On a Special Case of the two-variable Newton Interpolation Polynomial

Dimitris N. Varsamis and Nicholas P. Karampetakis

***Abstract*— The paper, proposes two new algorithms for the construction of a two-variable Newton-interpolation polynomial, for the special case where we have a rectangular basis with equidistant points. The complexity of the proposed algorithms is better than the ones given in [1] since it is based only on additions. One of the two algorithms is based on matrix multiplications and thus it is easily implemented in programming languages which supports such kind of operations.**

I. Introduction

Interpolation, 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 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.

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. The most known bases of interpolation points for two-variable polynomial interpolation are the triangular and the rectangular basis. The most known methods for multivariate interpolation are : a) the use of a multivariate Vandermode matrix and its LU factorization [2], b) the Lagrange interpolation [3], [4], c) the Hermite-Birkhoff interpolation [5] d) the Newton-form interpolation [4], [6], [1] and e) the Discrete Fourier Transforms (DFT) techniques or better Fast Fourier Transforms (FFT) [7].

Polynomial interpolation plays a crucial role in a number of applications such as : the calculation of the determinantal polynomial [8], [9], the computation of the transfer function of generalized n-dimensional systems [10], the solutions of polynomial matrix Diophantine equations [11], and the computation of the inverse or the generalized inverse of multivariable polynomial matrices [12], [13], [9], [14].

D. Varsamis is with Department of Informatics & Communications, Technological Educational Institution of Serres, 62124 Serres, Greece dvarsam@teiser.gr

N. Karampetakis is with the Department of Mathematics, Aristotle University of Thessaloniki, Thessaloniki, 54124, Greece karampet@math.auth.gr

The Newton multivariate polynomial interpolation can be applied both to triangular and rectangular bases. In the special case where the function f is polynomial with known upper bound on the degree of x (resp. y) let k_1 (resp. k_2), we use the rectangular basis. The difference of the two bases is based on the selection of the sets of the interpolation points. For the triangular basis the set is

$$S_{\Delta}^{n} = \{(x_i, y_j) \mid i, j \in \mathbb{N}, i + j \leq n\}$$

and the number of interpolation points is $\binom{n+2}{n} = \frac{(n+2)!}{n!2!}$, whereas for the rectangular basis the set is

$$\tilde{S}_{\Delta}^{(k_1,k_2)} = \{(x_i, y_i) \mid i = 0, 1, \ldots, k_1 \ , \ j = 0, 1, \ldots, k_2\}$$

and the number of interpolation points is $(k_1 + 1)(k_2 + 1)$ with $k_1 + k_2 = n$. In that case the computational cost for the construction of the Newton interpolating polynomial is less than the ones with the triangular basis [1] and the interpolation polynomial is given by the following formula:

$$\tilde{p}(x, y) \equiv p(x, y) = \tilde{X}_p^T \cdot \tilde{P} \cdot \tilde{Y}_p \tag{1}$$

where

$$\tilde{X}_p = \begin{pmatrix} 1 \\ x - x_0 \\ (x - x_0)(x - x_1) \\ \vdots \\ (x - x_0)(x - x_1) \cdots (x - x_{k_1 - 1}) \end{pmatrix} \in \mathbb{R}[x]^{(k_1+1)} \tag{2}$$

$$\tilde{Y}_p = \begin{pmatrix} 1 \\ y - y_0 \\ (y - y_0)(y - y_1) \\ \vdots \\ (y - y_0)(y - y_1) \cdots (y - y_{k_2 - 1}) \end{pmatrix} \in \mathbb{R}[y]^{(k_2+1)}$$

and in case $k_1 > k_2$

$$\tilde{P} = \begin{pmatrix} p_{0,0}^{(0)} & p_{0,1}^{(1)} & p_{0,2}^{(2)} & \cdots & p_{0,k_2}^{(k_2)} \\ p_{1,0}^{(1)} & p_{1,1}^{(1)} & p_{1,2}^{(2)} & \cdots & p_{1,k_2}^{(k_2)} \\ p_{2,0}^{(2)} & p_{2,1}^{(2)} & p_{2,2}^{(2)} & \cdots & p_{2,k_2}^{(k_2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{k_2,0}^{(k_2)} & p_{k_2,1}^{(k_2)} & p_{k_2,2}^{(k_2)} & \cdots & p_{k_2,k_2}^{(k_2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{k_1,0}^{(k_1)} & p_{k_1,1}^{(k_1)} & p_{k_1,2}^{(k_1)} & \cdots & p_{k_1,k_2}^{(k_1)} \end{pmatrix} \in \mathbb{R}^{(k_1+1)\times(k_2+1)} \tag{3}$$

where $p_{i,j}^{(k)}$ with $i \leq k_1$ and $j \leq k_2$ are given by the recursive type

$$p_{i,j}^{(k)} := \begin{cases} \frac{p_{i,j}^{(k-1)} - p_{i-1,j}^{(k-1)}}{x_i - x_{i-k}} & if \quad \begin{pmatrix} j < k \\ i \geq k \end{pmatrix} \\ \frac{p_{i,j}^{(k-1)} - p_{i,j-1}^{(k-1)}}{y_j - y_{j-k}} & if \quad \begin{pmatrix} i < k \\ j \geq k \end{pmatrix} \\ \frac{p_{i,j}^{(k-1)} + p_{i-1,j-1}^{(k-1)} - p_{i-1,j}^{(k-1)} - p_{i,j-1}^{(k-1)}}{(x_i - x_{i-k})(y_j - y_{j-k})} & if \quad \begin{pmatrix} i \geq k \\ j \geq k \end{pmatrix} \\ p_{i,j}^{(k-1)} & if \quad \begin{pmatrix} i < k \\ j < k \end{pmatrix} \end{cases} \tag{4}$$

In this work, we initially study the form of the above recursive formula (4) in the special case where the interpolation points are equidistant. Then, we try to rewrite this recursive formula in terms of matrix multiplications. Finally, the complexity of the proposed algorithms that comes from this special case, is studied and a discussion is given about its implementation in a parallel programming environment. The whole theory is illustrated via a specific example in order to show the difference between the suggested algorithms.

II. The two-variable Newton Interpolation Polynomial on a Rectangular Basis with Equidistant Interpolation Points

In this section we study the form of the two-variable Newton interpolation polynomial in case where we have a rectangular basis with equidistant interpolation points. Let us consider the set $\tilde{S}_{\Delta}^{(k_1,k_2)}$ with

$$x_i = x_0 + i \cdot h_x \text{ and } y_j = y_0 + j \cdot h_y$$
$$0 \leq i \leq k_1 \text{ and } 0 \leq j \leq k_2$$

The points x_i (resp. y_j) are equidistant with distance h_x (resp. h_y).

Theorem 1: Define recursively the differences $d_{i,j}^{(k)}$:

$$d_{i,j}^{(k)} := \begin{cases} d_{i,j}^{(k-1)} - d_{i-1,j}^{(k-1)} & if \quad \begin{pmatrix} j < k \\ i \geq k \end{pmatrix} \\ d_{i,j}^{(k-1)} - d_{i,j-1}^{(k-1)} & if \quad \begin{pmatrix} i < k \\ j \geq k \end{pmatrix} \\ d_{i,j}^{(k-1)} + d_{i-1,j-1}^{(k-1)} - d_{i-1,j}^{(k-1)} - d_{i,j-1}^{(k-1)} & if \quad \begin{pmatrix} i \geq k \\ j \geq k \end{pmatrix} \\ d_{i,j}^{(k-1)} & if \quad \begin{pmatrix} i < k \\ j < k \end{pmatrix} \end{cases} \tag{5}$$

with initial values $d_{i,j}^{(0)} = p_{i,j}^{(0)} \equiv p(x_i, y_j)$ and $1 \leq k \leq \max\{k_1, k_2\}$. The relation between the divided differences $p_{i,j}^{(k)}$ and the differences $d_{i,j}^{(k)}$ are given below :

$$p_{i,j}^{(k)} = \frac{d_{i,j}^{(k)}}{i! \cdot h_x^i \cdot j! \cdot h_y^j}$$

where $k = \max\{i, j\}$.

Proof: In case $(j < k \ \wedge \ i \geq k)$ the statement

$$p_{i,j}^{(k)} = \frac{d_{i,j}^{(k)}}{k! \cdot h_x^k}$$

holds for $n = 1$:

$$p_{i,j}^{(1)} \overset{(4)}{=} \frac{p_{i,j}^{(0)} - p_{i-1,j}^{(0)}}{x_i - x_{i-1}} = \frac{d_{i,j}^{(0)} - d_{i-1,j}^{(0)}}{x_i - x_{i-1}} \overset{(5)}{=} \frac{d_{i,j}^{(1)}}{h_x}$$

Assume that the statement holds for $n = k - 1$:

$$p_{i,j}^{(k-1)} = \frac{d_{i,j}^{(k-1)}}{(k-1)! \cdot h_x^{k-1}} \tag{6}$$

Then the statement holds for $n = k$:

$$p_{i,j}^{(k)} \overset{(4)}{=} \frac{p_{i,j}^{(k-1)} - p_{i-1,j}^{(k-1)}}{x_i - x_{i-k}} \overset{(6)}{=}$$
$$= \frac{\frac{d_{i,j}^{(k-1)}}{(k-1)! \cdot h_x^{k-1}} - \frac{d_{i-1,j}^{(k-1)}}{(k-1)! \cdot h_x^{k-1}}}{k \cdot h_x} =$$
$$= \frac{d_{i,j}^{(k-1)} - d_{i-1,j}^{(k-1)}}{(k-1)! \cdot h_x^{k-1} \cdot k \cdot h_x} \overset{(5)}{=} \frac{d_{i,j}^{(k)}}{k! \cdot h_x^k}$$

Similarly, we can prove that in case where $(i < k \ \wedge \ j \geq k)$ we have that

$$p_{i,j}^{(k)} = \frac{p_{i,j}^{(k-1)} - p_{i,j-1}^{(k-1)}}{y_j - y_{j-k}} = \frac{d_{i,j}^{(k)}}{k! \cdot h_y^k}$$

In case where $(i \geq k \ \wedge \ j \geq k)$ the statement

$$p_{i,j}^{(k)} = \frac{d_{i,j}^{(k)}}{k! \cdot h_x^k \cdot k! \cdot h_y^k}$$

holds for $n = 1$:

$$p_{i,j}^{(1)} \overset{(4)}{=} \frac{p_{i,j}^{(0)} + p_{i-1,j-1}^{(0)} - p_{i-1,j}^{(0)} - p_{i,j-1}^{(0)}}{(x_i - x_{i-1})(y_j - y_{j-1})} =$$
$$= \frac{d_{i,j}^{(0)} + d_{i-1,j-1}^{(0)} - d_{i-1,j}^{(0)} - d_{i,j-1}^{(0)}}{(x_i - x_{i-1})(y_j - y_{j-1})} \overset{(5)}{=} \frac{d_{i,j}^{(1)}}{h_x \cdot h_y}$$

Assume that the statement holds for $n = k - 1$:

$$p_{i,j}^{(k-1)} = \frac{d_{i,j}^{(k-1)}}{(k-1)! \cdot h_x^{k-1} \cdot (k-1)! \cdot h_y^{k-1}} \tag{7}$$

Then the statement holds for $n = k$:

$$p_{i,j}^{(k)} \overset{(4)}{=} \frac{p_{i,j}^{(k-1)} + p_{i-1,j-1}^{(k-1)} - p_{i-1,j}^{(k-1)} - p_{i,j-1}^{(k-1)}}{(x_i - x_{i-k})(y_j - y_{j-k})} \overset{(7)}{=}$$
$$= \frac{\frac{d_{i,j}^{(k-1)} + d_{i-1,j-1}^{(k-1)} - d_{i-1,j}^{(k-1)} - d_{i,j-1}^{(k-1)}}{(k-1)! \cdot h_x^{k-1} \cdot (k-1)! \cdot h_y^{k-1}}}{k \cdot h_x \cdot k \cdot h_y} \overset{(5)}{=}$$
$$= \frac{d_{i,j}^{(k)}}{k! \cdot h_x^k \cdot k! \cdot h_y^k}$$

■

Note that in each step the computation of the first three parts in (5) can be done independently by taking the advantage of the capabilities of multi-core central processing units.
Therefore, the interpolation polynomial given in (1) can be rewritten as follows

$$\tilde{p}(x, y) = \tilde{X}_d^T \cdot \tilde{D} \cdot \tilde{Y}_d \tag{8}$$

where

$$\tilde{X}_d = \begin{pmatrix} 1 \\ \frac{x-x_0}{h_x} \\ \frac{(x-x_0)(x-x_1)}{2\cdot h_x^2} \\ \vdots \\ \frac{(x-x_0)(x-x_1)\cdots\left(x-x_{k_1-1}\right)}{k_1!\cdot h_x^{k_1}} \end{pmatrix}$$

$$\tilde{Y}_d = \begin{pmatrix} 1 \\ \frac{y-y_0}{h_y} \\ \frac{(y-y_0)(y-y_1)}{2\cdot h_y^2} \\ \vdots \\ \frac{(y-y_0)(y-y_1)\cdots\left(y-y_{k_2-1}\right)}{k_2!\cdot h_y^{k_2}} \end{pmatrix} \tag{9}$$

and in case where $k_1 > k_2$

$$\tilde{D} = \begin{pmatrix} d_{0,0}^{(0)} & d_{0,1}^{(1)} & d_{0,2}^{(2)} & \cdots & d_{0,k_2}^{(k_2)} \\ d_{1,0}^{(1)} & d_{1,1}^{(1)} & d_{1,2}^{(2)} & \cdots & d_{1,k_2}^{(k_2)} \\ d_{2,0}^{(2)} & d_{2,1}^{(2)} & d_{2,2}^{(2)} & \cdots & d_{2,k_2}^{(k_2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{k_2,0}^{(k_2)} & d_{k_2,1}^{(k_2)} & d_{k_2,2}^{(k_2)} & \cdots & d_{k_2,k_2}^{(k_2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{k_1,0}^{(k_1)} & d_{k_1,1}^{(k_1)} & d_{k_1,2}^{(k_1)} & \cdots & d_{k_1,k_2}^{(k_1)} \end{pmatrix} \tag{10}$$

By setting $\frac{x-x_0}{h_x} = \hat{x}$ and $\frac{y-y_0}{h_y} = \hat{y}$ in (8) and (9) respectivey we get

$$\hat{p}(\hat{x},\hat{y}) = \hat{X}_d^T \cdot \tilde{D} \cdot \hat{Y}_d \tag{11}$$

where

$$\hat{X}_d = \begin{pmatrix} 1 \\ \hat{x} \\ \frac{\hat{x}(\hat{x}-1)}{2} \\ \vdots \\ \frac{\hat{x}(\hat{x}-1)\cdots(\hat{x}-(k_1-1))}{k_1!} \end{pmatrix}$$

$$\hat{Y}_d = \begin{pmatrix} 1 \\ \hat{y} \\ \frac{\hat{y}(\hat{y}-1)}{2} \\ \vdots \\ \frac{\hat{y}(\hat{y}-1)\cdots(\hat{y}-(k_2-1))}{k_2!} \end{pmatrix} \tag{12}$$

In the special case where $x_0 = y_0 = 0$ and $h_x = h_y = 1$ we get $\hat{x} \equiv x$ and $\hat{y} \equiv y$, and thus, the polynomial (11) coincides with the polynomial (8). Based on above we can consider the following algorithm.

Algorithm 1: Construction of a Newton interpolating two-variable polynomial $\tilde{p}(x,y)$ (defined in (8)) for a two-variable polynomial $p(x,y)$ with known upper bound on the degree of x (resp. y) k_1 (resp. k_2). The number of required interpolation points is $(k_1+1)(k_2+1)$ and the set of these points is given by $\tilde{S}_{\Delta}^{(k_1,k_2)} = \{(x_i,y_i) \mid i = 0,1,...,k_1\ ,\ j = 0,1,...,k_2\}$ with $x_i = x_0 + i \cdot h_x$ and $y_j = y_0 + j \cdot h_y$.

Step 1: k=0. Based on the values of $p(x,y)$ at the set of points $\tilde{S}_{\Delta}^{(k_1,k_2)}$ create the zero order matrix of initial values

$$\tilde{D}_0 = \begin{pmatrix} d_{0,0}^{(0)} & d_{0,1}^{(0)} & d_{0,2}^{(0)} & \cdots & d_{0,k_2}^{(0)} \\ d_{1,0}^{(0)} & d_{1,1}^{(0)} & d_{1,2}^{(0)} & \cdots & d_{1,k_2}^{(0)} \\ d_{2,0}^{(0)} & d_{2,1}^{(0)} & d_{2,2}^{(0)} & \cdots & d_{2,k_2}^{(0)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{k_1,0}^{(0)} & d_{k_1,1}^{(0)} & d_{k_1,2}^{(0)} & \cdots & d_{k_1,k_2}^{(0)} \end{pmatrix} \tag{13}$$

where $d_{i,j}^{(0)} := p(x_i,y_j)$.

Step 2: For k=1 to $\tilde{n}$, where $\tilde{n} = \max\{k_1,k_2\}$ compute the $\tilde{n}^{th}$-order matrix of differences given by the recursive form (5)

$$\tilde{D} = \tilde{D}_{\tilde{n}} = \begin{pmatrix} d_{0,0}^{(\tilde{n})} & d_{0,1}^{(\tilde{n})} & d_{0,2}^{(\tilde{n})} & \cdots & d_{0,k_2}^{(\tilde{n})} \\ d_{1,0}^{(\tilde{n})} & d_{1,1}^{(\tilde{n})} & d_{1,2}^{(\tilde{n})} & \cdots & d_{1,k_2}^{(\tilde{n})} \\ d_{2,0}^{(\tilde{n})} & d_{2,1}^{(\tilde{n})} & d_{2,2}^{(\tilde{n})} & \cdots & d_{2,k_2}^{(\tilde{n})} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{k_1,0}^{(\tilde{n})} & d_{k_1,1}^{(\tilde{n})} & d_{k_1,2}^{(\tilde{n})} & \cdots & d_{k_1,k_2}^{(\tilde{n})} \end{pmatrix}$$

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

$$\tilde{D} = \tilde{D}_{\tilde{n}} = \begin{pmatrix} d_{0,0}^{(0)} & d_{0,1}^{(1)} & d_{0,2}^{(2)} & \cdots & d_{0,k_2}^{(k_2)} \\ d_{1,0}^{(1)} & d_{1,1}^{(1)} & d_{1,2}^{(2)} & \cdots & d_{1,k_2}^{(k_2)} \\ d_{2,0}^{(2)} & d_{2,1}^{(2)} & d_{2,2}^{(2)} & \cdots & d_{2,k_2}^{(k_2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{k_2,0}^{(k_2)} & d_{k_2,1}^{(k_2)} & d_{k_2,2}^{(k_2)} & \cdots & d_{k_2,k_2}^{(k_2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{k_1,0}^{(k_1)} & d_{k_1,1}^{(k_1)} & d_{k_1,2}^{(k_1)} & \cdots & d_{k_1,k_2}^{(k_1)} \end{pmatrix}$$

Step 3: The Newton interpolating polynomial is the following:

$$\tilde{p}(x,y) = \tilde{X}_d^T \cdot \tilde{D} \cdot \tilde{Y}_d$$

and $\tilde{X}_d, \tilde{Y}_d$ have been defined in (9).

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 3^{rd} order polynomial $\tilde{p}(x,y)$, we have to know its values at a set $\tilde{S}_{\Delta}^{(2,1)} = \{(x_i,y_i) \mid i = 0,1,2\ ,\ j = 0,1\}$ of $(k_1+1)(k_2+1) = 3 \times 2 = 6$ points with $x_i = x_0 + i \cdot h_x = 0 + i \cdot 1 \Rightarrow x_i = i$ and $y_j = y_0 + j \cdot h_y = 0 + j \cdot 1 \Rightarrow y_j = j$. By following the Algorithm 1 we get

Step 1: Create the zero order matrix of initial values

$$\tilde{D}_0 = \begin{pmatrix} d_{0,0}^{(0)} = -2 & d_{0,1}^{(0)} = -2 \\ d_{1,0}^{(0)} = -4 & d_{1,1}^{(0)} = 8 \\ d_{2,0}^{(0)} = -6 & d_{2,1}^{(0)} = 48 \end{pmatrix}$$

Step 2: For k=1 to $\tilde{n}$, where $\tilde{n} = \max\{k_1,k_2\} = \max\{2,1\} = 2$, we have

For k=1

$$\tilde{D}_1 = \begin{pmatrix} \mathbf{d_{0,0}^{(0)} = -2} & d_{0,1}^{(1)} = 0 \\ d_{1,0}^{(1)} = -2 & d_{1,1}^{(1)} = 12 \\ d_{2,0}^{(1)} = -2 & d_{2,1}^{(1)} = 42 \end{pmatrix}$$

For k=2

$$\tilde{D}_2 = \begin{pmatrix} \mathbf{d_{0,0}^{(0)} = -2} & \mathbf{d_{0,1}^{(1)} = 0} \\ \mathbf{d_{1,0}^{(1)} = -2} & \mathbf{d_{1,1}^{(1)} = 12} \\ d_{2,0}^{(2)} = 0 & d_{2,1}^{(2)} = 30 \end{pmatrix}$$

Step 3: The Newton interpolating polynomial is

$$p\,(x,y) = \tilde{p}(x,y) = \tilde{X}_d^T \cdot \tilde{D} \cdot \tilde{Y}_d = \\ = -2 - 2x - 3xy + 15x^2y$$

where $\tilde{D} = \tilde{D}_2$ and

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

III. An Alternative Algorithm in Terms of Matrix Multiplications

As we have seen in the previous section, in case where we have a rectangular basis with equidistant interpolation points, the two-variable Newton interpolating polynomial is given by $\tilde{p}(x,y) = \tilde{X}_d^T \cdot \tilde{D} \cdot \tilde{Y}_d$ with $\tilde{X}_d$, $\tilde{Y}_d$ and $\tilde{D}$ defined in (9) and (10) respectively. The elements of the initial matrix $\tilde{D}_0$ are given by the values of the polynomial $p(x,y)$ on the set $\tilde{S}_{\Delta}^{(k_1,k_2)}$ i.e. $d_{i,j}^{(0)} = p(x_i, y_j)$. Thus

$$\tilde{D}_0 = \begin{pmatrix} d_{0,0}^{(0)} & d_{0,1}^{(0)} & d_{0,2}^{(0)} & \cdots & d_{0,k_2}^{(0)} \\ d_{1,0}^{(0)} & d_{1,1}^{(0)} & d_{1,2}^{(0)} & \cdots & d_{1,k_2}^{(0)} \\ d_{2,0}^{(0)} & d_{2,1}^{(0)} & d_{2,2}^{(0)} & \cdots & d_{2,k_2}^{(0)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{k_1,0}^{(0)} & d_{k_1,1}^{(0)} & d_{k_1,2}^{(0)} & \cdots & d_{k_1,k_2}^{(0)} \end{pmatrix}$$

According to the recursive type (5) the matrix $\tilde{D}_k$ with $k < k_2 < k_1$ is given by

$$\tilde{D}_k = \begin{pmatrix} d_{0,0}^{(0)} & d_{0,1}^{(1)} & d_{0,2}^{(2)} & \cdots & d_{0,k}^{(k)} & \cdots & d_{0,k_2}^{(k)} \\ d_{1,0}^{(1)} & d_{1,1}^{(1)} & d_{1,2}^{(2)} & \cdots & d_{1,k}^{(k)} & \cdots & d_{1,k_2}^{(k)} \\ d_{2,0}^{(2)} & d_{2,1}^{(2)} & d_{2,2}^{(2)} & \cdots & d_{2,k}^{(k)} & \cdots & d_{2,k_2}^{(k)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ d_{k,0}^{(k)} & d_{k,1}^{(k)} & d_{k,2}^{(k)} & \cdots & d_{k,k}^{(k)} & \cdots & d_{k,k_2}^{(k)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ d_{k_1,0}^{(k)} & d_{k_1,1}^{(k)} & d_{k_1,2}^{(k)} & \cdots & d_{k_1,k}^{(k)} & \cdots & d_{k_1,k_2}^{(k)} \end{pmatrix}$$

We define the square matrix $B_{k,m} \in \mathbb{K}^{(m \times m)}$ where $\mathbb{K} = \{-1, 0, 1\}$ with elements given by

$$b_{i,j} = \begin{cases} 1 & i = j \\ -1 & j = i - 1 \wedge i > k \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

For example, if we consider $k_1 = 3$ and $k_2 = 2$ then for $k = 1$ we have two square martices B with dimensions $k_1 + 1 = 4$ and $k_2 + 1 = 3$ respectively

$$B_{1,4} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix}$$

and respectively

$$B_{1,3} = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix}$$

Similarly, for $k = 2$ we have

$$B_{2,4} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix}$$

$$B_{2,3} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix}$$

and for $k = 3$

$$B_{3,4} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix}$$

$$B_{3,3} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

If

$$\tilde{D}_0 = \begin{pmatrix} d_{0,0}^{(0)} & d_{0,1}^{(0)} & d_{0,2}^{(0)} \\ d_{1,0}^{(0)} & d_{1,1}^{(0)} & d_{1,2}^{(0)} \\ d_{2,0}^{(0)} & d_{2,1}^{(0)} & d_{2,2}^{(0)} \\ d_{3,0}^{(0)} & d_{3,1}^{(0)} & d_{3,2}^{(0)} \end{pmatrix}$$

then the multiplication $B_{1,4} \cdot \tilde{D}_0 \cdot (B_{1,3})^T$ gives

$$\begin{pmatrix} d_{0,0}^{(0)} & d_{0,1}^{(0)} - d_{0,0}^{(0)} & d_{0,2}^{(0)} - d_{0,1}^{(0)} \\ d_{1,0}^{(0)} - d_{0,0}^{(0)} & d_{1,1}^{(0)} + d_{0,0}^{(0)} - d_{1,0}^{(0)} - d_{0,1}^{(0)} & d_{1,2}^{(0)} + d_{0,1}^{(0)} - d_{1,1}^{(0)} - d_{0,2}^{(0)} \\ d_{2,0}^{(0)} - d_{1,0}^{(0)} & d_{2,1}^{(0)} + d_{1,0}^{(0)} - d_{2,0}^{(0)} - d_{1,1}^{(0)} & d_{2,2}^{(0)} + d_{1,1}^{(0)} - d_{2,1}^{(0)} - d_{1,2}^{(0)} \\ d_{3,0}^{(0)} - d_{2,0}^{(0)} & d_{3,1}^{(0)} + d_{2,0}^{(0)} - d_{3,0}^{(0)} - d_{2,1}^{(0)} & d_{3,2}^{(0)} + d_{2,1}^{(0)} - d_{3,1}^{(0)} - d_{2,2}^{(0)} \end{pmatrix}$$

or equivalently

$$\tilde{D}_1 = \begin{pmatrix} d_{0,0}^{(0)} & d_{0,1}^{(1)} & d_{0,2}^{(1)} \\ d_{1,0}^{(1)} & d_{1,1}^{(1)} & d_{1,2}^{(1)} \\ d_{2,0}^{(1)} & d_{2,1}^{(1)} & d_{2,2}^{(1)} \\ d_{3,0}^{(1)} & d_{3,1}^{(1)} & d_{3,2}^{(1)} \end{pmatrix} = B_{1,4} \cdot \tilde{D}_0 \cdot (B_{1,3})^T$$

Similarly, the multiplication $B_{2,4} \cdot \tilde{D}_1 \cdot (B_{2,3})^T$ gives

$$\begin{pmatrix} d_{0,0}^{(0)} & d_{0,1}^{(1)} & d_{0,2}^{(1)} - d_{0,1}^{(1)} \\ d_{1,0}^{(1)} & d_{1,1}^{(1)} & d_{1,2}^{(1)} - d_{1,1}^{(1)} \\ d_{2,0}^{(1)} - d_{1,0}^{(1)} & d_{2,1}^{(1)} - d_{1,1}^{(0)} & d_{2,2}^{(1)} + d_{1,1}^{(1)} - d_{2,1}^{(1)} - d_{1,2}^{(1)} \\ d_{3,0}^{(1)} - d_{2,0}^{(1)} & d_{3,1}^{(1)} - d_{2,1}^{(0)} & d_{3,2}^{(1)} + d_{2,1}^{(1)} - d_{3,1}^{(1)} - d_{2,2}^{(1)} \end{pmatrix}$$

or equivalently

$$\tilde{D}_2 = \begin{pmatrix} d_{0,0}^{(0)} & d_{0,1}^{(1)} & d_{0,2}^{(2)} \\ d_{1,0}^{(1)} & d_{1,1}^{(1)} & d_{1,2}^{(2)} \\ d_{2,0}^{(2)} & d_{2,1}^{(2)} & d_{2,2}^{(2)} \\ d_{3,0}^{(2)} & d_{3,1}^{(2)} & d_{3,2}^{(2)} \end{pmatrix} = B_{2,4} \cdot \tilde{D}_1 \cdot (B_{2,3})^T$$

Finally, the multiplication $B_{3,4} \cdot \tilde{D}_2 \cdot (B_{3,3})^T$ gives

$$\begin{pmatrix} d_{0,0}^{(0)} & d_{0,1}^{(1)} & d_{0,2}^{(2)} \\ d_{1,0}^{(1)} & d_{1,1}^{(1)} & d_{1,2}^{(2)} \\ d_{2,0}^{(2)} & d_{2,1}^{(2)} & d_{2,2}^{(2)} \\ d_{3,0}^{(2)} - d_{2,0}^{(2)} & d_{3,1}^{(2)} - d_{2,1}^{(2)} & d_{3,2}^{(2)} - d_{2,2}^{(2)} \end{pmatrix}$$

or equivalently

$$\tilde{D} = \tilde{D}_3 = \begin{pmatrix} d_{0,0}^{(0)} & d_{0,1}^{(1)} & d_{0,2}^{(2)} \\ d_{1,0}^{(1)} & d_{1,1}^{(1)} & d_{1,2}^{(2)} \\ d_{2,0}^{(2)} & d_{2,1}^{(2)} & d_{2,2}^{(2)} \\ d_{3,0}^{(3)} & d_{3,1}^{(3)} & d_{3,2}^{(3)} \end{pmatrix} = B_{3,4} \cdot \tilde{D}_2 \cdot (B_{3,3})^T$$

Therefore, we can calculate the final matrix $\tilde{D}$ of differences by using only matrix manipulation. Based on this observation we can provide a simplified version of Algorithm 1.

Algorithm 2: Construction of a Newton interpolating two-variable polynomial $\tilde{p}(x,y)$ (defined in (8)) for a two-variable polynomial $p(x,y)$ with known upper bound on the degree of x (resp. y) k_1 (resp. k_2).The number of required interpolation points is $(k_1+1)(k_2+1)$ and the set of these points is given by $\tilde{S}_{\Delta}^{(k_1,k_2)} = \{(x_i, y_i) \mid i = 0,1,...,k_1\ ,\ j = 0,1,...,k_2\}$ with $x_i = x_0 + i \cdot h_x$ and $y_j = y_0 + j \cdot h_y$.

Step 1: k=0. Based on the values of $p(x,y)$ at the set of points $\tilde{S}_{\Delta}^{(k_1,k_2)}$ create the zero order table of initial values $\tilde{D}_0$ which has been defined in (13).

Step 2: For k=1 to $\tilde{n}$, where $\tilde{n} = \max\{k_1, k_2\}$ compute the $\tilde{n}^{th}$-order matrix of differences with matrix multiplications by the recursive form

$$\tilde{D}_k = B_{k,k_1+1} \cdot \tilde{D}_{k-1} \cdot (B_{k,k_2+1})^T \quad k = 1,2,\ldots,\tilde{n} \quad (15)$$

where $B_{i,j}$ have been defined in (14).

Step 3: The Newton interpolating polynomial is the following:

$$\tilde{p}(x,y) = \tilde{X}_d^T \cdot \tilde{D} \cdot \tilde{Y}_d$$

where $\tilde{X}_d, \tilde{Y}_d$ have been defined in (9) and $\tilde{D} = \tilde{D}_{\tilde{n}}$.

The above algorithm can be easily implemented in programming environments that support matrix manipulations, such as, Matlab, Mathematica etc.

Example 2: Consider the Example 1. By following the Algorithm 2

Step 1: We create the table of initial values

$$\tilde{D}_0 = \begin{pmatrix} -2 & -2 \\ -4 & 8 \\ -6 & 48 \end{pmatrix}$$

Step 2: For k=1 to $\tilde{n}$, where $\tilde{n} = \max\{k_1, k_2\} = \max\{2,1\} = 2$, compute the k^{th} order matrix of differences according to the recursive form (15).

For k=1 the 1^{st} order table of differences is

$$\tilde{D}_1 = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \cdot \begin{pmatrix} -2 & -2 \\ -4 & 8 \\ -6 & 48 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} = \\ = \begin{pmatrix} -2 & 0 \\ -2 & 12 \\ -2 & 42 \end{pmatrix}$$

For k=2 the 2^{nd} order table of differences is

$$\tilde{D}_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \cdot \begin{pmatrix} -2 & 0 \\ -2 & 12 \\ -2 & 42 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \\ = \begin{pmatrix} -2 & 0 \\ -2 & 12 \\ 0 & 30 \end{pmatrix}$$

Step 3: The interpolating polynomial is:

$$p(x,y) = \tilde{p}(x,y) = \tilde{X}_d^T \cdot \tilde{D} \cdot \tilde{Y}_d = \\ = -2 - 2x - 3xy + 15x^2y$$

where $\tilde{D} = \tilde{D}_2$ and

$$\tilde{X}_d^T = \begin{pmatrix} 1 \\ x \\ \frac{x(x-1)}{2} \end{pmatrix}; \tilde{Y}_d = \begin{pmatrix} 1 \\ y \end{pmatrix}$$

IV. On the Complexity of the Proposed Algorithms

According to [1], the total number of computations in Algorithm 1, in case where we use a rectangular basis and random interpolation points with $k_1 > k_2$ is given by

$$\tilde{F}(k_1,k_2) = \\ = \left(\frac{k_1^2k_2}{2} + \frac{k_1^2}{2} - \frac{k_1k_2^2}{2} + \frac{k_1}{2} + \frac{k_2^3}{3} + \frac{k_2^2}{2} + \frac{k_2}{6}\right)a + \\ + \left(\frac{k_2}{6} + \frac{k_1k_2}{2} + \frac{k_1k_2^2}{2} - \frac{k_2^3}{6}\right)b$$

where a (resp. b) is the computational cost of the first and second part (resp. third) of the recursive type (4). In the special case where we have equidistance points the number of computations is the same with the main difference that a and b corresponds to less computational cost since the first and second part (resp. third) in (5) requires less operations than the ones in (4) i.e. just one addition in the first and second part and three additions on the third part (no multiplications at all). By assuming that $k1 = k2$ (worst case) and $3a = b$ we have

$$\tilde{F}(k_1,k_2) = \left(\frac{4k_1^3}{3} + \frac{5k_1^2}{2} + \frac{7k_1}{6}\right)a$$

and therefore the complexity is $O(k_1^3)$.

In Algorithm 2 we can rewrite the recursive type (15) as follows

$$C = B_{k,k_1+1} \cdot \tilde{D}_{k-1} \quad k = 1,2,\ldots,\tilde{n}$$
$$\tilde{D}_k = C \cdot (B_{k,k_2})^T \quad k = 1,2,\ldots,\tilde{n}$$

with final matrix $\tilde{D} = \tilde{D}_{k_1}$ if $\tilde{n} = k_1$. For each iteration in Step 2 of this algorithm, we need two matrix multiplications. Therefore, the number of operations needed for each iteration is $2(k_1+1)^3$ multiplications and $2k_1^3$ additions. Thus, for all iterations the complexity is $O(k_1^4)$. We can reduce this complexity by using parallel programming. According [15], [16] we can partition the matrix multiplication in $(k_1+1)^3$ fine-grain tasks and with 1-D agglomeration the total calculation time will become

$$T_p = t_c \frac{(k_1+1)^3}{p} + T_{comm}$$

where t_c is the time for execution of multiplication or addition, p is the number of processors and T_{comm} is the required time for communication between CPUs and memory. Since we have two matrix multiplications, the total calculation time for all iterations will become

$$T_p = 2\left(t_c \frac{(k_1+1)^3}{p} + T_{comm}\right) k_1$$

Since the matrices $B_{i,j}$ are triangular with elements in $\mathbb{K} = \{-1, 0, 1\}$ we actually don't need any multiplication. By using this observation we can further reduce the calculation time and the corresponding complexity of Algorithm 2.

V. Conclusions

A special case, of the two-variable Newton difference algorithm [1], is proposed in this work for a rectangular basis with equidistant interpolation points. The proposed algorithm is presented by two different ways, with the second one based on matrix multiplications. The complexity of both algorithms has proved that is better than the ones given in [1], since it is based only on additions. The second algorithm is very simple for implementation in any software which support matrix manipulations i.e. Matlab, and its complexity can be further reduced by using parallel computations. The proposed algorithms can easily be extended to polynomials with more than two variables. The proposed algorithms can found applications in control theory for 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. Further work, has to be done: a) in the implementation of Algorithm 2 in a parallel programming environment, and b) on the reduction of operations used in the matrix multiplications due to the special form of the multiplying matrices.

VI. Acknowledgments

This work was supported by the Research Committee of the Serres Institute of Education and Technology.

References

[1] D. N. Varsamis and N. P. Karampetakis, "On the newton multivariate polynomial interpolation with applications," in *2011 7th International Workshop on Multidimensional (nD) Systems (nDs), Poitiers, France*, 2011.

[2] P. J. Olver, "On multivariate interpolation," *Studies in Applied Mathematics*, vol. 116, pp. 201–240, 2006.

[3] T. Sauer and Y. Xu, "On multivariate lagrange interpolation," *Mathematics of Computation*, vol. 64, pp. 1147–1170, 1995.

[4] G. M. Phillips, *Interpolation and approximation by polynomials*. Springer-Verlag, 2003.

[5] M. Gasca and J. J. Martinez, "Bivariate hermite-birkhof interpolation and vandermonde determinants," *Numerical Algorithms*, vol. 2, pp. 193–199, 1992.

[6] R. D. Neidinger, "Multivariable interpolating polynomials in newton forms," in *Joint Mathematics Meetings 2009, Washington D.C., January 5-8*, 2009.

[7] J. D. Lipson, "The fast fourier transform, its role as an algebraic algorithm," in *Proceedings of the ACM Annual Conference/Annual Meeting*, 1976, pp. 436–441.

[8] L. E. Paccagnella and G. L. Pierobon, "Fft calculation of a determinantal polynomial," *IEEE Transactions on Automatic Control*, vol. 29, no. 3, pp. 401–402, 1976.

[9] N. Karampetakis and A. Evripidou, "On the computation of the inverse of a two-variable polynomial matrix by interpolation," *Multidimensional Systems and Signal Processing*, vol. 23, pp. 97–118, 2012.

[10] E. G. Antoniou, "Transfer function computation for generalized ndimensional systems," *Journal of the Franklin Institute*, vol. 338, pp. 83–90, 2001.

[11] M. Hromcik and M. Sebek, "Fast fourier transform and linear polynomial matrix equations," in *Proceedings of the 1rst IFAC Workshop on Systems Structure and Control, Prague, Czech Republic*, 2001.

[12] N. P. Karampetakis and S. Vologiannidis, "Dft calculation of the generalized and drazin inverse of a polynomial matrix," *Applied Mathematics and Computation*, vol. 143, pp. 501–521, 2003.

[13] S. Vologiannidis and N. P. Karampetakis, "Inverses of multivariable polynomial matrices by discrete fourier transforms," *Multidimensional Systems and Signal Processing*, vol. 15, pp. 341–361, 2004.

[14] A. Schuster and P. Hippe, "Inversion of polynomial matrices by interpolation," *IEEE Transactions on Automatic Control*, vol. 37, pp. 363–365, 1992.

[15] J. W. Demmel, M. T. Heath, and H. A. van der Vorst, "Parallel numerical linear algebra," *Acta Numerica*, vol. 2, pp. 111–197, 1993.

[16] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, 1989.