

Multivariable Interpolating Polynomials in Newton Forms

Richard D. Neidinger
Davidson College

February 24, 2009

1 Introduction

Undergraduate Linear Algebra or Numerical Analysis courses often discuss finding a polynomial that goes through specified points, agreeing with given values at a set of one-variable nodes. We show how multivariable algorithms that are not well known can be introduced at this level. The multivariable case is more interesting and involved since a unique solution may not exist. The theme will be finding a good basis for the polynomial space relative to the given nodes for interpolation. In one variable, most numerical analysis texts introduce the classic Newton form basis with an efficient divided-difference algorithm to compute the coefficients of the interpolating polynomial. The multivariable generalization has been known since before computers ([10, p. 204], [4, p. 267], and [3, p. 296]) but with scarce modern exposition as done here. However, this classic generalization only works on nodes with special structure. To handle arbitrary nodes, we use variations on Gaussian elimination to explain more recent algorithms ([8] and [7]) that form different generalizations of the Newton basis. For brevity and clarity, we use two-variable quadratic or cubic examples, though algorithms will be specified and justified for n variables to degree d .

The basic problem is: does there exist, and how do you find, a unique polynomial

$$p(x, y) = c_{00} + c_{10}x + c_{01}y + c_{20}x^2 + c_{11}xy + c_{02}y^2$$

with specified function values at six distinct nodes? The matrix formulation of this problem is a nice exercise in beginning linear algebra, since students must change perspective to where x, y are numerical values and c 's are unknowns. We use multi-index subscripts corresponding to the powers of the variables in terms of the polynomial. In general, these are n -tuples of non-negative integers, where we simply write 11 to stand for $(1, 1)$. The six nodes and function values are similarly indexed, where (x_{10}, y_{10}, f_{10}) is just the second triple of any three

real values whatsoever. The linear system becomes

$$\begin{bmatrix} 1 & x_{00} & y_{00} & x_{00}^2 & x_{00}y_{00} & y_{00}^2 \\ 1 & x_{10} & y_{10} & x_{10}^2 & x_{10}y_{10} & y_{10}^2 \\ 1 & x_{01} & y_{01} & x_{01}^2 & x_{01}y_{01} & y_{01}^2 \\ 1 & x_{20} & y_{20} & x_{20}^2 & x_{20}y_{20} & y_{20}^2 \\ 1 & x_{11} & y_{11} & x_{11}^2 & x_{11}y_{11} & y_{11}^2 \\ 1 & x_{02} & y_{02} & x_{02}^2 & x_{02}y_{02} & y_{02}^2 \end{bmatrix} \begin{bmatrix} c_{00} \\ c_{10} \\ c_{01} \\ c_{20} \\ c_{11} \\ c_{02} \end{bmatrix} = \begin{bmatrix} f_{00} \\ f_{10} \\ f_{01} \\ f_{20} \\ f_{11} \\ f_{02} \end{bmatrix} \quad (1)$$

abbreviated to $V^T \mathbf{c} = \mathbf{f}$, where V is a generalized Vandermonde matrix. (Though not universal, orienting V with a first row of ones will be useful for our purposes.) The existence and uniqueness problem asks if V is invertible. A polynomial of degree d in \mathbb{R}^n would require values at $\binom{n+d}{d}$ nodes. In the one-variable case with distinct x nodes, V is always invertible, as can be seen in several interesting linear algebra exercises. In the multivariable case, distinct nodes are not enough; V may or may not be invertible. This leads to many different types of results about multivariable interpolation, describing geometry, subspaces, algorithms, etc. ([5, Section 6.10], [2]).

Three examples in Figure 1 show different situations that will guide our discussion. Consider specified function values at each of these sets of nodes. In (a), if there exists a quadratic polynomial with specified function values on

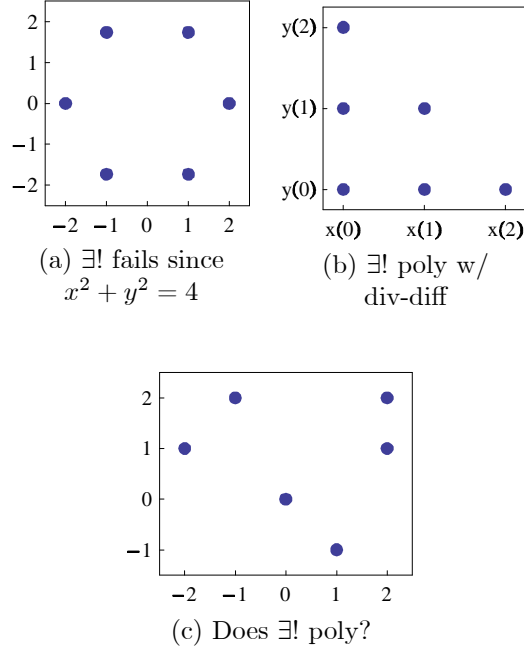


Figure 1: Impossible Circle, Easy Corner, and Generic Example

this circle of nodes, then adding $x^2 + y^2 - 4$ will produce another polynomial solution. Thus, the linear system must either have no solutions or infinitely many. Uniqueness is impossible and the matrix V is singular. This contrasts with (b), so Section 2 first investigates why a unique polynomial always exists for function values on such a corner of nodes. In fact, Section 3 shows it is easy to find with a divided-difference algorithm. The key is a "Newton" basis for the polynomial space, where interpolation on each node could be handled incrementally, with a new basis term for each new node. For an arbitrary set of nodes, such as (c), it is usually not obvious whether or not a unique interpolating polynomial exists. The algorithm in Section 4 will either create a "Newton" basis or show that the nodes lie on a level surface of a polynomial, a conic section in the case of six nodes. We use matrix operations similar to Gaussian elimination, which work just as well for higher n or d where visualization is difficult. Returning to the circle of nodes (a), Section 5 shows that the algorithm can produce a polynomial subspace of minimal degree where unique polynomial interpolation does succeed. Finally, Section 6 uses Gaussian elimination in a block matrix form to produce a "Newton" basis that is algebraically simpler.

2 Newton polynomials for a corner of nodes

To describe a corner of nodes, as in Figure 1(b) or Figure 2, we use functional notation for distinct *tick marks* on each axis, as in $x(0), x(1), x(2), \dots$, which need not be ordered or spaced in any particular way. In fact, the coordinate axes need not be rectangular. Grid points are then given by a multi-index which we abbreviate as in $\mathbf{x}(10) = (x(1), y(0))$, which is more specific than arbitrary (x_{10}, y_{10}) in (1). A single subscript is reserved for the n coordinate variable names, for example $\mathbf{x}(0213) = (x_1(0), x_2(2), x_3(1), x_4(3))$. In general, a *multi-index* $\boldsymbol{\lambda} \in \mathbb{N}_0^n$, i.e., $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \dots, \lambda_n)$ where each $\lambda_i \in \{0, 1, 2, 3, \dots\}$. Define $|\boldsymbol{\lambda}| = \lambda_1 + \lambda_2 + \dots + \lambda_n$ to be the *order* of the multi-index. A *corner* of nodes is $\{\mathbf{x}(\boldsymbol{\lambda}) : |\boldsymbol{\lambda}| \leq d\}$. For a given set of tick marks on each axis, define

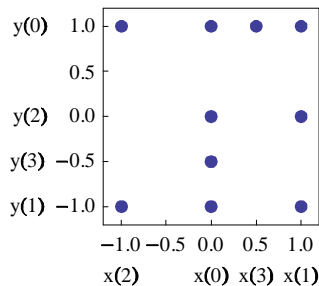


Figure 2: Example corner of nodes

the *classic Newton polynomial*, corresponding to each multi-index $\boldsymbol{\lambda} \in \mathbb{N}_0^n$, by

$$q_{\boldsymbol{\lambda}}(\mathbf{x}) = \prod_{m=1}^n \prod_{i=0}^{\lambda_m-1} (x_m - x_m(i)).$$

Nodes in Figure 1(b) give rise to the classic Newton polynomial basis for quadratic polynomials:

$$\begin{aligned} q_{00}(x, y) &= 1, \\ q_{10}(x, y) &= x - x(0), \\ q_{01}(x, y) &= y - y(0), \\ q_{20}(x, y) &= (x - x(0))(x - x(1)), \\ q_{11}(x, y) &= (x - x(0))(y - y(0)), \\ q_{02}(x, y) &= (y - y(0))(y - y(1)). \end{aligned} \tag{2}$$

Clearly, $\text{Span}\{q_{00}, q_{10}, q_{01}, q_{20}, q_{11}, q_{02}\} = \text{Span}\{1, x, y, x^2, xy, y^2\}$. To find an interpolating polynomial, the coefficients of the Newton basis are given by a simple triangular linear system. The table of values of this Newton basis at the corner nodes is

	$\mathbf{x}(00)$	$\mathbf{x}(10)$	$\mathbf{x}(01)$	$\mathbf{x}(20)$	$\mathbf{x}(11)$	$\mathbf{x}(02)$
q_{00}	1	1	1	1	1	1
q_{10}	0	Δ	0	Δ	Δ	0
q_{01}	0	0	Δ	0	Δ	Δ
q_{20}	0	0	0	$\Delta\Delta$	0	0
q_{11}	0	0	0	0	$\Delta\Delta$	0
q_{02}	0	0	0	0	0	$\Delta\Delta$

(3)

where each Δ denotes a non-zero difference between tick mark coordinates. For example, $q_{10}(\mathbf{x}(20)) = x(2) - x(0)$ and $q_{11}(\mathbf{x}(11)) = (x(1) - x(0))(y(1) - y(0))$. Let (3) define matrix N , corresponding to the nodes. Then $N^T \mathbf{a} = \mathbf{f}$ is the system giving coefficients \mathbf{a} of the Newton basis to achieve desired function values \mathbf{f} at the nodes (compare to (1)). Clearly, a simple forward-substitution will solve for interpolating polynomial coefficients and an even simpler algorithm can be designed for this structure. Indeed, the divided-difference algorithm (next section) directly manipulates the vector of values \mathbf{f} and ends with \mathbf{a} , using memory only the square root of the size of N .

The structure in (3) holds for any set of grid points and the corresponding Newton polynomials. In general,

$$q_{\boldsymbol{\lambda}}(\mathbf{x}(\boldsymbol{\mu})) \neq 0 \text{ if and only if } \boldsymbol{\lambda} \leq \boldsymbol{\mu} \text{ coordinate-wise.} \tag{4}$$

To get the block structure in (3), we list in increasing order of the multi-index, corresponding to degree of the polynomial and associating an order with each node. Multi-indices of equal order are grouped by partition lines and listed

reverse lexically (as is typical in listing multivariable powers, though this is not important). Then, each q_{λ} is a polynomial that

- (a) is zero at nodes of lower order,
 - (b) is nonzero at node $\mathbf{x}(\lambda)$ and zero at other nodes of the same order, and
 - (c) has degree $|\lambda|$ and only one term of highest degree.
- (5)

This means that a lower-order interpolating polynomial will not change but be augmented by new terms as higher-order nodes are added. Such nesting of interpolating polynomials is an important feature of the Newton basis that will be explicitly observed in the next section. In later sections, we will associate arbitrary (not grid point) nodes with multi-indices of order $\leq d$ and attempt to find a generalized polynomial basis that preserves these properties as much as possible.

Actually, the properties of the Newton polynomials apply to any grid points, even if not a corner of nodes. When grid points are given by some multi-indices and listed by order, Newton polynomial values will make an invertible matrix. Looking at it another way, taking a "full corner" matrix such as (3) and deleting the same column and row any number of times will not change the triangular shape with nonzero diagonal. We deduce the following.

Theorem 1 (General Existence and Uniqueness) *Let $I \subseteq \mathbb{N}_0^n$ be any subset of multi-indices with corresponding points $\{(\mathbf{x}(\lambda), f_{\lambda}) : \lambda \in I\}$ from a grid. Then $\{q_{\lambda} : \lambda \in I\}$ is linearly independent and there exists a unique interpolating polynomial in $\text{Span}\{q_{\lambda} : \lambda \in I\}$.*

Nodes determine the span. The special property of a corner of nodes is that the $\text{Span}\{q_{\lambda} : |\lambda| \leq d\}$ is the space of polynomials of degree $\leq d$. This actually provides design direction in applications where one chooses nodes for interpolation, such as in the application to automatic differentiation [6] that motivated this study. Another important case is an n -dimensional box $I = \{\lambda : \mathbf{0} \leq \lambda \leq \beta \text{ coordinate-wise}\}$ for some fixed multi-index β ; then $\text{Span}\{q_{\lambda} : \lambda \in I\} = \text{Span}\{x_1^{\lambda_1} x_2^{\lambda_2} \dots x_n^{\lambda_n} : \mathbf{0} \leq \lambda \leq \beta\}$ is the tensor-product space. Such boxes will be featured in proving the divided-difference algorithm. All grid point results of this section and the next are sometimes referred to simply as the tensor-product case in modern surveys of interpolation [2]. Of course, any nodes whatsoever can specify tick marks and form some set of grid points, though the corresponding span may not be of interest. The circle of six nodes from Figure 1(a) can be viewed as grid points using tick marks $x(0) = -1$, $x(1) = 1$, $y(0) = -\sqrt{3}$, $y(1) = \sqrt{3}$ (four nodes in this rectangle) and $y(2) = 0$, $x(2) = -2$, $x(3) = 2$ (two more nodes on x-axis). For these grid points, the associated multi-indices and classic Newton polynomials are q_{00} , q_{10} , q_{01} , q_{11} as in (2) plus $q_{22}(x, y) = (x - x(0))(x - x(1))(y - y(0))(y - y(1))$, and $q_{32}(x, y) = (x - x(2))q_{22}(x, y)$. So, for values at these nodes, there is a unique interpolating polynomial in the Span of these q 's which equals $\text{Span}\{1, x, y, xy, q_{22}, xq_{22}\}$.

The terminology of [1] would say that the pairing of these nodes and this span is *correct*. However, this subspace of polynomials of degree ≤ 5 may not be useful, since degree 5 is enough to interpolate any six nodes, even if they lie on a straight line (where degree 5 is necessary). Later we will show how to extend the method of Section 4 to find a subspace of polynomials of minimal degree 3 that is correct for the circle of nodes.

3 The divided-difference algorithm

Before justifying the procedure, we start by computing an interpolating polynomial for values on the nodes in Figure 2. The desired function values are given in Table 1(a) and result in coefficients of the classic Newton basis found in Table 1(d). For example, multi-index $(i, j) = (2, 1)$ corresponds to $(x, y) = (-1, -1)$

\underline{j}	$\underline{y(j)}$	(a) function values				\underline{j}	$\underline{y(j)}$	(b) after pass 1			
3	-0.5	2.75				3	-0.5	0.5			
2	0	3	3			2	0	-6	0		
1	-1	9	10	16		1	-1	-2	1	-3	
0	1	5	8	2	4.25	0	1	5	3	3	1.5
$x(i) :$		0	1	-1	0.5	$x(i) :$		0	1	-1	0.5
$i :$		0	1	2	3	$i :$		0	1	2	3
\underline{j}	$\underline{y(j)}$	(c) after pass 2				\underline{j}	$\underline{y(j)}$	(d) after pass 3			
3	-0.5	13				3	-0.5	-6			
2	0	4	-1			2	0	4	2		
1	-1	-2	1	4		1	-1	-2	1	-2	
0	1	5	3	0	3	0	1	5	3	0	6
$x(i) :$		0	1	-1	0.5	$x(i) :$		0	1	-1	0.5
$i :$		0	1	2	3	$i :$		0	1	2	3

Table 1: Pass k computes a divided-difference as in (6) for each $i + j \geq k$

where we want function value 16. The value at $(2, 1)$ will be changed in each of $3(= 2 + 1)$ passes through the array; the first 2 passes use the difference with the value to the left and then 1 pass uses the difference with the value below. Differences are divided by corresponding differences in tick-mark values over a range that steps down one further each time a direction is repeated. So values at multi-index $(2, 1)$ are computed by $\frac{16-10}{-1-1} = -3$, $\frac{-3-1}{-1-0} = 4$, and $\frac{4-0}{-1-1} = -2$. We can use array values $A[(i, j)]$ that are overwritten on each pass, as long as entries are computed from higher to lower order of the multi-index, so that needed values are used before they are overwritten. In this example, we only need the 10 array values with $i + j \leq 3$, as opposed to the 100 entries (including many zeros) that would be in a matrix formulation as in (3). In general, each

divided-difference is given by

$$\text{pass } k \text{ computation of } A[(i, j)] = \begin{cases} \frac{A[(i, j)] - A[(i-1, j)]}{x^{(i)} - x^{(i-k)}} & \text{if } k \leq i, \\ \frac{A[(i, j)] - A[(i, j-1)]}{y^{(j)} - y^{(i+j-k)}} & \text{if } i < k \leq i + j, \\ A[(i, j)] \text{ (done)} & \text{if } k > i + j. \end{cases} \quad (6)$$

The result is a nested set of interpolating polynomials, reading coefficients in Table 1 up the finished diagonal after each pass:

$$\begin{aligned} p_1(x, y) &= 5 + 3x - 2(y - 1) \\ p_2(x, y) &= p_1(x, y) + 0x(x - 1) + 1x(y - 1) + 4(y - 1)(y + 1) \\ p_3(x, y) &= p_2(x, y) + 6x(x - 1)(x + 1) - 2x(x - 1)(y - 1) \\ &\quad + 2x(y - 1)(y + 1) - 6(y - 1)(y + 1)y. \end{aligned}$$

Then, p_1 is the plane through points $(0, 1, 5)$, $(1, 1, 3)$, $(0, -1, -2)$. The quadric surface p_2 is some type of paraboloid through values on the six corner nodes of order ≤ 2 (so this also serves as an example solution for the general Figure 1(b) nodes). The cubic polynomial through all ten points may be multiplied-out to find the standard basis form,

$$p_3(x, y) = 3 - 8x + 4y + 2x^2 + 3xy + 2y^2 + 6x^3 - 2x^2y + 2xy^2 - 6y^3.$$

To see what is going on and why these divided-differences work, we provide a proof for arbitrary n variables up to degree d (which does not seem to have appeared in print, although the idea is very old [10, p. 204]). Actually the proof is valid in a slightly more general context. Fix a set of nodes and values $\{(\mathbf{x}(\boldsymbol{\lambda}), f_{\boldsymbol{\lambda}}) : \boldsymbol{\lambda} \in J\}$ over some set of multi-indices $J \subseteq \mathbb{N}_0^n$ with the property that whenever $\boldsymbol{\beta} \in J$ and $\mathbf{0} \leq \boldsymbol{\lambda} \leq \boldsymbol{\beta}$ coordinate-wise, then $\boldsymbol{\lambda} \in J$. For multi-indices, \leq will always mean coordinate-wise, so $\{\boldsymbol{\lambda} : \boldsymbol{\alpha} \leq \boldsymbol{\lambda} \leq \boldsymbol{\beta}\}$ is an n -dimensional rectangular box of multi-indices. The proof relies on the fact: for any box of nodes and values $\{(\mathbf{x}(\boldsymbol{\lambda}), f_{\boldsymbol{\lambda}}) : \boldsymbol{\alpha} \leq \boldsymbol{\lambda} \leq \boldsymbol{\beta}\}$, a simple shift of Theorem 1 shows that there is a unique interpolating polynomial in

$$\begin{aligned} &\text{Span} \left\{ \prod_{m=1}^n \prod_{i=\alpha_m}^{\lambda_m-1} (x_m - x_m(i)) : \boldsymbol{\alpha} \leq \boldsymbol{\lambda} \leq \boldsymbol{\beta} \right\} \\ &= \text{Span}\{x_1^{\mu_1} x_2^{\mu_2} \cdots x_n^{\mu_n} : \mathbf{0} \leq \boldsymbol{\mu} \leq \boldsymbol{\beta} - \boldsymbol{\alpha}\}. \end{aligned}$$

The property of J insures that all boxes of nodes and values with top index $\boldsymbol{\beta}$ are given whenever $\boldsymbol{\beta} \in J$. This property clearly holds for a corner of nodes.

Definition 2 For $\boldsymbol{\alpha}, \boldsymbol{\beta} \in J$ with $\boldsymbol{\alpha} \leq \boldsymbol{\beta}$, define $f[\boldsymbol{\alpha}, \boldsymbol{\beta}]$ to be the coefficient of the highest power in the interpolating polynomial for nodes and values $\{(\mathbf{x}(\boldsymbol{\lambda}), f_{\boldsymbol{\lambda}}) : \boldsymbol{\alpha} \leq \boldsymbol{\lambda} \leq \boldsymbol{\beta}\}$.

So $f[\boldsymbol{\alpha}, \boldsymbol{\beta}]$ is the unique coefficient of $x_1^{\mu_1} x_2^{\mu_2} \cdots x_n^{\mu_n}$ where $\boldsymbol{\mu} = \boldsymbol{\beta} - \boldsymbol{\alpha}$, which equals the unique coefficient of highest order term in the Newton form

of the polynomial. In traditional divided-difference notation [3, p. 296], this coefficient $f[\boldsymbol{\alpha}, \boldsymbol{\beta}]$ would be written as

$$f[x_1(\alpha_1), \dots, x_1(\beta_1); x_2(\alpha_2), \dots, x_2(\beta_2); \dots; x_n(\alpha_n), \dots, x_n(\beta_n)]$$

which would be much more cumbersome.

Theorem 3 *The polynomial $p(\mathbf{x}) = \sum_{\boldsymbol{\lambda} \in J} f[\mathbf{0}, \boldsymbol{\lambda}] q_{\boldsymbol{\lambda}}(\mathbf{x})$ satisfies $p(\mathbf{x}(\boldsymbol{\lambda})) = f_{\boldsymbol{\lambda}}$ for all $\boldsymbol{\lambda} \in J$.*

Proof. By Theorem 1, there exists a unique interpolating polynomial of the form $\hat{p}(\mathbf{x}) = \sum_{\boldsymbol{\lambda} \in J} a_{\boldsymbol{\lambda}} q_{\boldsymbol{\lambda}}(\mathbf{x})$. Let $\boldsymbol{\mu} \in J$. We need only show that $a_{\boldsymbol{\mu}} = f[\mathbf{0}, \boldsymbol{\mu}]$. For this fixed $\boldsymbol{\mu}$,

$$f_{\boldsymbol{\mu}} = \hat{p}(\mathbf{x}(\boldsymbol{\mu})) = \sum_{\boldsymbol{\lambda} \leq \boldsymbol{\mu}} a_{\boldsymbol{\lambda}} q_{\boldsymbol{\lambda}}(\mathbf{x}(\boldsymbol{\mu})) + \sum_{\boldsymbol{\lambda} \in J, \boldsymbol{\lambda} \not\leq \boldsymbol{\mu}} a_{\boldsymbol{\lambda}} q_{\boldsymbol{\lambda}}(\mathbf{x}(\boldsymbol{\mu})).$$

By (4), the second sum is zero. Thus, the first sum is the interpolating polynomial for the box $\{(\mathbf{x}(\boldsymbol{\lambda}), f_{\boldsymbol{\lambda}}) : \mathbf{0} \leq \boldsymbol{\lambda} \leq \boldsymbol{\mu}\}$ and $a_{\boldsymbol{\mu}} = f[\mathbf{0}, \boldsymbol{\mu}]$ by definition. ■

The algorithm is based on initializing each $f[\boldsymbol{\lambda}, \boldsymbol{\lambda}] = f_{\boldsymbol{\lambda}}$ and using divided-differences to work the first argument down to $f[\mathbf{0}, \boldsymbol{\lambda}]$ by repeated use of the following.

Theorem 4 *For $\boldsymbol{\alpha}, \boldsymbol{\beta} \in J$ with $\boldsymbol{\alpha} \leq \boldsymbol{\beta}$ and $\alpha_m > 0$,*

$$f[\boldsymbol{\alpha} - \mathbf{e}_m, \boldsymbol{\beta}] = \frac{f[\boldsymbol{\alpha}, \boldsymbol{\beta}] - f[\boldsymbol{\alpha} - \mathbf{e}_m, \boldsymbol{\beta} - \mathbf{e}_m]}{x_m(\beta_m) - x_m(\alpha_m - 1)}$$

where $\mathbf{e}_m = (0, \dots, 0, 1, 0, \dots, 0)$ with 1 in m th coordinate.

Proof. We consider unique interpolating polynomials over different boxes of multi-indices as diagrammed in Figure 3. Let $p_+(\mathbf{x})$ be the interpolating polynomial for $\{(\mathbf{x}(\boldsymbol{\lambda}), f_{\boldsymbol{\lambda}}) : \boldsymbol{\alpha} \leq \boldsymbol{\lambda} \leq \boldsymbol{\beta}\}$. Let $p_-(\mathbf{x})$ be the interpolating polynomial for $\{(\mathbf{x}(\boldsymbol{\lambda}), f_{\boldsymbol{\lambda}}) : \boldsymbol{\alpha} - \mathbf{e}_m \leq \boldsymbol{\lambda} \leq \boldsymbol{\beta} - \mathbf{e}_m\}$. Define

$$p(\mathbf{x}) = p_-(\mathbf{x}) + \left(\frac{x_m - x_m(\alpha_m - 1)}{x_m(\beta_m) - x_m(\alpha_m - 1)} \right) (p_+(\mathbf{x}) - p_-(\mathbf{x})). \quad (7)$$

Since both $p_+(\mathbf{x})$ and $p_-(\mathbf{x})$ are in $\text{Span}\{x_1^{\mu_1} x_2^{\mu_2} \dots x_n^{\mu_n} : \mathbf{0} \leq \boldsymbol{\mu} \leq \boldsymbol{\beta} - \boldsymbol{\alpha}\}$, $p(\mathbf{x})$ is in $\text{Span}\{x_1^{\mu_1} x_2^{\mu_2} \dots x_n^{\mu_n} : \mathbf{0} \leq \boldsymbol{\mu} \leq \boldsymbol{\beta} - \boldsymbol{\alpha} + \mathbf{e}_m\}$. We now show that $p(\mathbf{x})$ interpolates over the union of the boxes $\{(\mathbf{x}(\boldsymbol{\lambda}), f_{\boldsymbol{\lambda}}) : \boldsymbol{\alpha} - \mathbf{e}_m \leq \boldsymbol{\lambda} \leq \boldsymbol{\beta}\}$. The overlapping indices $\boldsymbol{\alpha} \leq \boldsymbol{\lambda} \leq \boldsymbol{\beta} - \mathbf{e}_m$ satisfy $p_+(\mathbf{x}(\boldsymbol{\lambda})) - p_-(\mathbf{x}(\boldsymbol{\lambda})) = 0$, so that $p(\mathbf{x}(\boldsymbol{\lambda})) = p_-(\mathbf{x}(\boldsymbol{\lambda})) = f_{\boldsymbol{\lambda}}$. The non-overlapping edge with $\lambda_m = \alpha_m - 1$ satisfies $x_m(\lambda_m) - x_m(\alpha_m - 1) = 0$, so that $p(\mathbf{x}(\boldsymbol{\lambda})) = p_-(\mathbf{x}(\boldsymbol{\lambda})) = f_{\boldsymbol{\lambda}}$. The other non-overlapping edge with $\lambda_m = \beta_m$ satisfies $p(\mathbf{x}(\boldsymbol{\lambda})) = p_-(\mathbf{x}(\boldsymbol{\lambda})) + (1)(p_+(\mathbf{x}(\boldsymbol{\lambda})) - p_-(\mathbf{x}(\boldsymbol{\lambda}))) = p_+(\mathbf{x}(\boldsymbol{\lambda})) = f_{\boldsymbol{\lambda}}$. We conclude that $p(\mathbf{x})$ is the interpolating polynomial over

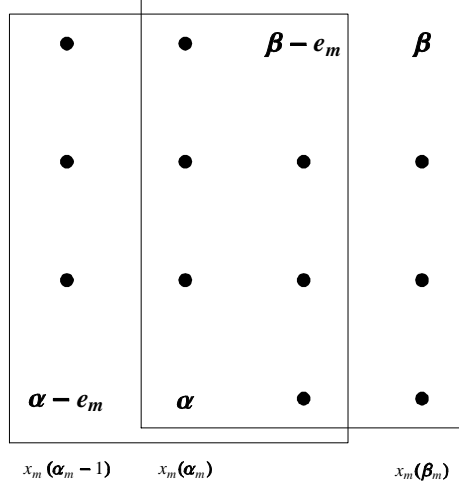


Figure 3: Multi-indices in proof of divided-difference

the union and, by definition, the coefficient of the highest power is $f[\alpha - e_m, \beta]$. This highest power is $x_1^{\mu_1} x_2^{\mu_2} \cdots x_n^{\mu_n}$ for $\mu = \beta - \alpha + e_m$, which occurs in (7) as the highest power term of $p_+(\mathbf{x})$ minus the highest power term of $p_-(\mathbf{x})$, all times the linear factor. The coefficients of these highest powers are respectively, $f[\alpha, \beta]$ for $p_+(\mathbf{x})$, $f[\alpha - e_m, \beta - e_m]$ for $p_-(\mathbf{x})$, and $1/(x_m(\beta_m) - x_m(\alpha_m - 1))$ for the linear factor, resulting in the desired divided-difference formula. ■

The algorithm that generalizes the example in divided-difference Table 1 is conceptually straight-forward and does not require pairs of multi-indices as long as the order of computation is handled carefully. For each multi-index $\beta \in J$, initialize memory location $A[\beta] = f[\beta, \beta] = f_\beta$. This location will be changed $|\beta|$ times by Theorem 4 to reach $f[0, \beta]$. We consistently choose the direction of decrements so that, before each pass through the array, there exist α and e_m such that $A[\beta] = f[\alpha, \beta]$ and $A[\beta - e_m] = f[\alpha - e_m, \beta - e_m]$. The divided difference will then compute $A[\beta] = f[\alpha - e_m, \beta]$. One method, corresponding to the "left before below" choice in Table 1, is to simply decrement the coordinates of β left to right, i.e., for the k th decrement, choose the first coordinate m such that $(\sum_{i=1}^m \beta_i) - k \geq 0$, and observe that this non-negative quantity is $\alpha_m - 1$. For example, on the 5th pass, $\beta = (2, 0, 4, 1)$ will have been decremented 4 times to $\alpha = (0, 0, 2, 1)$ and will compute further to $\alpha - e_m = (0, 0, 1, 1)$, lowering the $m = 3$ coordinate to $\alpha_m - 1 = 1$. Since $\beta - e_m = (2, 0, 3, 1)$, the 5th application of Theorem 4 at $\beta = (2, 0, 4, 1)$, updates

$$A[(2, 0, 4, 1)] = \frac{A[(2, 0, 4, 1)] - A[(2, 0, 3, 1)]}{x_3(4) - x_3(1)}.$$

The previous four passes will have computed $A[(2, 0, 3, 1)] = f[(0, 0, 1, 1), (2, 0, 3, 1)]$. (If we would have computed $A[(2, 0, 3, 1)]$ by a right-to-left method, then the

location would hold erroneously $f[(2, 0, 0, 0), (2, 0, 3, 1)]$, so a consistent method is necessary throughout the array.) Also, the k th pass through the array must proceed "backwards" updating higher order multi-indices first and stopping after updating multi-indices of order k , since these entries contain the final $f[\mathbf{0}, \boldsymbol{\beta}]$ value.

Computer implementation of this algorithm is straight-forward in a true n -dimensional array. However, non-rectangular index sets and arbitrary dimensions are probably best handled by a "virtual array" using indirect referencing into a one-dimensional array. In this case, finding the location of $A[\boldsymbol{\beta} - \mathbf{e}_m]$ can be tricky, as addressed in [6, p. 333]. An interesting first exercise is to develop a successor function, depending on n , that increments from one multi-index to the next in the desired ordering.

4 Polynomials for arbitrary nodes

Having seen how a classic Newton polynomial basis handled a corner of nodes, we seek to generalize the idea to arbitrary nodes. The problem goes back to the Vandermonde matrix equation $V^T \mathbf{c} = \mathbf{f}$ in (1) for the coefficients of the interpolating polynomial. Assume, for now, that solution by Gaussian elimination succeeds without row exchanges. This direct solution would be equivalent to producing the LU -decomposition of V^T and solving $L\mathbf{y} = \mathbf{f}$ and $U\mathbf{c} = \mathbf{y}$, to get the coefficients \mathbf{c} in the standard basis. Instead, we will do steps very similar to an LU -decomposition on V and find an upper-triangular W and lower-triangular matrix R , where R defines polynomials in a generalized Newton basis for the nodes, and where solving just $W^T \mathbf{a} = \mathbf{f}$ will give the coefficients of nested interpolating polynomials.

Again we start with a specific example of nodes, this time from Figure 1(c). The Vandermonde matrix is augmented with an identity matrix, forming

$$\begin{array}{c|cccccc|cccc}
 & \text{nodes} & & & & & 1 & x & y & x^2 & xy & y^2 \\
 \hline
 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
 0 & 1 & 2 & 2 & -1 & -2 & 0 & 1 & 0 & 0 & 0 & 0 \\
 0 & -1 & 1 & 2 & 2 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
 \hline
 0 & 1 & 4 & 4 & 1 & 4 & 0 & 0 & 0 & 1 & 0 & 0 \\
 0 & -1 & 2 & 4 & -2 & -2 & 0 & 0 & 0 & 0 & 1 & 0 \\
 0 & 1 & 1 & 4 & 4 & 1 & 0 & 0 & 0 & 0 & 0 & 1
 \end{array} \quad (8)$$

Each column in the left-half corresponds to a node in an arbitrary but fixed ordering and with coordinates initially shown in rows 2 through $n + 1$ (for n variables). Each column in the right-half corresponds to a standard basis monomial, listed with increasing degree, as labeled in the example.

Proposition 5 *For every row of the Vandermonde-Identity matrix, or any row-equivalent matrix, the polynomial with monomial coefficients listed in the right-half matrix row has values in the left-half matrix row at each of the nodes, corresponding to columns.*

Initially this holds by definition of the Vandermonde matrix. For an arbitrary example, suppose we add -3 times the third row into the fourth row in (8). Then, the right-half of row four will be $[0, 0, -3, 1, 0, 0]$ which defines polynomial $-3y + x^2$ and the left-half row four will contain -3 times the y values plus the x^2 values. Adding one row into another always adds the polynomial coefficients on the right while adding the corresponding polynomial values on the left. Clearly, any elementary row operation will not change the property as described in the Proposition 5. Actually, row exchanges are never done, as motivated below. Instead, when necessary, perform a column exchange and note the change in the ordering of nodes from the arbitrary start.

The goal of row operations and column exchanges will be an upper-triangular matrix with identity matrices in the diagonal blocks on the left, where we divide into blocks according to the order of the multi-index associated with each row and column. Our example already has zeros in the first column since we started with the origin. (Otherwise, pivoting on the upper-left 1 would translate all nodes so that the first becomes the origin.) The process reduces (8), without any exchanges, to $[W | R]$:

nodes						1	x	y	x^2	xy	y^2
1	1	1	1	1	1	1	0	0	0	0	0
0	1	0	$-\frac{2}{3}$	$-\frac{5}{3}$	$-\frac{4}{3}$	0	$\frac{1}{3}$	$-\frac{2}{3}$	0	0	0
0	0	1	$\frac{4}{3}$	$\frac{1}{3}$	$-\frac{1}{3}$	0	$\frac{1}{3}$	$\frac{1}{3}$	0	0	0
0	0	0	1	0	0	0	$-\frac{4}{13}$	$-\frac{9}{26}$	$\frac{1}{39}$	$\frac{4}{13}$	$\frac{19}{78}$
0	0	0	0	1	0	0	$\frac{3}{13}$	$\frac{5}{13}$	$-\frac{4}{39}$	$-\frac{3}{13}$	$\frac{1}{39}$
0	0	0	0	0	1	0	$\frac{13}{52}$	$-\frac{11}{52}$	$\frac{9}{52}$	$\frac{1}{13}$	$\frac{1}{52}$

The matrix R defines polynomials r_{λ} for each row multi-index λ , e.g. $r_{10}(x, y) = \frac{1}{3}x - \frac{2}{3}y$. The values of these polynomials are given in W and reveal properties similar to the Newton basis. The last three quadratics are zero on five of the six points as shown in Figure 4 where we see one ellipse and two hyperbolas. In particular, each r_{λ} is a polynomial that

- (a) is zero at nodes of lower order,
- (b) is one at the corresponding node and zero at other nodes of the same order, and
- (c) has degree $|\lambda|$.

In comparison with properties of the classic Newton basis (5), this slightly strengthens (b) by normalizing to value one, but weakens (c) since simple algebraic structure may not be possible. Row exchanges are avoided since they could change the degree (c). If we had started with a corner of nodes in multi-index order, this algorithm would produce the classic Newton polynomial basis only normalized (as uniqueness demands) and not in factored form.

We call $\{r_{\lambda}\}$ the *Newton-Sauer polynomials*, since they are produced in [8]. Our row-operations with column exchanges are equivalent to the steps in [8,

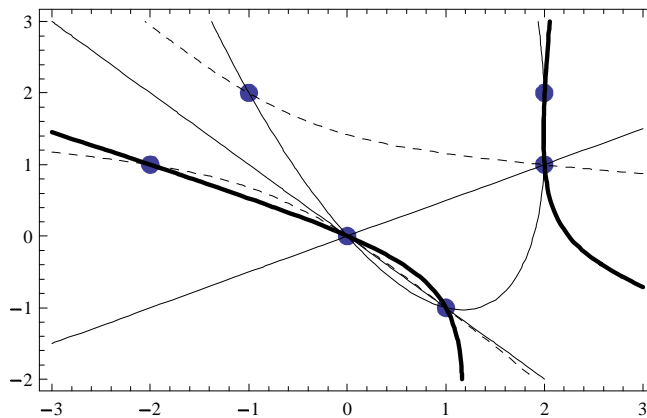


Figure 4: Zero contours of Newton-Sauer polynomials

Algorithm 4.3] which does not use matrices but operates on polynomials directly (equivalent to right-half matrix coefficients only). Polynomial values are always recomputed in [8, Algorithm 4.4], which says that it might be convenient to store these values as we have done in matrix W . Of course, our actual matrix memory could be halved as is commonly done in implementation of LU -decomposition, by storing the lower-triangular values below upper-triangular values. The process $[V | I] \rightarrow [W | R]$ can be seen as producing a matrix factorization. The row operations (scaling and additions) can be encapsulated in a matrix E and column exchanges in a permutation matrix P , so that $E [V P | I] = [W | R]$. We conclude that $R V P = W$, which again says that values of the resulting polynomials are given in W . (An LU factorization is given by $V P = R^{-1} W$.)

To finish the example, we find the interpolating polynomial for values, say $\mathbf{f} = [5, 6, 7, 8, 9, 10]^T$, at the six nodes in Figure 4. Values should match the ordering of nodes after any needed column exchanges (of which there were none in the example). Since W holds values of the Newton-Sauer polynomials at the nodes, $W^T \mathbf{a} = \mathbf{f}$ describes coefficients of the interpolating polynomial in this basis. This is a particularly easy lower-triangular system, though W is not quite as simple as the classic (3) that was solved by divided-differences. Here, forward-substitution (in row or block forms) may be used and results in $\mathbf{a} = [5, 1, 2, 1, 5, 7]^T$. Thus, $p_1(x, y) = 5 + r_{10}(x, y) + 2 r_{01}(x, y)$ is the interpolating polynomial for the three nodes in the lines of Figure 4. For all six, the interpolating polynomial is

$$p_2(x, y) = p_1(x, y) + r_{20}(x, y) + 5 r_{11}(x, y) + 7 r_{02}(x, y)$$

which multiplies into $\frac{1}{156}(780 - 69x + 15y + 113x^2 - 48xy + 79y^2)$ if desired.

Before summarizing this algorithm in general, let's see the insight it brings to an "impossible" example such as the circle of nodes in Figure 1(a). The

nodes are put into the Vandermonde matrix in the ordering

$$(-1, -\sqrt{3}), (1, -\sqrt{3}), (-1, \sqrt{3}), (1, \sqrt{3}), (-2, 0), (2, 0).$$

The algorithm produces equivalent results for any ordering although it affects the exact Newton-Sauer polynomials. Row multiples and additions on the Vandermonde-Identity matrix produce the 2×2 identity block and leave a zero in the pivot position in column 4. This is overcome by exchanging columns 4 and 5, so that the ordering of nodes $(1, \sqrt{3})$ and $(-2, 0)$ is flipped. The real interest comes after five pivots and we are left with:

$$\begin{array}{c|ccc|ccc|ccc}
 & \text{circle of nodes} & & 1 & x & y & x^2 & xy & y^2 \\
 \hline
 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
 0 & 1 & 0 & \frac{-1}{2} & 1 & \frac{3}{2} & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\
 0 & 0 & 1 & \frac{1}{2} & 1 & \frac{1}{2} & \frac{1}{2} & 0 & \frac{\sqrt{3}}{6} & 0 \\
 0 & 0 & 0 & 1 & 0 & 1 & \frac{-1}{3} & 0 & 0 & \frac{1}{3} \\
 0 & 0 & 0 & 0 & 1 & 1 & \frac{1}{6} & \frac{1}{4} & \frac{\sqrt{3}}{12} & \frac{1}{12} \\
 0 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & 0 & 1
 \end{array} \quad (9)$$

The last row gives the polynomial $p(x, y) = -4 + x^2 + y^2$ where all nodes have value zero; finding the circle that makes the unique interpolation impossible! For other nodes, a row of zeros may appear before the last row. In any case, the outcome of this algorithm is satisfying. Either we find all nonzero pivots and a basis for nested interpolating polynomials (of desired degree), or we find a row of zeros on the left and an algebraic hypersurface (of desired degree) that contains all the nodes.

Theorem 6 *Given a set of $\binom{n+d}{d}$ nodes in n variables, form a Vandermonde-Identity matrix $[V | I]$. Using elementary row operations with column instead of row exchanges, attempt to row-reduce to $[W | R]$ where W is an upper-triangular matrix with identity matrices on the diagonal blocked by order of multi-index. This attempt will either*

(a) *succeed, in which case, there is an ordering of the nodes \mathbf{x}_μ (after column exchanges) and Newton-Sauer polynomials $r_\lambda(\mathbf{x})$ of degree $|\lambda|$ given by coefficients (in terms of the standard monomial basis) for each row λ of R such that*

$$r_\lambda(\mathbf{x}_\mu) = \begin{cases} 0 & \text{if } |\mu| \leq |\lambda| \text{ and } \mu \neq \lambda \\ 1 & \text{if } \mu = \lambda \end{cases}; \text{ or}$$

(b) *fail, because some row ν becomes all zeros in the left-half matrix, in which case, the right-half matrix row ν is coefficients of polynomial $r_\nu(\mathbf{x})$ of degree $\leq d$ such that $r_\nu(\mathbf{x}_\mu) = 0$ for all nodes.*

5 Interpolation subspace of minimal degree

Even in the "impossible" case of Theorem 6(b), the method can be extended to find a polynomial subspace of minimal degree that does contain unique interpolating polynomials for the given nodes. Whenever a row of zeros is encountered,

6 Block Gaussian elimination

Different generalizations of the Newton polynomials will result from different choices for the goal of the row reduction. In the one-dimensional case, diagonal entries become either 1 or non-zero. For multivariable diagonal blocks, this suggests either an identity matrix as in Section 4 or an invertible matrix as in this Section and in [7]. The intermediate goal of upper-triangular is another reasonable choice. The choices represent a trade-off between which properties of the classic polynomials (5) will be retained in the generalization. We retain simpler algebraic structure by using block Gaussian elimination to produce invertible diagonal matrices.

For example, we start with the same Vandermonde-Identity matrix (8) for the nodes from Figure 1(c). The first column is already in desired form and the next diagonal 2×2 block is invertible. To pivot on this block, form the block below it times the inverse of this pivot block

$$\begin{bmatrix} 1 & 4 \\ -1 & 2 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ -1 & 1 \end{bmatrix}^{-1}$$

and add this multiple of the order-one rows into rows below, resulting in

nodes						1	x	y	x ²	xy	y ²
1	1	1	1	1	1	1	0	0	0	0	0
0	1	2	2	-1	-2	0	1	0	0	0	0
0	-1	1	2	2	1	0	0	1	0	0	0
0	0	0	$\frac{-2}{3}$	$\frac{4}{3}$	$\frac{20}{3}$	0	$\frac{-5}{3}$	$\frac{-2}{3}$	1	0	0
0	0	0	$\frac{2}{3}$	$\frac{-13}{3}$	$\frac{16}{3}$	0	$\frac{-1}{3}$	$\frac{-4}{3}$	0	1	0
0	0	0	$\frac{10}{3}$	$\frac{16}{3}$	$\frac{3}{3}$	0	$\frac{-2}{3}$	$\frac{1}{3}$	0	0	1

(10)

The 3×3 block is invertible, so unique interpolation is possible, using the polynomial basis given by rows of the right-half. We call these *Newton-Olver polynomials*

$$1, x, y, \frac{-1}{3}(5x + 2y) + x^2, \frac{-1}{3}(x + 4y) + xy, \frac{-1}{3}(2x - y) + y^2.$$

Compared to properties of the classic polynomials (5), property (b) about values at nodes of the same order is essentially lost in favor of regaining property (c): each Newton-Olver polynomial has degree $|\lambda|$ and only one term of highest degree, and the term has coefficient one.

Some notation will help us better describe this method and the forward substitution to solve for a specific interpolating polynomial. For $\binom{n+d}{d}$ nodes in n variables, start with the Vandermonde-Identity $[V | I]$ and reduce to a block-upper-triangular $[U | S]$ with invertible matrices on the diagonal, where $m_k = \binom{n+k-1}{k}$ is the size of the block for order k . Name resulting blocks, as

done here for $d = 3$:

$$\left[\begin{array}{cccc} \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & U_{11} & U_{12} & U_{13} \\ \mathbf{0} & \mathbf{0} & U_{22} & U_{23} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & U_{33} \end{array} \parallel \begin{array}{cccc} 1 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ S_{10} & I_{m_1} & \mathbf{0} & \mathbf{0} \\ S_{20} & S_{21} & I_{m_2} & \mathbf{0} \\ S_{30} & S_{31} & S_{32} & I_{m_3} \end{array} \right].$$

This end is achieved by block Gaussian elimination where every step is pivoting on block U_{kk} : assume all rows above and columns to left of U_{kk} are done, group all remaining parts of the augmented matrix into $\begin{bmatrix} U_{kk} & B \\ C & D \end{bmatrix}$, and replace D by $D - CU_{kk}^{-1}B$. The next diagonal block is now defined. If it is not invertible, it is probably more practical to switch to the row operations of Section 4 in order to find the necessary column exchanges or find the hypersurface containing the nodes. We assume nodes have been ordered so that diagonal blocks are invertible.

The Newton-Olver name reflects that this process is equivalent to the recursive algorithm described in [7, Theorem 5]. Their notation $\Omega_k^i(X_j)$ denotes row block i and column block j on the k th iteration in our left-half matrix. However, the corresponding polynomials are not given by coefficients as in our right-half but by this process on the standard basis monomials. Actually, [7] includes binomial coefficients with the monomials in the original Vandermonde matrix, so our results differ by this scaling of terms

To find the interpolating polynomial for given node values, block-forward-substitution can be done recursively in a manner reminiscent of the divided-difference algorithm. The block-triangular U holds values of the Newton-Olver basis at the nodes. Rather than transpose as before, let the desired node values \mathbf{f} and coefficients \mathbf{a} be row vectors and solve $\mathbf{a}U = \mathbf{f}$. Partition $\mathbf{f} = [\mathbf{f}_0, \mathbf{f}_1, \dots, \mathbf{f}_d]$ in blocks according to order of the multi-index. Similar to the divided-difference algorithm, initialize an array of function values $\mathbf{a} = \mathbf{f}$ and update on passes for each $k = 1, \dots, d$: update $\mathbf{a}_k = \mathbf{a}_k - \mathbf{a}_i U_{ik}$ for $i = 0, \dots, k-1$ and then "divide" $\mathbf{a}_k = \mathbf{a}_k U_{kk}^{-1}$. The equivalent algorithm in [7, equation (75)] is called "a full multivariate version of the classical recursive divided difference scheme." For the generic example in (10) and initial $\mathbf{f} = [[5], [6, 7], [8, 9, 10]]$, we get

$$\begin{aligned} \mathbf{a}_0 &= [5], \\ \mathbf{a}_1 &= ([6, 7] - \mathbf{a}_0[1, 1])U_{11}^{-1} \\ &= [1, 2]U_{11}^{-1} = [1, 0] \\ \mathbf{a}_2 &= ([8, 9, 10] - \mathbf{a}_0[1, 1, 1] - \mathbf{a}_1 U_{12})U_{22}^{-1} \\ &= [1, 5, 7]U_{22}^{-1} = [\frac{113}{156}, \frac{-4}{13}, \frac{79}{156}] \end{aligned}$$

Thus, the unique interpolating polynomial is

$$\begin{aligned} p(x, y) &= 5 + x + \frac{113}{156} \left(-\frac{5}{3}x - \frac{2}{3}y + x^2 \right) \\ &\quad - \frac{4}{13} \left(-\frac{1}{3}x - \frac{4}{3}y + xy \right) + \frac{79}{156} \left(-\frac{2}{3}x + \frac{1}{3}y + y^2 \right) \\ &= \frac{1}{156} (780 - 69x + 15y + 113x^2 - 48xy + 79y^2). \end{aligned}$$

For a corner of nodes in multi-index ordering, the Newton-Olver polynomials are the classic Newton polynomials and all U_{i_k} are the "tick-mark" differences as in (3). Again uniqueness demands this. If the corner has a classic Newton $q_\lambda(\mathbf{x})$ and the Newton-Olver is $\mathbf{x}^\lambda + u(\mathbf{x})$ where $u(\mathbf{x})$ is lower degree, then $q_\lambda(\mathbf{x}) - \mathbf{x}^\lambda$ and $u(\mathbf{x})$ must be the same interpolating polynomial for nodes of lower order.

7 Conclusion

Multivariable interpolation is more involved and more interesting than the one-dimensional case since a unique solution is not guaranteed and a convenient polynomial basis is not so clear. It depends on the nodes, assuming $m = \binom{n+d}{d}$ nodes in n dimensions. If the nodes are in the corner of some arbitrary grid, then we have a guaranteed solution and clear Newton basis, with an efficient divided-difference algorithm to compute an interpolating polynomial. More generally, existence and uniqueness depend on whether the nodes fall on an algebraic hypersurface (a level curve of a polynomial) of degree d . A matrix algorithm, similar to Gaussian elimination, answers this question and produces the Newton-Sauer basis. In fact, the algorithm produces an appropriate polynomial subspace of minimal degree, where unique interpolation is possible, for any distinct nodes of any number. Alternatively, a block version of Gaussian elimination can be used to form the algebraically simpler Newton-Olver basis.

The divided-difference algorithm and the matrix algorithms represent very different levels of work. Given an interpolation problem, our first phase is finding the appropriate Newton basis and second phase is solving for the coefficients of an interpolating polynomial. For a corner of nodes, the first phase takes no work, just write down the classic Newton basis. The divided-difference algorithm is an extremely efficient second phase using an array of m entries updated on d passes, so the operation count is roughly on the order of dm . Compare this to both matrix algorithms for general nodes that work with $m \times m$ matrices. The first phase is much like LU -decomposition requiring roughly m^3 operations. The second phase solves a triangular system requiring roughly m^2 operations. Of course, only the second phase need be repeated for different values on the same nodes. Numerical tests in [8] show that the Newton-Sauer basis is an efficient and accurate method for handling arbitrary nodes, at least in comparison with an alternative Lagrange basis method. Also, the column exchanges can be used to reduce roundoff error. Since initial ordering of nodes is generally meaningless, partial pivoting that picks the next (node) column with the largest relative pivot element is a standard numerical analysis technique to reduce error [5, Section 4.3]. Thus, the matrix algorithms are a practical means to decide and accomplish polynomial interpolation on arbitrary nodes. Still, the classic Newton polynomials and divided-difference algorithm make it very worthwhile to impose or deduce the structure of a corner of nodes when possible.

References

- [1] C. de Boor and A. Ron, Computational aspects of polynomial interpolation in several variables, *Mathematics of Computation* **58** (1992) 705-727.
- [2] M. Gasca and T. Sauer, Polynomial interpolation in several variables, *Advances in Computational Math.* **12** (2000), 377-410.
- [3] E. Isaacson and H. B. Keller, *Analysis of Numerical Methods*, Wiley, New York, 1966.
- [4] K. S. Kunz, *Numerical Analysis*, McGraw-Hill, New York, 1957.
- [5] D. Kincaid and W. Cheney, *Numerical Analysis*, 3rd ed., Brooks/Cole, Pacific Grove, CA, 2002.
- [6] R. D. Neidinger, Directions for Computing Truncated Multivariate Taylor Series, *Mathematics of Computation* **74** (2005) 321-340.
- [7] P. J. Olver, On Multivariate Interpolation, *Studies in Applied Math.*, **116** (2006) 201-240.
- [8] T. Sauer, Computational aspects of multivariate polynomial interpolation, *Advances in Computational Math.* **3** (1995) 219-237.
- [9] T. Sauer, Polynomial interpolation of minimal degree, *Numerische Mathematik* **78** (1997) 59-85.
- [10] J.F. Steffensen, *Interpolation*, 2nd ed., Chelsea Publishing, New York, 1950 (1st ed., 1927).
- [11] T. Sauer and Y. Xu, On multivariate Lagrange interpolation, *Mathematics of Computation* **64** (1995) 1147-1170.