem, the Hermite, Lagrange and Newton-form multivariate interpolation problem is not always poised. However, by selecting a specific location of the interpolation points, we can always have a unique solution.

In the special case of the *polynomial interpolation* problem, the speed of interpolation algorithms can be increased by using Discrete Fourier Transforms (DFT) techniques or better Fast Fourier Transforms (FFT) [10]. DFT-based algorithms, are used in many applications i.e. the calculation of the determinantal polynomial [13], the computation of the transfer function of generalized n-dimensional systems [1], the solutions of polynomial matrix Diophantine equations [7], and the computation of the generalized inverse of multivariable polynomial matrices [8], [17]. Recently, [9] proposed a Lagrange interpolation method for the computation of the inverse (and thus of the determinant) of a two-variable polynomial matrix by extending the results presented by [13] and [16].

In this work, we provide a better insight on the Newton-form *polynomial interpolation* method. More specifically, we present in section 2, the Newton divided-difference algorithm presented by [11] for an arbitrary function $f(x,y)$ in a slightly different way. Then, in section 3, we present an alternative algorithm for the special case where $f(x,y)$ is a polynomial with known upper bounds for the degrees of each variable. This new algorithm is using a subset of the interpolation points that Neidinger's algorithm is using and therefore it is a better one as concerns the complexity. Finally, in the last section we present an application of this new algorithm on the computation of the determinant of a two-variable polynomial matrix.

## II. The Algorithm for Multivariable Interpolating Polynomial in Newton Form

In this section, we present the Newton divided-difference algorithm presented by [11] for two-variable interpolation polynomials in a slightly different way. The main difference with [11] is the procedure that we use for computing the divided-differences.

It is known [14, Theorem 5.2.1, pp.178], that given any positive integer n and a set of points $S_{\Delta}^{n}=\left\{\left(x_i,y_j\right)\middle| i,j\geq 0, i+j\leq n\right\}$ where $x_i$ are distinct and $y_j$ are distinct, there is a *unique* polynomial of the form

$$P_n(x,y)=\sum_{i=0}^{n}\sum_{j=0}^{n-i}a_{i,j}x^i y^j \qquad (2.1)$$

that takes the same values as a given function $f(x,y)$ on

Dimitris N. Varsamis is with the Department of Informatics and Communications, Technological Educational Institute of Serres, 62124 Serres, Greece, email : dvarsam@teiser.gr.

Nicholas P. Karampetakis, is with the Department of Mathematics, Aristotle University of Thessaloniki, 54123 Thessaloniki, Greece, email : karampet@math.auth.gr.

the set $S_{\Delta}^{n}$.

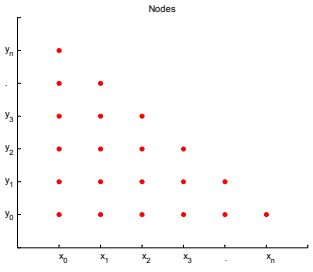


Fig. 1. An example of the set $S_{\Delta}^{n}$.

In the following algorithm, we show how to construct this interpolating polynomial by using an alternative Newton divided difference formula for a set of points $S_{\Delta}^{n}$.

**Algorithm 1.** Construction of a Newton interpolating two-variable polynomial $P_n(x,y)$ (defined in (2.1)) for a function $f(x,y)$.

**Step 1**: **k=0**. Based on the values of $f(x,y)$ at the set of points $S_{\Delta}^{n}$ we can construct the zero order table of initial values

| | $x_0$ | $x_1$ | $x_2$ | $\cdots$ | $x_n$ |
|---|---|---|---|---|---|
| $y_0$ | $P_{0,0}^{(0)}$ | $P_{1,0}^{(0)}$ | $P_{2,0}^{(0)}$ | $\cdots$ | $P_{n,0}^{(0)}$ |
| $y_1$ | $P_{0,1}^{(0)}$ | $P_{1,1}^{(0)}$ | $\cdots$ | $P_{n-1,1}^{(0)}$ | |
| $y_2$ | $P_{0,2}^{(0)}$ | $\vdots$ | $\iddots$ | | |
| $\vdots$ | $\vdots$ | $P_{1,n-1}^{(0)}$ | | | |
| $y_n$ | $P_{0,n}^{(0)}$ | | | | |

(2.2)

where $P_{i,j}^{(0)} = f(x_i, y_j)$.

**Step 2**: For **k=1(1)n**, compute the k[th]-order table of divided differences given by the recursive form

$$P_{i,j}^{(k)} := \begin{cases} \dfrac{P_{i,j}^{(k-1)} - P_{i-1,j}^{(k-1)}}{x_i - x_{i-k}} & \text{if } (j \le k-1 \wedge i > k-1) \\ \dfrac{P_{i,j}^{(k-1)} - P_{i,j-1}^{(k-1)}}{y_j - y_{j-k}} & \text{if } (i \le k-1 \wedge j > k-1) \\ \dfrac{P_{i,j}^{(k-1)} + P_{i-1,j-1}^{(k-1)} - P_{i,j-1}^{(k-1)} - P_{i-1,j}^{(k-1)}}{(x_i - x_{i-k})(y_j - y_{j-k})} & \text{if } (i > k-1 \wedge j > k-1 \wedge i+j \le n) \end{cases} \tag{2.3}$$

The $n^{th}$-order table of divided differences is given by

| | $x_0$ | $x_1$ | $x_2$ | $\cdots$ | $x_n$ |
|---|---|---|---|---|---|
| $y_0$ | $P_{0,0}^{(0)}$ | $P_{1,0}^{(1)}$ | $P_{2,0}^{(2)}$ | $\cdots$ | $P_{n,0}^{(n)}$ |
| $y_1$ | $P_{0,1}^{(1)}$ | $P_{1,1}^{(1)}$ | $\cdots$ | $P_{n-1,1}^{(n-1)}$ | |
| $y_2$ | $P_{0,2}^{(2)}$ | $\vdots$ | $\iddots$ | | |
| $\vdots$ | $\vdots$ | $P_{1,n-1}^{(n-1)}$ | | | |
| $y_n$ | $P_{0,n}^{(n)}$ | | | | |

(2.4)

since some of the elements of $P_{i,j}^{(k)}$ remain invariant in each step i.e.

$P_{0,0}^{(0)}$ in the 1[rst] step,

$\begin{pmatrix} P_{0,0}^{(0)} & P_{1,0}^{(1)} \\ P_{0,1}^{(1)} & P_{1,1}^{(1)} \end{pmatrix}$ in the 2[nd] step,

.....

$\begin{pmatrix} P_{0,0}^{(0)} & P_{1,0}^{(1)} & \cdots & P_{n-1,0}^{(n-1)} \\ P_{0,1}^{(1)} & P_{1,1}^{(1)} & \cdots & P_{n-1,1}^{(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ P_{0,n-1}^{(n-1)} & P_{1,n-1}^{(n-1)} & \cdots & 0 \end{pmatrix}$ in the n[th] step.

Therefore, we do not need to re-compute these terms in each step, by saving a lot of computations during the algorithm.

**Step 3:** The Newton interpolating polynomial is given by the following formula:

$$P_n(x,y) = Y^T A X \tag{2.5}$$

where

$$A = \begin{pmatrix} P_{0,0}^{(0)} & P_{1,0}^{(1)} & \cdots & P_{n,0}^{(n-1)} & P_{n,0}^{(n)} \\ P_{0,1}^{(1)} & P_{1,1}^{(1)} & \cdots & P_{n-1,1}^{(n-1)} & 0 \\ \vdots & \vdots & \iddots & \vdots & \vdots \\ P_{0,n-1}^{(n-1)} & P_{1,n-1}^{(n-1)} & \cdots & 0 & 0 \\ P_{0,n}^{(n)} & 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.6a}$$

$$X = \begin{pmatrix} 1 \\ x - x_0 \\ (x-x_0)(x-x_1) \\ \vdots \\ (x-x_0)(x-x_1)\cdots(x-x_{n-1}) \end{pmatrix} \in \mathbb{R}[x]^{(n+1)\times 1} \tag{2.6b}$$

$$Y = \begin{pmatrix} 1 \\ y - y_0 \\ (y-y_0)(y-y_1) \\ \vdots \\ (y-y_0)(y-y_1)\cdots(y-y_{n-1}) \end{pmatrix} \in \mathbb{R}[y]^{(n+1)\times 1} \tag{2.6c}$$

■

**Example 1** Let us find a 3rd order two-variable Newton interpolating polynomial $P_3(x,y)=\sum_{i=0}^{3}\sum_{j=0}^{3-i}a_{i,j}x^i y^j$ of a function $f(x,y)$. The number of nodes we need is $\binom{3+2}{3}=\frac{5!}{3!(5-3)!}=10$. Let these nodes,

$$P_3(x_i,y_j)=P_{i,j}^{(0)}=f(x_i,y_j)$$

for $i=0(1)3$, $j=0(1)3$ and $i+j\le 3$, given by the following table :

| $f(x_i,y_i)$ | $x_0=0$ | $x_1=1$ | $x_2=2$ | $x_3=3$ |
|---|---|---|---|---|
| $y_0=0$ | −2 | −4 | −6 | −8 |
| $y_1=1$ | −2 | 8 | 48 | |
| $y_2=2$ | −2 | 20 | | |
| $y_3=3$ | −2 | | | |

(2.7)

Applying Algorithm 1, we get:

**Step 1**: Construct the 0th-order table of initial values

| | $x_0=0$ | $x_1=1$ | $x_2=2$ | $x_3=3$ |
|---|---|---|---|---|
| $y_0=0$ | $P_{0,0}^{(0)}=-2$ | $P_{1,0}^{(0)}=-4$ | $P_{2,0}^{(0)}=-6$ | $P_{3,0}^{(0)}=-8$ |
| $y_1=1$ | $P_{0,1}^{(0)}=-2$ | $P_{1,1}^{(0)}=8$ | $P_{2,1}^{(0)}=48$ | |
| $y_2=2$ | $P_{0,2}^{(0)}=-2$ | $P_{1,2}^{(0)}=20$ | | |
| $y_3=3$ | $P_{0,n}^{(0)}=-2$ | | | |

(2.8)

**Step 2**: For **k=1(1)3**, compute the kth-order table of divided differences given by the recursive form (2.3).

For **k=1** the 1rst order table of divided differences is given by:

| | $x_0=0$ | $x_1=1$ | $x_2=2$ | $x_3=3$ |
|---|---|---|---|---|
| $y_0=0$ | $\boldsymbol{P_{0,0}^{(0)}=-2}$ | $P_{1,0}^{(1)}=-2$ | $P_{2,0}^{(1)}=-2$ | $P_{3,0}^{(1)}=-2$ |
| $y_1=1$ | $P_{0,1}^{(1)}=0$ | $P_{1,1}^{(1)}=12$ | $P_{2,1}^{(1)}=42$ | |
| $y_2=2$ | $P_{0,2}^{(1)}=0$ | $P_{1,2}^{(1)}=12$ | | |
| $y_3=3$ | $P_{0,3}^{(1)}=0$ | | | |

(2.9)

Note that the value in bold of $P_{0,0}^{(0)}$ in (2.9) is the same with the ones in (2.8), and therefore we only need ***9 computations*** for the values of $P_{i,j}^{(1)}$.

For **k=2** the 2nd order table of divided differences is given by:

| | $x_0=0$ | $x_1=1$ | $x_2=2$ | $x_3=3$ |
|---|---|---|---|---|
| $y_0=0$ | $\boldsymbol{P_{0,0}^{(0)}=-2}$ | $\boldsymbol{P_{1,0}^{(1)}=-2}$ | $P_{2,0}^{(2)}=0$ | $P_{3,0}^{(2)}=0$ |
| $y_1=1$ | $\boldsymbol{P_{0,1}^{(1)}=0}$ | $\boldsymbol{P_{1,1}^{(1)}=12}$ | $P_{2,1}^{(2)}=15$ | |
| $y_2=2$ | $P_{0,2}^{(2)}=0$ | $P_{1,2}^{(2)}=0$ | | |
| $y_3=3$ | $P_{0,3}^{(2)}=0$ | | | |

(2.10)

Note also that the values in bold of $P_{0,0}^{(0)},P_{1,0}^{(1)},P_{0,1}^{(1)},P_{1,1}^{(1)}$ in (2.10) are the same with the ones in (2.9), and therefore we only need ***6 computations*** for the values of $P_{i,j}^{(2)}$.

For **k=3** the 3rd order table of divided differences is given by:

| | $x_0=0$ | $x_1=1$ | $x_2=2$ | $x_3=3$ |
|---|---|---|---|---|
| $y_0=0$ | $\boldsymbol{P_{0,0}^{(0)}=-2}$ | $\boldsymbol{P_{1,0}^{(1)}=-2}$ | $\boldsymbol{P_{2,0}^{(2)}=0}$ | $P_{3,0}^{(3)}=0$ |
| $y_1=1$ | $\boldsymbol{P_{0,1}^{(1)}=0}$ | $\boldsymbol{P_{1,1}^{(1)}=12}$ | $\boldsymbol{P_{2,1}^{(2)}=15}$ | |
| $y_2=2$ | $\boldsymbol{P_{0,2}^{(2)}=0}$ | $\boldsymbol{P_{1,2}^{(2)}=0}$ | | |
| $y_3=3$ | $P_{0,3}^{(3)}=0$ | | | |

(2.11)

Note also that the values in bold on the above matrix are the same with the ones in (2.10), and therefore we only need ***2 computations*** for the values of $P_{i,j}^{(3)}$.

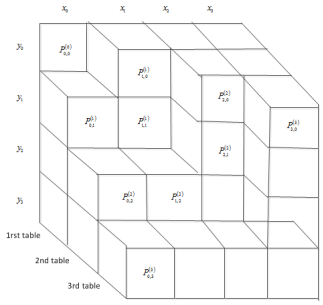


**Fig.2** Values that remain invariants in the next steps.

Therefore, the total computations that we need for the construction of the final matrix are 9+6+2=17.

**Step 3:** The Newton interpolating polynomial is the following:

$$P_3(x,y)=Y^T AX=-2-2x-3xy+15x^2y$$

where

$$A=\begin{pmatrix} P_{0,0}^{(0)}=-2 & P_{1,0}^{(1)}=-2 & P_{2,0}^{(2)}=0 & P_{3,0}^{(3)}=0 \\ P_{0,1}^{(1)}=0 & P_{1,1}^{(1)}=12 & P_{2,1}^{(2)}=15 & 0 \\ P_{0,2}^{(2)}=0 & P_{1,2}^{(2)}=0 & 0 & 0 \\ P_{0,3}^{(3)}=0 & 0 & 0 & 0 \end{pmatrix}$$

and

$$X=\begin{pmatrix}1\\ x\\ x(x-1)\\ x(x-1)(x-2)\end{pmatrix}\quad;\quad Y=\begin{pmatrix}1\\ y\\ y(y-1)\\ y(y-1)(y-2)\end{pmatrix}$$

■

## III. Interpolating Bivariate Polynomials With Known Degrees in Terms of Each Variable

The previous section has focused on the computation of a n[th] order Newton two-variable interpolating polynomial for a function $f(x,y)$. In this section, we are considering the special case where the function $f(x,y)$ is a two-variable polynomial $p(x,y)$ with known upper bounds on the degrees in terms of x and y (let $k_1$ and $k_2$ respectively) i.e.

$$f(x,y)\equiv p(x,y)=\sum_{i=0}^{k_1}\sum_{j=0}^{k_2}a_{i,j}x^i y^j \tag{3.1}$$

The total order of $p(x,y)$ is $n=k_1+k_2$. Therefore, according to Algorithm 1, we can approximate it, by a $n^{\text{th}}$ order Newton two-variable interpolating polynomial

$$P_n(x,y)=\sum_{i=0}^{n}\sum_{j=0}^{n-i}a_{i,j}x^i y^j \tag{3.2}$$

if we know its values at a set

$$S_\Delta^n=\left\{(x_i,y_j)\middle| i,j\ge 0, i+j\le n\right\}$$

of $N=\binom{n+2}{n}=\dfrac{(n+2)!}{n!2!}=\dfrac{(k_1+k_2+2)!}{(k_1+k_2)!2!}$ points.

Consider the matrices $Y, A$ and $X$ defined in (2.6) :

$$A=\begin{pmatrix}\boldsymbol{A_{1,1}} & A_{1,2}\\ A_{2,1} & A_{2,2}\end{pmatrix}=$$

$$=\left(\begin{array}{ccccc|ccc}\boldsymbol{P_{0,0}^{(n)}} & \boldsymbol{P_{1,0}^{(n)}} & \boldsymbol{P_{2,0}^{(n)}} & \cdots & \boldsymbol{P_{k_1,0}^{(n)}} & & \cdots & P_{n,0}^{(n)}\\ \boldsymbol{P_{0,1}^{(n)}} & \boldsymbol{P_{1,1}^{(n)}} & \cdots & & \vdots & & P_{n-1,1}^{(n)} & \\ \boldsymbol{P_{0,2}^{(n)}} & \vdots & & & \vdots & \iddots & & \\ \vdots & & & & \vdots & \iddots & & \\ \boldsymbol{P_{0,k_2}^{(n)}} & \cdots & \cdots & \cdots & \boldsymbol{P_{k_1,k_2}^{(n)}} & & & \\ \hline & & & \iddots & & & & \\ \vdots & & \iddots & & & & & \\ & P_{1,n-1}^{(n)} & & & & & & \\ P_{0,n}^{(n)} & & & & & & & \end{array}\right) \tag{3.3a}$$

$$X=\begin{pmatrix}\boldsymbol{X_1}\\ X_2\end{pmatrix}=\begin{pmatrix}\boldsymbol{1}\\ \boldsymbol{x-x_0}\\ \vdots\\ \dfrac{(\boldsymbol{x-x_0})(\boldsymbol{x-x_1})\cdots(\boldsymbol{x-x_{k_1-1}})}{(x-x_0)(x-x_1)\cdots(x-x_{k_1})}\\ \vdots\\ (x-x_0)(x-x_1)\cdots(x-x_{n-1})\end{pmatrix} \tag{3.3b}$$

$$Y=\begin{pmatrix}\boldsymbol{Y_1}\\ Y_2\end{pmatrix}=\begin{pmatrix}\boldsymbol{1}\\ \boldsymbol{y-y_0}\\ \vdots\\ \dfrac{(\boldsymbol{y-y_0})(\boldsymbol{y-y_1})\cdots(\boldsymbol{y-y_{k_2-1}})}{(y-y_0)(y-y_1)\cdots(y-y_{k_2})}\\ \vdots\\ (y-y_0)(y-y_1)\cdots(y-y_{n-1})\end{pmatrix} \tag{3.3c}$$

Then

$$P_n(x,y)=\begin{pmatrix}\mathrm{Y}_1 & Y_2\end{pmatrix}\begin{pmatrix}A_{1,1} & A_{1,2}\\ A_{2,1} & 0_{2,2}\end{pmatrix}\begin{pmatrix}\mathrm{X}_1\\ X_2\end{pmatrix}= \tag{3.4}$$
$$=\mathrm{Y}_1^T\mathrm{A}_{1,1}\mathrm{X}_1+Y_2^T A_{2,1}\mathrm{X}_1+Y_1^T A_{1,2}X_2$$

The coefficients of $y^j$ with $j>k_2-1$ in the following term

$$Y_2^T A_{2,1}\mathrm{X}_1=$$
$$=\begin{pmatrix}(y-y_0)\times\cdots\times(y-y_{k_2}) & \ldots & (y-y_0)\times\cdots\times(y-y_{k_1+k_2-1})\end{pmatrix}\times$$
$$\times\begin{pmatrix}P_{0,k_2+1}^{(n)} & P_{1,k_2+1}^{(n)} & \cdots & P_{k_1-1,k_2+1}^{(n)} & 0\\ P_{0,k_2+2}^{(n)} & P_{1,k_2+2}^{(n)} & \iddots & 0 & 0\\ \vdots & \iddots & \iddots & \vdots & \vdots\\ P_{0,k_1+k_2}^{(n)} & 0 & \cdots & \cdots & 0\end{pmatrix}\begin{pmatrix}1\\ x-x_0\\ \vdots\\ (x-x_0)(x-x_1)\cdots(x-x_{k_1-1})\end{pmatrix} \tag{3.5}$$

are equal to zero, since according to our initial assumption the upper bound of the degree in terms of $y$ does not exceed $k_2$. Therefore, $P_{i,j}^{(n)}=0$ for $i=0,1,\ldots,k_1$ and $j=k_2+1,k_2+2,\ldots,k_1+k_2$ $(n=k_1+k_2)$ or equivalently $Y_2^T A_{2,1}\mathrm{X}_1=0$. Similarly, we can show that $P_{i,j}^{(n)}=0$ for $i=k_1+1,k_1+2,\ldots,k_1+k_2$ and $j=0,1,\ldots,k_2$ $(n=k_1+k_2)$ or equivalently $Y_1^T A_{1,2}X_2=0$ due to our initial assumption that the upper bound of the degree in terms of $x$ does not exceed $k_1$. Thus, the matrix $A$ must has the following form

$$A=\begin{pmatrix}\boldsymbol{A_{1,1}} & 0\\ 0 & 0\end{pmatrix} \tag{3.6}$$

and $P_n(x,y)$ is reduced to

$$P_n(x,y)=\mathrm{Y}_1^T\mathrm{A}_{1,1}\mathrm{X}_1 \tag{3.7}$$

Therefore, in order to compute $A_{1,1}$ we need only $(k_1+1)(k_2+1)$ of the $N=(k_1+k_2+1)(k_1+k_2+2)/2$ initial values of $p(x,y)$ i.e. $\left(k_1+k_1^2+k_2+k_2^2\right)/2$ less values. In case where these values are coming from complex computations i.e. the determinant of a two-variable matrix at these specific initial values, then we save a lot of computations from the very beginning.

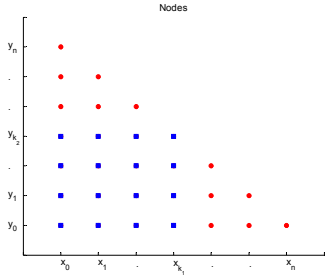


Fig. 3. An example of the set $S_{\Delta}^{n}$ (red and blue) and the set points we actually need (in blue).

Based on the above results, Algorithm 1 can be reduced to the next less consuming in speed and memory algorithm.

**Algorithm 2.** Construction of a Newton interpolating two-variable polynomial $P_n(x,y)$ (defined in (3.2)) for a two-variable polynomial $p(x,y)$ with upper bound on the degree of $x$ (resp. $y$) $k_1$ (resp. $k_2$).

**Step 1**: **k=0**. Based on the values of $p(x,y)$ at the set of points $\tilde{S}_{\Delta}^{n}=\left\{(x_i,y_j)\middle| i=0,1,...,k_1, j=0,1,...,k_2\right\}$, create the k[th]-order table of initial values $P_{i,j}^{(0)}:=p(x_i,y_j)$

$$\begin{array}{c|ccccc} & x_0 & x_1 & x_2 & \cdots & x_{k_1} \\ \hline y_0 & P_{0,0}^{(0)} & P_{1,0}^{(0)} & P_{2,0}^{(0)} & \cdots & P_{k_1,0}^{(0)} \\ y_1 & P_{0,1}^{(0)} & P_{1,1}^{(0)} & P_{2,1}^{(0)} & \cdots & P_{k_1,1}^{(0)} \\ y_2 & P_{0,2}^{(0)} & P_{1,2}^{(0)} & P_{2,2}^{(0)} & \cdots & P_{k_1,2}^{(0)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ y_{k_2} & P_{0,k_2}^{(0)} & P_{1,k_2}^{(0)} & P_{2,k_2}^{(0)} & \cdots & P_{k_1,k_2}^{(0)} \end{array} \tag{3.8}$$

**Step 2**: For **k=1(1)** $\tilde{n}$, where $\tilde{n}=\max\{k_1,k_2\}$ compute the $\tilde{n}$ [th]-order table of divided differences given by the recursive form (2.3)

$$\begin{array}{c|ccccc} & x_0 & x_1 & x_2 & \cdots & x_{k_1} \\ \hline y_0 & P_{0,0}^{(\tilde{n})} & P_{1,0}^{(\tilde{n})} & P_{2,0}^{(\tilde{n})} & \cdots & P_{k_1,0}^{(\tilde{n})} \\ y_1 & P_{0,1}^{(\tilde{n})} & P_{1,1}^{(\tilde{n})} & P_{2,1}^{(\tilde{n})} & \cdots & P_{k_1,1}^{(\tilde{n})} \\ y_2 & P_{0,2}^{(\tilde{n})} & P_{1,2}^{(\tilde{n})} & P_{2,2}^{(\tilde{n})} & \cdots & P_{k_1,2}^{(\tilde{n})} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ y_{k_2} & P_{0,k_2}^{(\tilde{n})} & P_{1,k_2}^{(\tilde{n})} & P_{2,k_2}^{(\tilde{n})} & \cdots & P_{k_1,k_2}^{(\tilde{n})} \end{array} \tag{3.9}$$

which coincides for example in case where $\tilde{n}=k_1>k_2$ with the matrix

$$\begin{array}{c|ccccccc} & x_0 & x_1 & x_2 & \cdots & x_{k_2} & \cdots & x_{k_1} \\ \hline y_0 & P_{0,0}^{(0)} & P_{1,0}^{(1)} & P_{2,0}^{(2)} & \cdots & P_{k_2,0}^{(k_2)} & \cdots & P_{k_1,0}^{(k_1)} \\ y_1 & P_{0,1}^{(1)} & P_{1,1}^{(1)} & P_{2,1}^{(2)} & \cdots & P_{k_2,1}^{(k_2)} & \cdots & P_{k_1,1}^{(k_1)} \\ y_2 & P_{0,2}^{(2)} & P_{1,2}^{(2)} & P_{2,2}^{(2)} & \cdots & P_{k_2,2}^{(k_2)} & \cdots & P_{k_1,2}^{(k_1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ y_{k_2} & P_{0,k_2}^{(k_2)} & P_{1,k_2}^{(k_2)} & P_{2,k_2}^{(k_2)} & \cdots & P_{k_2,k_2}^{(k_2)} & \cdots & P_{k_1,k_2}^{(k_1)} \end{array}$$

**Step 3:** The Newton interpolating polynomial is the following :

$$P_n(x,y)\equiv p(x,y)=\mathrm{Y}_1^T\mathrm{A}_{1,1}\mathrm{X}_1 \tag{3.10}$$

where $\mathrm{Y}_1,\mathrm{A}_{1,1},\mathrm{X}_1$ have been defined in (3.3). ■

Let us assume that in order to compute

$$\frac{P_{i,j}^{(k-1)}-P_{i-1,j}^{(k-1)}}{x_i-x_{i-k}} \text{ or } \frac{P_{i,j}^{(k-1)}-P_{i,j-1}^{(k-1)}}{y_j-y_{j-k}} \tag{3.11}$$

in (2.3) we need ***a*** operations (2 additions and 1 multiplication), whereas for

$$\frac{P_{i,j}^{(k-1)}+P_{i-1,j-1}^{(k-1)}-P_{i,j-1}^{(k-1)}-P_{i-1,j}^{(k-1)}}{(x_i-x_{i-k})(y_j-y_{j-k})} \tag{3.12}$$

we need ***b*** operations (5 additions and 2 multiplications).

**Complexity of Algorithm 1.**

The computation of the first-order table, requires $na$ operations for the first row, $na$ operations for the first column and $\frac{n(n-1)}{2}b$ operations for the rest elements. Thus, the total number of operations is $s_1=2na+\frac{n(n-1)}{2}b$ or $s_1=2(S_n-S_{n-1})a+S_{n-1}b$ where $S_n$ the sum of arithmetic progression with $a_1=1$ and $d=1$.

The computation of the second-order table, requires $\left[(n-1)+(n-2)\right]a$ operations for the first and second row

and $\left[(n-1)+(n-2)\right]a$ operations for the first and second column, since the elements in

$$\begin{pmatrix} P_{0,0}^{(0)} & P_{1,0}^{(1)} \\ P_{0,1}^{(1)} & P_{1,1}^{(1)} \end{pmatrix}$$

remain invariant and $\left(\frac{(n-2)(n-3)}{2}\right)b$ operations for the rest elements. Thus, the total number of operations is $s_2 = 2\left[(n-1)(n-2)\right]a + \left(\frac{(n-2)(n-3)}{2}\right)b$ or $s_2 = 2\left(S_{n-1} - S_{n-3}\right)a + S_{n-3}b$.

Finally, the computation of the $n^{\text{th}}$-order table, requires $a$ operations for the n first rows and $a$ operations for the n first columns, since the elements in

$$\begin{pmatrix} P_{0,0}^{(0)} & P_{1,0}^{(1)} & \cdots & P_{n-1,0}^{(n-1)} \\ P_{0,1}^{(1)} & P_{1,1}^{(1)} & \cdots & P_{n-1,1}^{(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ P_{0,n-1}^{(n-1)} & P_{1,n-1}^{(n-1)} & \cdots & 0 \end{pmatrix}$$

remain invariant and no one operation for the rest elements. Thus, the total number of operations is $s_n = 2a$ or $s_n = 2S_1 a$ .

| Steps | Operations |
|---|---|
| $s_1$ | $2\left(S_n - S_{n-1}\right)a + S_{n-1}b$ |
| $s_2$ | $2\left(S_{n-1} - S_{n-3}\right)a + S_{n-3}b$ |
| …. | …. |
| $s_n$ | $2S_1 a$ |

Thus, the total operations for Algorithm 1, are given by

$$s = \sum_{i=1}^{n} s_i =$$
$$= \left[2\left(S_n - S_{n-1}\right)a + S_{n-1}b\right] + \left[2\left(S_{n-1} - S_{n-3}\right)a + S_{n-3}b\right] +$$
$$+ \ldots + \left[2S_1 a\right]$$

or equivalently by

$$s = \frac{1}{24}n(n+2)\left[2(2n+5)a + (2n-1)b\right]$$

if n is even, and

$$s = \frac{1}{24}(n+1)\left[2(n+3)(2n+1)a + (n-1)(2n+3)b\right]$$

if n is odd.

Therefore, the total complexity is $O\left(n^3\right)$ or equivalently $O\left(\left(k_1 + k_2\right)^3\right)$ (apart from Step 1).

**Complexity of Algorithm 2.**

The computation of the first-order table, requires $k_1 a$ operations for the first row, $k_2 a$ operations for the first column and $k_1 k_2 b$ operations for the rest elements. Thus, the total number of operations is $s_1 = (k_1 + k_2)a + k_1 k_2 b$ .

The computation of the second-order table requires $2(k_1 - 1)a$ operations for the first and second row, $2(k_2 - 1)a$ operations for the first and second column and $(k_1 - 1)(k_2 - 1)b$ operations for the rest elements. Thus, the total number of operations is $s_2 = 2(k_1 - 1)2(k_2 - 1)a + (k_1 - 1)(k_2 - 1)b$ .

Under the assumption that $k_1 > k_2$ the computation of the $k_2$ -order table, requires $k_2(k_1 + 1 - k_2)a$ operations for the $k_2$ first rows, $k_2 a$ operations for the $k_2$ first columns and $(k_1 + 1 - k_2)b$ operations for the rest elements. Thus, the total number of operations is $s_{k_2} = k_2(k_1 + 1 - k_2)a + k_2 a + (k_1 + 1 - k_2)b$ .

Finally, the computation of the divided-differences of the $k_1$ th order table, requires $(k_2 + 1)a$ operations for the $(k_2 + 1)$ first rows, and no one operation for the rest elements. Thus, the total number of operations is $s_{k_1} = (k_2 + 1)a$ .

| Steps | Operations |
|---|---|
| $s_1$ | $(k_1 + k_2)a + k_1 k_2 b$ |
| $s_2$ | $2(k_1 - 1)2(k_2 - 1)a + (k_1 - 1)(k_2 - 1)b$ |
| …. | …. |
| $s_{k_2}$ | $k_2(k_1 + 1 - k_2)a + k_2 a + (k_1 + 1 - k_2)b$ |
| … | …. |
| $s_{k_1}$ | $(k_2 + 1)a$ |

Thus, the total operations for Algorithm 2, are given by

$$s = \sum_{i=1}^{\tilde{n}} s_i =$$
$$= \left[(k_1 + k_2)a + k_1 k_2 b\right] +$$
$$+ \left[2(k_1 - 1)2(k_2 - 1)a + (k_1 - 1)(k_2 - 1)b\right] + \cdots +$$
$$+ \left[k_2(k_1 + 1 - k_2)a + k_2 a + (k_1 + 1 - k_2)b\right] + \cdots + \left[(k_2 + 1)a\right]$$

or equivalently by

$$s = \frac{a}{6}(k_2 + 1)\left(3k_1 + k_2 + 3k_1^2 - 3k_1 k_2 + 2k_2^2\right) +$$
$$+ \frac{b}{6}\left(k_2 + 3k_1 k_2 + 3k_1 k_2^2 - k_2^3\right)$$

The worst case is when $k_1 = k_2$ . In that case the total number of computations is

$$s = a\frac{\left(4k_1 + 2k_1^2\right)\left(k_1 + 1\right)}{6} + b\left(\frac{k_1 + 3k_1^2 + 2k_1^3}{6}\right)$$

and therefore, the worst case complexity is $O\left(k_1^3\right)$.

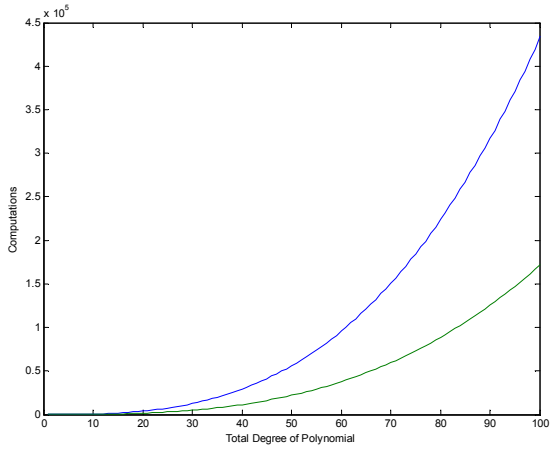


Fig. 5. The total number of operations (if a=1 and b=2) in step 2 of Algorithm 1 (blue) and Algorithm 2 (green) when $k_1 = k_2$.

**Example 2.** Consider the Example 1. Suppose that we know from the beginning that the polynomial $p(x,y)$ that we are looking for, has an upper bound 2 (resp. 1) for the degree in terms of $x$ (resp. $y$) i.e. $k_1 = 2$, $k_2 = 1$ and $n = 2+1 = 3$. In order to approximate $p(x,y)$ with a 3rd order polynomial $P_3(x,y)$, we have to know its values at a set

$$S_\Delta^3 = \left\{\left(x_i, y_j\right) \middle| i, j \geq 0, i + j \leq 3\right\}$$

of $N = \binom{3+2}{3} = 10$ points. However, according to the above theory and the results of Example 1, we have that:

**Step 1**: We create the table of initial values

| | $x_0 = 0$ | $x_1 = 1$ | $x_2 = 2$ |
|---|---|---|---|
| $y_0 = 0$ | $P_{0,0}^{(0)} = -2$ | $P_{1,0}^{(0)} = -4$ | $P_{2,0}^{(0)} = -6$ |
| $y_1 = 1$ | $P_{0,1}^{(0)} = -2$ | $P_{1,1}^{(0)} = 8$ | $P_{2,1}^{(0)} = 48$ |

**Step 2**: Compute k[th] order divided differences according to the recursive form (2.3).

For **k=1** the 1st order table of divided differences is

| | $x_0 = 0$ | $x_1 = 1$ | $x_2 = 2$ |
|---|---|---|---|
| $y_0 = 0$ | $\boldsymbol{P_{0,0}^{(0)} = -2}$ | $P_{1,0}^{(1)} = -2$ | $P_{2,0}^{(1)} = -2$ |
| $y_1 = 1$ | $P_{0,1}^{(1)} = 0$ | $P_{1,1}^{(1)} = 12$ | $P_{2,1}^{(1)} = 42$ |

For **k=2** the 2nd order table of divided differences is

| | $x_0 = 0$ | $x_1 = 1$ | $x_2 = 2$ |
|---|---|---|---|
| $y_0 = 0$ | $\boldsymbol{P_{0,0}^{(0)} = -2}$ | $\boldsymbol{P_{1,0}^{(1)} = -2}$ | $P_{2,0}^{(2)} = 0$ |
| $y_1 = 1$ | $\boldsymbol{P_{0,1}^{(1)} = 0}$ | $\boldsymbol{P_{1,1}^{(1)} = 12}$ | $P_{2,1}^{(2)} = 15$ |

**Step 3**: The interpolating polynomial is :

$$p(x,y) = P_3(x,y) = Y_1^T A_{1,1} X_1 = -2 - 2x - 3xy + 15x^2 y$$

where

$$A_{1,1} = \begin{pmatrix} P_{0,0}^{(0)} = -2 & P_{1,0}^{(1)} = -2 & P_{2,0}^{(2)} = 0 \\ P_{0,1}^{(1)} = 0 & P_{1,1}^{(1)} = 12 & P_{2,1}^{(2)} = 15 \end{pmatrix}$$

and

$$X_1 = \begin{pmatrix} 1 \\ x \\ x(x-1) \end{pmatrix};\ Y_1 = \begin{pmatrix} 1 \\ y \end{pmatrix}$$

In order to compute the matrix $A_{1,1}$ we need to evaluate $(k_1+1)(k_2+1) = 3\times 2 = 6$ initial values of $p(x,y)$, instead of 10 in Algorithm 1. The total number of operations in Step 2 is 7 instead of 17 in Algorithm 1. Finally, in Step 3 we have a multiplication of 3 matrices with dimensions $(1\times 2)(2\times 3)(3\times 1)$ instead of $(1\times 4)(4\times 4)(4\times 1)$ in Algorithm 1.

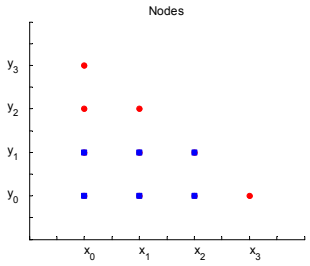


Fig. 4. An example of the set $S_\Delta^3$ (red and blue) and the set points we actually need (in blue).

■

## IV. On the Computation of the Determinant of a Two-Variable Polynomial Matrix

In this section we apply Algorithm 2 for the computation of a two-variable polynomial matrix.

Let the polynomial matrix

$$A(x,y) = \begin{pmatrix} -1 & 0 & x \\ 5 & 1 & -1 \\ 2 & 3xy & 2 \end{pmatrix}$$

**Step 0:** We compute the upper bounds on the degree of $x$ (resp. $y$) $k_1$ (resp. $k_2$) in $p(x,y) = \det A(x,y)$. Since the greatest degree in terms of $x$ of each column of the matrix is 0, 1 and 1 respectively, then $k_1 = 0+1+1 = 2$. Working in the same way for $y$ we have that $k_2 = 0+1+0 = 1$.

**Step 1**: We create the table of initial values $P_{i,j}^{(0)} := p\left(x_i, y_j\right) = \det A\left(x_i, y_i\right)$ :

| | $x_0 = 0$ | $x_1 = 1$ | $x_2 = 2$ |
|---|---|---|---|
| $y_0 = 0$ | $P_{0,0}^{(0)} = -2$ | $P_{1,0}^{(0)} = -4$ | $P_{2,0}^{(0)} = -6$ |
| $y_1 = 1$ | $P_{0,1}^{(0)} = -2$ | $P_{1,1}^{(0)} = 8$ | $P_{2,1}^{(0)} = 48$ |

which is the same with the ones of Example 1.
Since the computation of the determinant of a constant matrix $A \in \mathbb{R}^{\ell \times \ell}$ by using the Gauss method requires $O\left(\ell^3\right)$ operations, the computation of initial values needs $O\left(R\ell^3\right)$ operations where $R = \left(k_1 + 1\right)\left(k_2 + 1\right)$. The first step is similar to the ones used by [9] with the only difference that DFT methods are using specific points $\left(x_i, y_j\right)$ with $x_i$ , $y_j$ belonging to the unit circle. Thus, on the DFT methods we don't need to evaluate all the initial values. Apart of this, the complexity of the respective DFT method in [9] is $O\left(R\log\left(R\right)\right)$ in contrast to $O\left(\left(\max\left(k_1, k_2\right)\right)^3\right)$ in Step 2 of Algorithm 2. Therefore, the DFT method is better than the ones in Algorithm 2.

**Step 2**: For *k*=1(1)*n*, compute the k[th] order table of divided differences according to the recursive form (2.3).
All the tables are given in Example 2 i.e. the 3[rd] order table is given by

| | $x_0 = 0$ | $x_1 = 1$ | $x_2 = 2$ |
|---|---|---|---|
| $y_0 = 0$ | $\boldsymbol{P_{0,0}^{(0)} = -2}$ | $\boldsymbol{P_{1,0}^{(1)} = -2}$ | $P_{2,0}^{(2)} = 0$ |
| $y_1 = 1$ | $\boldsymbol{P_{0,1}^{(1)} = 0}$ | $\boldsymbol{P_{1,1}^{(1)} = 12}$ | $P_{2,1}^{(2)} = 15$ |

**Step 3:** Compute the interpolating polynomial in Newton form according to the following form

$$p\left(x, y\right) = P_3\left(x, y\right) = Y_1^T A_{1,1} X_1 = -2 - 2x - 3xy + 15x^2 y$$

where the matrices $Y_1, A_{1,1}, X_1$ have been defined in Example 2.

## V. CONCLUSION

A modification of the Newton divided-difference algorithm [11] for the special case of two-variable polynomials has been given. The complexity of this new algorithm has proved that it is better than the ones given in [11]. The reason is twofold: a) it uses less interpolation points and b) it does't need to re-compute elements that remain invariant in each step of the algorithm. The proposed algorithm has been tested on the computation of the determinant of a two-variable polynomial matrix, and thus providing a different way of approximation from the known ones in [9] and [17]. Further applications of this algorithm might be the computation of : a) the transfer function of two-variable systems, b) the solution of polynomial matrix Diophantine equations, and c) the generalized inverse of two-variable polynomial matrices. Following similar lines, the proposed algorithm can be extended to more variables than two.

## REFERENCES


[1] Antoniou E. G., 2001, Transfer Function Computation for Generalized Ndimensional Systems, Journal of the Franklin Institute, vol. 338, pp. 83-90.
[2] de Boor, C., and Ron, A., On multivariate polynomial interpolation, Constr. Approx. 6 (1990), 287–302.
[3] de Boor, C., A multivariate divided difference, in: Approximation theory VIII, Vol. 1, C.K. Chui and L.L. Schumaker, eds., World Sci. Publishing, River Edge, NJ, 1995, pp. 87–96.
[4] Gasca M. and Martinez Jose J., 1992, Bivariate Hermite-Birkhof interpolation and Vandermonde determinants. Numerical Algorithms, 2, 193-199.
[5] Gasca M. and Sauer T., Polynomial interpolation in several variables, Advances in Computational Math. 12 (2000), 377-410.
[6] Gasca M. and Sauer T., 2000, On the history of multivariate polynomial interpolation. Numerical analysis 2000, Vol. II: Interpolation and extrapolation. J. Comput. Appl. Math., 122, no. 1-2, 23-35.
[7] Hromcik M. and Sebek M., 2001, Fast Fourier Transform and Linear Polynomial Matrix Equations, Proceedings of the 1rst IFAC Workshop on Systems Structure and Control, Prague, Czech Republic.
[8] Karampetakis N.P., Vologiannidis S., 2003, DFT calculation of the generalized and Drazin inverse of a polynomial matrix, Applied Mathematics and Computation, 143, 501-521.
[9] Karampetakis N. P. and Evripidou A., 2010, On the computation of the inverse of a two-variable polynomial matrix by interpolation, Multidim Syst Sign Process, DOI 10.1007/s11045-010-0102-7.
[10] Lipson John D., 1976, The Fast Fourier Transform, Its role as an Algebraic Algorithm. Proceedings of the ACM Annual Conference/Annual Meeting, 436-441.
[11] Neidinger R. D., Multivariable Interpolating Polynomials in Newton Forms, Joint Mathematics Meetings 2009, Washington D.C., January 5-8, 2009.
[12] Olver P. J., On Multivariate Interpolation, Studies in Applied Math., 116 (2006) 201-240.
[13] Paccagnella L. E. and Pierobon G. L., 1976, FFT Calculation of a Determinantal Polynomial, IEEE Trans. Automatic Control, 29, Issue 3, 401-402.
[14] Philips G. M., 2003, Interpolation and approximation by polynomials, Springer-Verlag.
[15] Sauer T. and Xu Y., On multivariate Lagrange interpolation, Mathematics of Computation 64 (1995) 1147-1170.
[16] Schuster A. and Hippe P., 1992, Inversion of polynomial matrices by interpolation, IEEE Transactions on Automatic Control, 37, 363-365.
[17] Vologiannidis S. and Karampetakis N.P., 2004, Inverses of Multivariable Polynomial Matrices by Discrete Fourier Transforms, Multidimensional Systems and Signal Processing, 15, 341-361.