SIAM REVIEW
Vol. 61, No. 2, pp. 361–381

Multivariate Polynomial Interpolation in Newton Forms*

Richard D. Neidinger[†]

Abstract. Techniques of univariate Newton interpolating polynomials are extended to multivariate data points by different generalizations and practical algorithms. The Newton basis format, with divided-difference algorithm for coefficients, generalizes in a straightforward way when interpolating at nodes on a grid within certain regions. While this idea, known as the tensor product case, has been around for over a century, this exposition uses modern multi-index notation to provide generality, concise algorithms, and rigor, appropriate as a topic in introductory numerical analysis. Node configurations where the tensor product case applies are called lower sets, and they include n-dimensional rectangles (boxes) and triangles (corners), the latter having unique interpolation over polynomials of fixed degree. Arbitrary distinct nodes do not ensure unique interpolating polynomials but, when possible, a different basis generalization, which we call Newton–Sauer, results in a nice triangular linear system for the coefficients. For the right number of distinct nodes, an algorithm based on Gaussian elimination produces either this Newton–Sauer basis or a polynomial that is zero on all nodes, showing that unique interpolation is impossible. The algorithm may also be used on any distinct nodes to produce a polynomial subspace of minimal degree where unique interpolation is possible. A variation of this algorithm using matrix block operations produces another basis generalization that we call Newton–Olver, which exactly agrees with the classic one for a corner of nodes and computes an interpolating polynomial using formulas similar to divided differences.

Key words. interpolation, multivariate, Newton polynomial, divided difference, algorithm, multidimensional

AMS subject classifications. 65D05, 41A05, 41A63, 41A10, 97N50

DOI. 10.1137/17M1124188

1. Introduction.

Ideas and techniques of univariate interpolation, from undergraduate numerical analysis and linear algebra, can be expanded and applied in a multivariate setting. Interpolation seeks a polynomial with desired function values at some number of distinct nodes in the domain. We focus on constructive algorithms to build a convenient polynomial basis corresponding to the nodes and to produce interpolating polynomial coefficients for specific function values.

Unlike univariate interpolation, multivariate success depends on the number and relative location of the nodes. In one variable, distinct nodes always guarantee a unique interpolating polynomial. In just two variables, suppose we seek a quadratic polynomial

$$p(x, y) = a + bx + cy + dx^2 + exy + fy^2.$$

With six unknowns, we would hope to interpolate on six nodes and consider three

*Received by the editors April 4, 2017; accepted for publication (in revised form) July 6, 2018; published electronically May 8, 2019.
http://www.siam.org/journals/sirev/61-2/M112418.html

[†]Department of Mathematics and Computer Science, Davidson College, Davidson, NC 28035 (rineidinger@davidson.edu).

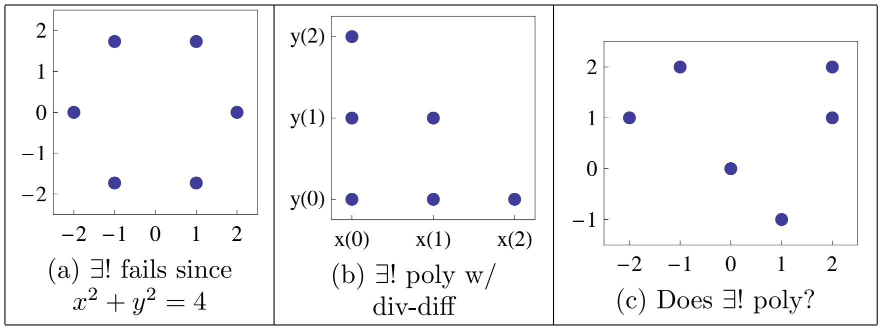

Fig. 1 *Impossible circle, easy corner, and generic example.*

examples in Figure 1 that will guide our discussion. For each set of nodes, a linear system of equations results from setting the polynomial at the nodes to some function values. In (a), if there exists a solution on this circle of nodes, then adding x^2+y^2-4 will produce another polynomial solution. Thus, the linear system must have either no solution or infinitely many. In (b), using a different polynomial basis (Newton form), with factors of the form $(x - x(i))$ and $(y - y(i))$, will give a particularly simple invertible triangular system. In (c), we could resort to Gaussian elimination on the matrix of the linear system to answer existence and uniqueness. For such an arbitrary set of nodes, we present an algorithm based on Gaussian elimination that will find either a generalization of the nice Newton basis in (b) that facilitates unique interpolation or a polynomial that evaluates as zero at all nodes, as in (a).

Sections 2 to 5 pursue the generalization of the case in Figure 1(b), where we implement the entire univariate program of Newton form basis and divided-difference algorithm in $\mathbb{R}^n$. The required structure (triangular, rectangular, or a generalization) is needed only in the index set over any distinct coordinates in each variable (see Figure 2, for example). The index set structure appropriately determines the interpolation space. When the sum of indices is bounded by d ($d = 2$ in Figure 1(b)), there is unique interpolation with a polynomial of degree $\leq d$. When each variable x_i ranges over $d_i + 1$ fixed reals, there is unique interpolation in $\text{Span}\{x_1^{\mu_1} x_2^{\mu_2} \cdots x_n^{\mu_n} : \mu_i \in \{0, \ldots, d_i\}\}$, a tensor product space. All results of sections 3 to 5 are sometimes referred to simply as the tensor product case in modern surveys of multivariate interpolation [6]. Indeed, the basic ideas have been known since before computers ([23, p. 204], [10, p. 267], and [8, p. 296]; see [5]) but with scant modern exposition using multi-indices (as in [4]) or concise general algorithms as in this article. Most references treat only the bivariate case as in [16], which presents an error term theorem that supplements this exposition. While our focus is on algorithms, we prove that the multivariate divided-difference algorithm works by using a general existence and uniqueness theorem.

Interpolation is often used as a tool to derive numerical methods by replacing a function with a polynomial that agrees at created nodes. In this context, a required structure as discussed above could be part of the method design. This is precisely what was done in the application to automatic differentiation [12] that originally motivated this study. There, interpolation was used to find a multivariate Taylor polynomial (or multivariate derivative values), though stability problems were presented in [13].

In sections 6 to 8, we drop restrictions on geometric structure and consider arbitrary nodes in $\mathbb{R}^n$. Variations on Gaussian elimination are used to explain more recent algorithms in [18] and [14] that form different generalizations of the Newton basis. Exactly $\binom{n+d}{n}$ nodes are needed to try to specify a unique polynomial in P_d^n, the space of n-variable polynomials of degree $\leq d$ (since the number of unknown coefficients is $\binom{n+d}{n}$, as justified by a stars-and-bars combinatorial argument). The nodes are called *poised* (or *unisolvent*) if they support unique interpolation in P_d^n. In such a case, as in Figure 1(c), the algorithms succeed in finding the Newton-like basis for P_d^n that facilitates interpolation on the nodes. When not poised, the first algorithm finds a polynomial in P_d^n that is a zero on all the nodes, as in Figure 1(a). Geometrically, the nodes lie on this level surface, called an algebraic hypersurface. In section 7, we augment the algorithm to produce a minimal degree polynomial subspace within which unique interpolation on these nodes is possible. The algorithm may be used on any nodes to produce a minimal degree subspace [19], although there can be many such subspaces. In references, interpolation in P_d^n is often called the Lagrange interpolation problem [22], which is not to be confused with the Lagrange form, of an interpolating polynomial. We do not consider the Lagrange form, which has conceptual advantages over the Newton form but is computationally inferior [9].

We encounter some interesting geometric structures in nodes and some curious subspaces of polynomials, but only begin to explore the many interpolation questions along these lines that are asked. There are many different types of results about multivariate interpolation: results that describe other geometric structures in the nodes that guarantee unique interpolation; results on pairing polynomial subspaces with sets of nodes and gauging how far nodes are or can be from certain subspaces; results on interpolating derivatives; and those that give other algorithms for other forms, etc. (see [9, section 6.10], [6], [7]).

2. Univariate Interpolation.

A review of univariate Newton interpolation will establish notation that nicely generalizes to multiple variables. Take any distinct reals $x(0), x(1), \ldots, x(d)$ with any desired function values $f_0, f_1, \ldots, f_d$, respectively. (The $x(i)$ notation will free us to later use x_i and y as multiple domain variables and will emphasize how grid points can be viewed as functions of multiple index values.) The distinct domain values $x(i)$ are called *nodes* and need not be ordered or regularly spaced in any way. Each index $k \in \{0, \ldots, d\}$ is tied to a *classic univariate Newton polynomial*

$$q_k(x) = \prod_{i=0}^{k-1} (x - x(i)).$$

These polynomials form a basis for the space of polynomials of degree $\leq d$, such that $q_k(x(j)) = 0$ for $j < k$. The interpolating polynomial

$$p(x) = \sum_{i=0}^{d} a_i \, q_i(x) \tag{1}$$

has coefficients determined by a triangular linear system of equations $p(x(j)) = f_j$. The *divided-difference algorithm* to compute these coefficients is even more efficient than triangular forward substitution. We introduce the shortened notation $f[i, j]$ for the divided difference usually denoted $f[x(i), x(i+1), \ldots, x(j)]$. With initial values of $f[j, j] = f_j$, the first argument can be decremented by

$$f[i-1, j] = \frac{f[i, j] - f[i-1, j-1]}{x(j) - x(i-1)} \tag{2}$$

until reaching $f[0, j]$, which is the coefficient a_j in (1). (The common proof of this nonobvious fact will be presented in multivariate generality in section 5.) Although the notation shows that a triangular array of values is computed, they are usually computed by overwriting the original one-dimensional array of function values $\mathbf{a} = [f_0, f_1, \ldots, f_d]$. Formula (2) is applied in decreasing order of j, on d passes through the array, in the algorithm

```
for $k$ from 1 to $d$
    for $j$ from $d$ down to $k$
        $\mathbf{a}(j) = \dfrac{\mathbf{a}(j) - \mathbf{a}(j-1)}{x(j) - x(j-k)}$,
```

where for loops include the "to" values.

3. Multivariate Interpolation on a Grid.

3.1. Nodes and Polynomials.

For any finite set of nodes in $\mathbb{R}^n$, take each coordinate x_i and label all reals used in that coordinate $x_i(0), x_i(1), \ldots, x_i(\mu_i)$; we call these *tick marks*, even though no regularity or ordering is imposed. Think of taking any of the examples in Figure 1 and projecting onto the axes. To take advantage of structure, label them in order of how many times that coordinate value is used, as shown in Figure 2, an $\mathbb{R}^2$ example that we will follow throughout this section. Though we use coordinate axes, the algebraic properties that we use do not require orthogonal axes, so structure could be exploited or designed using any invertible affine transformation of $\mathbb{R}^n$.

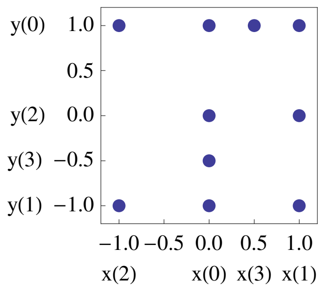

Fig. 2 *Example corner of nodes.*

Any combination of one tick mark in each coordinate makes a *grid point* in $\mathbb{R}^n$. All nodes are grid points, but not vice versa. In Figure 2, multi-index $\lambda = (2, 1)$ corresponds to node $\mathbf{x}((2, 1)) = (x(2), y(1)) = (-1, -1)$. A grid point in $\mathbb{R}^4$ could be $\mathbf{x}(0213) = (x_1(0), x_2(2), x_3(1), x_4(3))$, where the condensed notation 0213 is used, when clear from context, for the multi-index $\lambda = (0, 2, 1, 3)$. In general, a *multi-index* $\lambda \in \mathbb{N}_0^n$, i.e., $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n)$, where each $\lambda_i \in \{0, 1, 2, 3, \ldots\}$. Define $|\lambda| = \lambda_1 + \lambda_2 + \cdots + \lambda_n$ to be the *order* of the multi-index. Then $\lambda = (0, 2, 1, 3)$ has order 6 and we say $\mathbf{x}(0213)$ is a node of order 6. The notation $\lambda \leq \beta$ will mean coordinate-wise, i.e., $\lambda_i \leq \beta_i$ for all $i = 1, \ldots, n$.

For any multi-index λ that corresponds to some node $\mathbf{x}(\lambda)$, define the *classic Newton polynomial* by

$$q_\lambda(\mathbf{x}) = \prod_{m=1}^{n} \prod_{i=0}^{\lambda_m - 1} (x_m - x_m(i)). \tag{3}$$

Here, $\mathbf{x} = (x_1, \ldots, x_m, \ldots, x_n)$ are free variables and each $x_m(i)$ goes through all tick marks with $i < \lambda_m$. For multi-index $(2,1)$ in Figure 2, $q_{(2,1)}(x,y) = (x - x(0))(x - x(1))(y - y(0)) = x(x-1)(y-1)$. We seek $p(\mathbf{x}) = \sum_{\lambda \in I} a_\lambda q_\lambda(\mathbf{x})$ that interpolates (agrees with) nodes and function values $\{(\mathbf{x}(\lambda), f_\lambda) : \lambda \in I\}$ for some set of multi-indices $I \subseteq \mathbb{N}_0^n$. It is the index set I that distinguishes nodes from arbitrary grid points. Actually, any set of nodes with function values will have a unique interpolating polynomial in this (possibly uninteresting) span over the corresponding classic Newton polynomials. The coefficients are determined by a nonsingular triangular linear system, as will be shown in the proof of Theorem 2. However, with some structure in the index set, the span can be useful and the coefficients can be found by a multivariate divided-difference algorithm. For the corner of nodes in Figure 2, $I = \{(\lambda_1, \lambda_2) | \lambda_1 + \lambda_2 \leq 3\}$, the span is polynomials of degree ≤ 3 in two variables, and the algorithm detail will be shown in Table 1.

3.2. Structured Index Sets and Divided Differences.

Structured index sets allow us to apply the following generalization of the divided-difference formula:

$$f[\alpha - \mathbf{e}_m, \beta] = \frac{f[\alpha, \beta] - f[\alpha - \mathbf{e}_m, \beta - \mathbf{e}_m]}{x_m(\beta_m) - x_m(\alpha_m - 1)} \tag{4}$$

for multi-indices $\alpha \leq \beta$ with $\alpha_m > 0$, and $\mathbf{e}_m = (0, \ldots, 0, 1, 0, \ldots, 0)$ with 1 in the mth coordinate. To use (4), the index set for nodes and function values must include $\alpha - \mathbf{e}_m$ whenever it contains α.

Definition 1. *A finite set of multi-indices $J \subseteq \mathbb{N}_0^n$ is called a* lower set *if $(\beta \in J$ and $\mathbf{0} \leq \alpha \leq \beta) \implies \alpha \in J$. Important special cases are a* box *$J = \{\alpha : \mathbf{0} \leq \alpha \leq \beta\}$ for fixed β, and a* corner *$J = \{\alpha : |\alpha| \leq d\}$ for fixed order d. If a set of points in $\mathbb{R}^n$ is $\{\mathbf{x}(\lambda) : \lambda \in J\}$ for some lower set (box, corner) J, then we call it a* lower set of nodes *(*box of nodes*, or* corner of nodes*, respectively).*

The interpolation space spanned by q_λ over a corner for order d is P_d^n, the space of n-variable polynomials of degree $\leq d$. The space spanned by q_λ over a box up through β is the classic tensor product space $\mathrm{Span}\{x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n} : \mathbf{0} \leq \alpha \leq \beta\}$. (In [4], a lower set is seen to be a union of overlapping boxes (blocks) where interpolation is a combination of interpolation over boxes. For a lower set of nodes, [20] shows that any reasonable interpolation space amounts to a tensor product space spanned by corresponding monomials.) To evaluate a polynomial $p = \sum_{\lambda \in J} a_\lambda q_\lambda$ for a lower set J, nesting of common factors enables a multivariate Horner scheme (see [25] for a two-variable implementation, or very generally [15] and [2]). We now focus on computing the coefficients a_λ.

For $\{(\mathbf{x}(\lambda), f_\lambda) : \lambda \in J\}$, where J is a lower set, initialize each $f[\lambda, \lambda] = f_\lambda$ and use (4) to work the first argument down to $f[\mathbf{0}, \lambda]$, which is the coefficient a_λ in the interpolating polynomial $p(\mathbf{x}) = \sum_{\lambda \in J} a_\lambda q_\lambda(\mathbf{x})$ as proved in section 5. There are many options for the particular $\mathbf{e}_m$ to subtract in (4) and in which order to process entries from J, but a consistent scheme must be developed. The example $\beta = (1,2,1)$ requires subtracting $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_2$, and $\mathbf{e}_3$, which we do in this (left-to-right) order, so the only α's ever needed in $f[\alpha, \beta]$ are $(1,2,1), (0,2,1), (0,1,1), (0,0,1)$, and $(0,0,0)$. Other consistent decrementing schemes work just as well, resulting in different intermediate divided differences. (Even combinations of these schemes could be used as in [24].) Next, all divided differences are computed by overwriting the original array of function values $A[\lambda] = f[\lambda, \lambda] = f_\lambda$ for $\lambda \in J$, so that $A[(1,2,1)]$ will be changed four times through the divided differences mentioned above. To process all entries $\lambda \in J$ in multiple passes through array A, we show a two-variable example and then state arbitrary algorithms.

3.3. Two-Variable Interpolation Example. The nodes in Figure 2 provide a specific example indexed by a corner in $n = 2$ variables up through order $d = 3$. We seek the interpolating polynomial

$$p(x,y) = \sum_{\lambda_1+\lambda_2 \le 3} a_{(\lambda_1,\lambda_2)} q_{(\lambda_1,\lambda_2)}(x,y).$$

The desired function values are given in Table 1(a), which forms the initialization of the array A. The right half of Table 1 shows the divided difference that will be done in pass k through the array, based on multi-index locations given by (λ_1, λ_2) in decreasing order, so progressing down diagonals. At multi-index $\beta = (2,1)$, the first and second passes decrement by $\mathbf{e}_m = \mathbf{e}_1$ indicated by a left-arrow (subtracting the adjacent table value in the direction of the arrow), while the third pass decrements $\mathbf{e}_m = \mathbf{e}_2$ indicated by a down-arrow. The counter t, next to each arrow, is the number

Table 1 *Decreasing-order algorithm: each $\leftarrow t$ or $\downarrow t$ indicates a divided difference subtraction in the indicated direction over the span of t in that coordinate.*

λ_2	$y(\lambda_2)$	(a) function values				λ_2	pass $k=1$			
3	-0.5	2.75				3	$\downarrow 1$			
2	0	3	3			2	$\downarrow 1$	$\leftarrow 1$		
1	-1	9	10	16		1	$\downarrow 1$	$\leftarrow 1$	$\leftarrow 1$	
0	1	5	8	2	4.25	0		$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$
	$x(\lambda_1)$:	0	1	-1	0.5					
	λ_1 :	0	1	2	3	λ_1 :	0	1	2	3

λ_2	$y(\lambda_2)$	(b) after pass 1				λ_2	pass $k=2$			
3	-0.5	0.5				3	$\downarrow 2$			
2	0	-6	0			2	$\downarrow 2$	$\downarrow 1$		
1	-1	-2	1	-3		1		$\downarrow 1$	$\leftarrow 2$	
0	1	5	3	3	1.5	0			$\leftarrow 2$	$\leftarrow 2$
	$x(\lambda_1)$:	0	1	-1	0.5					
	λ_1 :	0	1	2	3	λ_1 :	0	1	2	3

λ_2	$y(\lambda_2)$	(c) after pass 2				λ_2	pass $k=3$			
3	-0.5	13				3	$\downarrow 3$			
2	0	4	-1			2		$\downarrow 2$		
1	-1	-2	1	4		1			$\downarrow 1$	
0	1	5	3	0	3	0				$\leftarrow 3$
	$x(\lambda_1)$:	0	1	-1	0.5					
	λ_1 :	0	1	2	3	λ_1 :	0	1	2	3

λ_2	$y(\lambda_2)$	(d) after pass 3			
3	-0.5	-6			
2	0	4	2		
1	-1	-2	1	-2	
0	1	5	3	0	6
	$x(\lambda_1)$:	0	1	-1	0.5
	λ_1 :	0	1	2	3

of times that the arrow has been in that direction at that location, and tells how far down to step for the difference of node coordinates in the denominator. For example, in pass $k = 2$ at location $(2, 1)$, the $\leftarrow 2$ describes the operation

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

which results in $f[(0,1),(2,1)]$. When overwriting, the operation for any arrow must be done before the operation that the arrow points to, which is clearly accomplished by proceeding in decreasing order $|\lambda| = \lambda_1 + \lambda_2$ (within fixed order doesn't matter). We call this the decreasing-order algorithm. Alternatively, a coordinate-wise algorithm would use six passes through the array, computing sequentially all $\leftarrow 1$, all $\leftarrow 2$, and $\leftarrow 3$, then all $\downarrow 1$, all $\downarrow 2$, and $\downarrow 3$, where within a pass $\leftarrow$ are done right-to-left and $\downarrow$ top-to-bottom.

The result is a nested set of interpolating polynomials, reading coefficients in Table 1 up the finished diagonal after each decreasing-order 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, $z = p_1(x,y)$ describes the plane through points $(0,1,5), (1,1,8), (0,-1,9)$. The quadric surface given by $z = p_2(x,y)$ is a hyperbolic paraboloid (saddle surface) through values on the six corner nodes of order ≤ 2. The unique cubic polynomial through all ten points may be multiplied out to find the standard basis form,

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

Interpolation on the subset box of nodes up through $\beta = (2,1)$ needs only that subset of the table (more naturally done by the coordinate-wise algorithm). The resulting six values give

$$\begin{aligned} p(x,y) &= 5 + 3x + 0x(x-1) + (-2 + 1x - 2x(x-1))(y-1) \\ &= 7 - 2y + 3xy + 2x^2 - 2x^2y, \end{aligned}$$

the unique tensor product space polynomial through these six points using powers of x up to 2 and powers of y up to 1.

3.4. General Divided-Difference Algorithms. We now generalize for a lower set of nodes in $\mathbb{R}^n$ with maximum multi-index order d. The *decreasing-order algorithm* [12] for divided differences is

```
for $k$ from 1 to $d$
    for $j$ from $d$ down to $k$
        for all $\lambda$ in $J$ with $|\lambda| = j$
            $m$ = largest $m$ such that $t = k - \left(\sum_{i=1}^{m-1} \lambda_i\right) > 0$;
            $A[\lambda] = \dfrac{A[\lambda] - A[\lambda - \mathbf{e}_m]}{x_m(\lambda_m) - x_m(\lambda_m - t)}$.
```

Since $\lambda - \mathbf{e}_m$ is of lower order, the required values of $A[\lambda - \mathbf{e}_m]$ exist and are used before being overwritten. This algorithm is more natural for a corner of nodes in $\mathbb{R}^n$ up through order d. Instead of a wasteful n-dimensional array, A can be stored in a linear array, corresponding to a *linear ordering* of the n-dimensional multi-indices. We say that α precedes β if

$$|\alpha| < |\beta| \text{ or } (\,|\alpha| = |\beta| \text{ and } \alpha_k > \beta_k \text{ while } \alpha_j = \beta_j \text{ for } j < k). \tag{5}$$

The first condition is natural in this context, but the second is an arbitrary choice. For example, linear indices corresponding to the array in Table 1 are

$$\begin{array}{rrrr} 10 & & & \\ 6 & 9 & & \\ 3 & 5 & 8 & \\ 1 & 2 & 4 & 7 \end{array}$$

although such a spatial array is not stored. A reference array is stored for the multi-index corresponding to each linear index, which is helpful for any computation or application, giving access to the coordinates λ_i. The j and λ loops, in the general decreasing-order algorithm, can be combined by simply decrementing the linear index from the end down to $\binom{n+k-1}{n} + 1$, assuming linear index origin 1. Multi-index references in this algorithm can be handled by linear indices: $A[\lambda]$ is simply A at the current linear index, call it d_0, but $A[\lambda - \mathbf{e}_m]$ requires combinatorics to find its linear index d_m. Specifically, $d_m = d_{m-1} - \binom{n-m+s_m}{n-m}$, where $s_m = \lambda_m + \lambda_{m+1} + \cdots + \lambda_n - 1$, as is argued in [13] for a different application. Our bivariate example $\lambda = (2,1)$ has linear index $d_0 = 8$, so $d_1 = 8 - \binom{2-1+2}{2-1} = 5$ and $d_2 = 5 - \binom{2-2+0}{2-2} = 4$ for decrementing by $m = 1$ (left-arrow) or $m = 2$ (down-arrow), respectively. All references and index computations could be done once for dimension n and stored in reference arrays keyed only by linear index and pass number k.

The alternative *coordinate-wise algorithm* simply works left-to-right, making all changes in A for each coordinate before proceeding to the next. Using the maximum in each coordinate $\mu_m = \max\{\lambda_m : \lambda \in J\}$, the algorithm (as done explicitly for a two-variable corner in [25]) is

```
for $m$ from 1 to $n$
    for $k$ from 1 to $\mu_m$
        for $j$ from $\mu_m$ down to $k$
            for all $\lambda$ in $J$ with $\lambda_m = j$
                $A[\lambda] = \dfrac{A[\lambda] - A[\lambda - \mathbf{e}_m]}{x_m(j) - x_m(j-k)}$.
```

Since $\lambda_m = j$ decreases (and all previous coordinates are finished), the required values of $A[\lambda - \mathbf{e}_m]$ exist and are used before being overwritten. This algorithm works for any lower set J and requires iterating through all entries in J with fixed mth coordinate, which may or may not be easy. Values at a box of nodes up through β (so $\mu = \beta$) could be stored in a true n-dimensional array, so the above coordinate-wise algorithm would be easy, especially if the dimension n is hard-coded into the program.

4. Existence and Uniqueness on a Multivariate Grid.

A very general existence and uniqueness result over some span will include important special cases for a corner of nodes and a box of nodes (used to prove the divided-difference formula). The result actually applies to any set of nodes whatsoever, not just a lower set. Any finite set of nodes can be described by $\{\mathbf{x}(\lambda) : \lambda \in I\}$ for some tick marks and some $I \subseteq \mathbb{N}_0^n$. The tick marks $x_i(j)$ are reals that are distinct in each variable x_i, though often irregular and unordered. For each $\lambda \in I$, (3) defines q_λ that has factors $(x_i - x_i(j))$ for all $j < \lambda_i$, even though the grid points $\mathbf{x}(\mu)$ with $\mu \le \lambda$ may not be in the node set $(\mu \notin I)$. Then

$$q_\lambda(\mathbf{x}(\mu)) = 0 \text{ if and only if } \exists m \in \{1, \dots, n\} \text{ such that } \mu_m < \lambda_m, \tag{6}$$

which results in a triangular linear system as seen in the following proof.

THEOREM 2 (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* $\operatorname{Span}\{q_\lambda : \lambda \in I\}$.

Proof. Suppose $p = \sum_{\lambda \in I} a_\lambda q_\lambda \equiv 0$. The equations

$$p(\mathbf{x}(\mu)) = \sum_{\lambda \in I} a_\lambda q_\lambda(\mathbf{x}(\mu)) \equiv 0 \text{ for all } \mu \in I$$

may be written in matrix form using the linear ordering (5) of the multi-indices in I. Define square matrix $G = [q_\lambda(\mathbf{x}(\mu))]$ with row index μ and column index λ. The equations become $G[a_\lambda] = \mathbf{0}$, where $[a_\lambda]$ and $\mathbf{0}$ are column vectors indexed by λ. To see that G is lower-triangular, consider a row index μ that precedes a column index λ. Since $|\mu| \le |\lambda|$ and $\mu \ne \lambda$, then some $\mu_m < \lambda_m$, so that $q_\lambda(\mathbf{x}(\mu)) = 0$ by (6). Along the diagonal, $q_\mu(\mathbf{x}(\mu)) \ne 0$. Thus, G is lower-triangular and invertible, so $[a_\lambda] = \mathbf{0}$ and $\{q_\lambda : \lambda \in I\}$ is linearly independent. An interpolating polynomial must satisfy $p(\mathbf{x}(\mu)) = \sum_{\lambda \in I} a_\lambda q_\lambda(\mathbf{x}(\mu)) = f_\mu$ for all $\mu \in I$, so $[a_\lambda] = G^{-1}[f_\mu]$. ∎

Actually, the proof shows that matrix G for the linear system has a block structure even better than invertible triangular. For $|\mu| \le |\lambda|$, $q_\lambda(\mathbf{x}(\mu)) \ne 0$ if and only if $\mu = \lambda$. Thus, every diagonal block, corresponding to multi-indices of the same order, is a diagonal matrix with nonzero diagonal entries. One consequence is that if another node is added, hence another $\beta \in I$ and another q_β, then the new interpolating polynomial will not change any previous terms with order $|\lambda| \le |\beta|$, an important feature of the classic Newton basis. Properties of the basis, which we will later try to generalize, include the following: each q_λ is a polynomial that

$$\begin{aligned} &\text{(a) is zero at nodes of lower order,}\\ &\text{(b) is nonzero at node } \mathbf{x}(\lambda) \text{ and}\\ &\qquad \text{zero at other nodes of the same order, and}\\ &\text{(c) has degree } |\lambda| \text{ and exactly one term of degree } |\lambda|. \end{aligned} \tag{7}$$

COROLLARY 3. *For $\{(\mathbf{x}(\lambda), f_\lambda) : \alpha \le \lambda \le \beta\}$, there exists a unique interpolating polynomial in*

$$\begin{aligned} &\operatorname{Span}\left\{ \prod_{m=1}^{n} \prod_{i=\alpha_m}^{\lambda_m - 1} (x_m - x_m(i)) : \alpha \le \lambda \le \beta \right\} \\ &= \operatorname{Span}\{x_1^{\mu_1} x_2^{\mu_2} \cdots x_n^{\mu_n} : \mathbf{0} \le \mu \le \beta - \alpha\}. \end{aligned} \tag{8}$$

This tensor product case is an immediate corollary of Theorem 2 with the origin shifted to α, and it is the key to proving the divided-difference formula for coefficients. We will repeatedly use the property that the unique interpolating polynomial for function values on such a *subbox of nodes* has exactly one term of highest degree $|\beta - \alpha|$, so the coefficient of the highest power basis polynomial is the same using either of the bases in (8).

The catch to the generality of Theorem 2 is that $\text{Span}\{q_\lambda : \lambda \in I\}$ may or may not be useful. For example, six nodes on a circle in $\mathbb{R}^2$ do not allow unique interpolation by a quadratic polynomial. However, 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 rectangular corners) and $y(2) = 0$, $x(2) = -2$, $x(3) = 2$ (two more nodes on the x-axis). For these grid points, the associated multi-indices and classic Newton polynomials are $q_{00}, q_{10}, q_{01}, q_{11}$ along with $q_{22}(x, y) = (x - x(0))(x - x(1))(y - y(0))(y - y(1))$ 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 [3] 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 sufficient to interpolate any six nodes, even if they lie on a straight line (where degree 5 is necessary). In section 7, we show one way to find a subspace of polynomials of minimal degree 3 that is correct for the circle of nodes.

5. The Divided-Difference Formula.

To understand why divided differences work, we provide a direct proof for arbitrary n variables (that generalizes, rather than assumes, a standard univariate proof). Fix nodes and values $\{(\mathbf{x}(\lambda), f_\lambda) : \lambda \in J\}$ over some lower set of multi-indices J. To establish that the divided-difference formula (4) actually produces coefficients of the interpolating polynomial, we turn it around and define the notation to be the coefficient and then show that it satisfies the formula, a common univariate proof approach. Specifically, Corollary 3 justifies the following notation.

DEFINITION 4. *For $\alpha, \beta \in J$ with $\alpha \leq \beta$, define $f[\alpha, \beta]$ to be the coefficient of the highest power in the interpolating polynomial for nodes and values $\{(\mathbf{x}(\lambda), f_\lambda) : \alpha \leq \lambda \leq \beta\}$.*

So $f[\alpha, \beta]$ is the unique coefficient of $x_1^{\mu_1} x_2^{\mu_2} \cdots x_n^{\mu_n}$, where $\mu = \beta - \alpha$, which equals the unique coefficient of the highest order term in the Newton form of the polynomial in (8). In traditional divided-difference notation [8, p. 296], this coefficient $f[\alpha, \beta]$ would be written as

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

which would be much more cumbersome. The multi-index notation facilitates this exposition. Assume the same lower set of nodes with values throughout this section.

THEOREM 5. *The polynomial $p(\mathbf{x}) = \sum_{\lambda \in J} f[\mathbf{0}, \lambda] q_\lambda(\mathbf{x})$ satisfies $p(\mathbf{x}(\lambda)) = f_\lambda$ for all $\lambda \in J$.*

Proof. By Theorem 2, there exists a unique interpolating polynomial of the form $\widehat{p}(\mathbf{x}) = \sum_{\lambda \in J} a_\lambda q_\lambda(\mathbf{x})$. Let $\mu \in J$. We need only show that $a_\mu = f[\mathbf{0}, \mu]$. For this fixed μ,

$$f_\mu = \widehat{p}(\mathbf{x}(\mu)) = \sum_{\lambda \leq \mu} a_\lambda q_\lambda(\mathbf{x}(\mu)) + \sum_{\lambda \in J,\ \lambda \nleq \mu} a_\lambda q_\lambda(\mathbf{x}(\mu)).$$

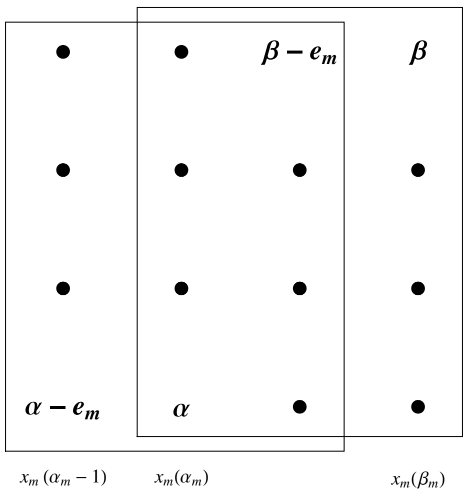

Fig. 3 *Multi-indices in proof of divided difference.*

By (6), the second sum is zero. Thus, the first sum is the interpolating polynomial for the box $\{(\mathbf{x}(\lambda), f_\lambda) : \mathbf{0} \le \lambda \le \mu\}$ and $a_\mu = f[\mathbf{0}, \mu]$ by definition. ∎

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

THEOREM 6. *For $\alpha, \beta \in J$ with $\alpha \le \beta$ and $\alpha_m > 0$,*

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

where $\mathbf{e}_m = (0, \ldots, 0, 1, 0, \ldots, 0)$ with 1 in the mth coordinate.

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

$$p(\mathbf{x}) = p_-(\mathbf{x}) + \left(\frac{x_m - x_m(\alpha_m - 1)}{x_m(\beta_m) - x_m(\alpha_m - 1)}\right)(p_+(\mathbf{x}) - p_-(\mathbf{x})). \tag{9}$$

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

$$p(\mathbf{x}(\lambda)) = p_-(\mathbf{x}(\lambda)) + (1)\,(p_+(\mathbf{x}(\lambda)) - p_-(\mathbf{x}(\lambda))) = p_+(\mathbf{x}(\lambda)) = f_\lambda.$$

We conclude that $p(\mathbf{x})$ is the interpolating polynomial over the union and, by definition, the coefficient of the highest power is $f[\alpha - \mathbf{e}_m, \beta]$. This highest power is $x_1^{\mu_1} x_2^{\mu_2} \cdots x_n^{\mu_n}$ for $\mu = \beta - \alpha + \mathbf{e_m}$, which occurs in (9) from the linear factor times the highest power term of $(p_+(\mathbf{x}) - p_-(\mathbf{x}))$. The coefficient of the highest power in $p_+(\mathbf{x})$ is $f[\alpha, \beta]$ and in $p_-(\mathbf{x})$ is $f[\alpha - \mathbf{e}_m, \beta - \mathbf{e}_m]$. The coefficient of the variable x_m in the linear factor is $1/(x_m(\beta_m) - x_m(\alpha_m - 1))$. Thus, the coefficient of the highest power on the right side of (9) is the desired divided difference formula. ∎

As an alternative to the next section, [17] gives one generalization of the concept of divided differences to any $\binom{n+d}{n}$ poised nodes. The result gives coefficients of our Newton–Olver generalization, in section 8, of the classic Newton basis. For a corner of nodes, the divided difference in [17] matches our classic tensor product divided difference above, as shown in [1].

6. Polynomials for Arbitrary Nodes.

We now leave grid points behind and determine to what extent a Newton-like basis, up through degree d, can be found for an arbitrary set of $\binom{n+d}{n}$ nodes in $\mathbb{R}^n$. A standard linear algebra approach can be modified to construct the basis or find an algebraic hypersurface containing the nodes and show that it is impossible. Consider the problem of interpolation in standard monomial basis form, as in the example

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

for $n = 2$ and $d = 2$ and six nodes with function values. The interpolation equations form a linear system using a matrix M times unknown coefficients; M consists of powers of node coordinates in a multivariate generalization of the Vandermonde matrix. If M is singular (nodes are not poised), then there cannot be a unique solution. If M is invertible (nodes are poised), then Gaussian elimination can produce $M = LU$ (maybe with a permutation) so that the monomial coefficients can be found with two triangular system solves for each given set of function values at the nodes. If the nodes are poised, a similar process on $V = M^T$ will produce (instead of LU) coefficients of a new basis, found by Sauer in [18], that shares many of the properties of a Newton basis. Then one (easier than) triangular solve will find coefficients of this basis for each given set of function values at the nodes. If not poised, the process will produce coefficients of a polynomial that is zero on all nodes (an algebraic hypersurface containing the nodes). Then, extending V can produce a minimal degree subspace as in section 7.

For nodes $\{\mathbf{x}_j\}$ for $j = 1, \ldots, \binom{n+d}{n}$ with each $\mathbf{x}_j \in \mathbb{R}^n$, define square matrix $V = [(\mathbf{x}_j)^\lambda]$, where columns are indexed by j and rows by λ, with $|\lambda| \le d$. The multi-index λ indicates monomial powers in each variable. Since the nodes are not considered as grid points, we use the linear index j. Still, nodes are now associated with a multi-index multi(j) starting with multi$(1) = (0, 0, \ldots, 0)$ and continuing consecutively to match the linear ordering (5) of multi-indices, and we can even define the *order of a node* as $|\text{multi}(j)|$. For example, if $\mathbf{x}_j = (a_j, b_j)$ for $n = 2$ and $d = 2$,

$$V = \left[\begin{array}{c|cc|ccc} 1 & 1 & 1 & 1 & 1 & 1 \\ \hline a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \\ b_1 & b_2 & b_3 & b_4 & b_5 & b_6 \\ \hline a_1^2 & a_2^2 & a_3^2 & a_4^2 & a_5^2 & a_6^2 \\ a_1b_1 & a_2b_2 & a_3b_3 & a_4b_4 & a_5b_5 & a_6b_6 \\ b_1^2 & b_2^2 & b_3^2 & b_4^2 & b_5^2 & b_6^2 \end{array}\right], \tag{11}$$

where lines group blocks according to the order of the associated multi-index. The linear system to directly solve for coefficients in (10) would be $\mathbf{c}V = \mathbf{f}$, for row vectors $\mathbf{c} = [c_{00}, c_{10}, c_{01}, c_{20}, c_{11}, c_{02}]$ and $\mathbf{f} = [f_1, f_2, f_3, f_4, f_5, f_6]$, where f_j is the value at node $\mathbf{x}_j$. However, we will perform Gaussian elimination steps directly on V and find an upper-triangular matrix W and lower-triangular matrix R, where R defines polynomials in a generalized Newton basis and where simply solving $\mathbf{a}W = \mathbf{f}$ will give the coefficients of the interpolating polynomial in this basis.

We illustrate with the specific example of nodes from Figure 1(c). The V matrix is augmented with an identity matrix, forming

$$\begin{array}{c|cc|ccc||c|cc|ccc}
\multicolumn{6}{c||}{\text{nodes}} & 1 & x & y & x^2 & xy & y^2 \\
\hline\hline
1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 1 & 2 & 2 & -1 & -2 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & -1 & 1 & 2 & 2 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
\hline
0 & 1 & 4 & 4 & 1 & 4 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & -1 & 2 & 4 & -2 & -2 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 4 & 4 & 1 & 0 & 0 & 0 & 0 & 0 & 1
\end{array} \quad . \tag{12}$$

Each column in the left half corresponds to a node with coordinates initially shown in rows 2 and 3 (2 through $n+1$ in general). Each column in the right half corresponds to a standard basis monomial as labeled in the example. Initially, the rows also correspond to these monomials. Again, we block all rows and columns according to the order of the associated multi-index.

PROPOSITION 7. *For every row of $[V \mid I]$, 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 V. For an arbitrary row operation, suppose we add -3 times the third row into the fourth row in (12). 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 of 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 Proposition 7. In fact, row exchanges are never done, as is 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. If successful, column exchanges may be needed to reorder the nodes so that they are *poised in block*. 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 example (12), without any exchanges, to $[W \mid R]$:

$$\begin{array}{c|cc|ccc||c|cc|ccc}
\multicolumn{6}{c||}{\text{nodes}} & 1 & x & y & x^2 & xy & y^2 \\
\hline\hline
1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 1 & 0 & \frac{-2}{3} & \frac{-5}{3} & \frac{-4}{3} & 0 & \frac{1}{3} & \frac{-2}{3} & 0 & 0 & 0 \\
0 & 0 & 1 & \frac{4}{3} & \frac{1}{3} & \frac{-1}{3} & 0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 1 & 0 & 0 & 0 & \frac{-4}{13} & \frac{-9}{26} & \frac{1}{39} & \frac{4}{13} & \frac{19}{78} \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & \frac{3}{13} & \frac{5}{13} & \frac{-4}{39} & \frac{-3}{13} & \frac{1}{39} \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & \frac{-17}{52} & \frac{-11}{52} & \frac{9}{52} & \frac{1}{13} & \frac{1}{52}
\end{array} \quad .$$

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

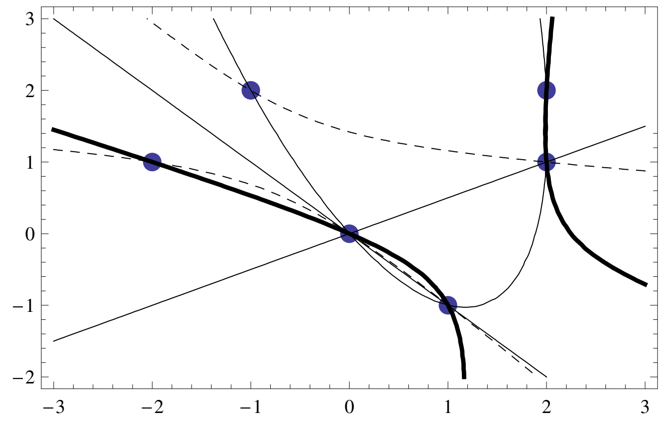

Fig. 4 *Zero contours of Newton–Sauer polynomials.*

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 (7), 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 and not in factored form.

We call $\{r_\lambda\}$ the *Newton–Sauer polynomials*, since they are the Newton fundamental polynomials produced in [18], which builds on [22]; see those papers for much more detail about their usefulness and effectiveness. Our row operations with column exchanges are equivalent to the steps in [18, 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 [18, Algorithm 4.4], which indicates 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 the implementation of LU-decomposition, by storing the lower-triangular values below upper-triangular values. The process $[V\,|\,I] \to [W\,|\,R]$ 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. (A triangular factorization of permuted V is given by $V\,P = R^{-1}\,W$.)

To finish the example, we find the interpolating polynomial for values, say, $\mathbf{f} = [5, 6, 7, 8, 9, 10]$, at the six nodes in Figure 4. Values should match the ordering of nodes after any necessary column exchanges (of which there were none in the example). Since W holds values of the Newton–Sauer polynomials at the nodes, $\mathbf{a}W = \mathbf{f}$ describes coefficients of the interpolating polynomial in this basis. This is a particularly easy triangular system. Here, forward substitution (in row or block forms) may be used and results in $\mathbf{a} = [5, 1, 2, 1, 5, 7]$. 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 when nodes are not poised, such as the circle of nodes in Figure 1(a). The nodes are put into V 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 $[V \,|\, I]$ 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 when we are left with

$$\begin{array}{c|cc|ccc||c|cc|ccc}
\multicolumn{6}{c||}{\text{circle of nodes}} & 1 & x & y & x^2 & xy & y^2 \\
\hline\hline
1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 1 & 0 & \frac{-1}{2} & 1 & \frac{3}{2} & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & \frac{1}{2} & 1 & \frac{1}{2} & \frac{1}{2} & 0 & \frac{\sqrt{3}}{6} & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 1 & 0 & 1 & \frac{-1}{3} & 0 & 0 & \frac{1}{3} & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & \frac{1}{6} & \frac{1}{4} & \frac{\sqrt{3}}{12} & \frac{1}{12} & \frac{\sqrt{3}}{12} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & 0 & 1 & 0 & 1
\end{array} \qquad (13)$$

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. If successful, we find all nonzero pivots and a basis of P_d^n that is convenient for building function values on the given nodes. (In fact, a change in function value f_j affects only coefficient c_μ, where $\text{multi}(j) = \mu$ and coefficients of higher order.) If unsuccessful, we find an algebraic hypersurface containing all the nodes.

THEOREM 8. *Given a set of $\binom{n+d}{n}$ nodes $\{\mathbf{x}_j\}$ in n variables, form $V = [(\mathbf{x}_j)^\lambda]$ with rows indexed by λ and augmented $[V \,|\, I]$. 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 $\mathbf{x}_j$ (after column exchanges) and Newton–Sauer polynomials $r_\lambda(\mathbf{x})$ of degree $|\lambda|$ given by coefficients (in terms of the standard monomial basis) in each row λ of R such that*

$$r_\lambda(\mathbf{x}_j) = \begin{cases} 0 & \textit{if } |\text{multi}(j)| \le |\lambda| \textit{ and } \text{multi}(j) \ne \lambda \\ 1 & \textit{if } \text{multi}(j) = \lambda \end{cases} \quad ; \textit{ or}$$

(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 $r_\nu(\mathbf{x}_j) = 0$ for all nodes.*

7. Interpolation Subspace of Minimal Degree. Even when nodes are not poised, as in Theorem 8(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, simply delete that row and augment by a new row and column corresponding to the next monomial in the standard basis. For the circle of nodes, the next monomial is x^3 and the left-half row is given values of x^3 at the nodes. We also delete the right-half column corresponding to the deleted row since it is not needed (as would be shown by a column of zeros if it were retained). The new last row in our example would be

circle of nodes						1	x	y	x^2	xy	x^3
-1	1	-1	-8	1	8	0	0	0	0	0	1

The desired identity blocks in W are shrunken on each deletion, so they correspond to the degree of the monomials involved. Thus, our previous five rows of (13) remain unchanged and we do row additions into the last row to result in

circle of nodes						1	x	y	x^2	xy	x^3
0	0	0	0	0	1	$\frac{-1}{6}$	$\frac{-1}{12}$	0	$\frac{1}{6}$	0	$\frac{1}{12}$

So, for any function values on the circle of nodes, there exists a unique interpolating polynomial in $\text{Span}\{1, x, y, x^2, xy, x^3\}$ which equals the span of the Newton–Sauer polynomials given by the right-half matrix R with new last row. Due to the ordering, which took advantage of the common coordinates in this example, these Newton–Sauer polynomials factor nicely into

$$\begin{aligned}
&1,\\
&\tfrac{1}{2}(x+1),\\
&\tfrac{\sqrt{3}}{6}(y+\sqrt{3}),\\
&\tfrac{1}{3}(x+1)(x-1),\\
&\tfrac{1}{12}(x+1)(x+\sqrt{3}y+2),\\
&\tfrac{1}{12}(x+1)(x-1)(x+2).
\end{aligned}$$

This extended algorithm will work on any number of nodes to produce a polynomial subspace of minimal degree that always contains the unique interpolating polynomial for values on the nodes. It is equivalent to, and gives the same results as, the algorithm in [19], which is not described in terms of row operations. Often, m nodes will not produce a row of zeros, so only the first m standard basis monomials are required. However, one may need to repeatedly delete rows and augment. The extreme example would be $d+1$ nodes along a line, in which case the extended algorithm would delete all rows corresponding to monomials using variables other than x (assuming the line is not orthogonal to the x-axis) and generate the classic one-dimensional Newton polynomials up to degree d, normalized. In any case, the resulting Newton–Sauer basis for the subspace will produce nested interpolating polynomials. In particular, the first k Newton–Sauer basis terms of an interpolating polynomial will interpolate the first k nodes in the final ordering, since W is always triangular.

8. Block Gaussian Elimination.

We now vary the algorithm that created the Newton–Sauer polynomials in order to obtain a different basis that recovers some of the algebraic simplicity of the classic Newton basis and also recaptures something like a divided difference algorithm. Even for a corner of nodes corresponding to the classic Newton polynomials, the algorithm of section 6 rescales them by normalizing the diagonal entries to 1 in the Gaussian elimination. The idea is to avoid this normalization, thinking in terms of blocks. In the one-dimensional case, a scalar diagonal entry would be either normalized to the multiplicative identity 1 or left nonzero so that we may divide by it. For multivariable diagonal blocks, this suggests either an identity matrix as in section 6 or an invertible matrix as in this section and in the paper by Olver [14]. The intermediate goal of upper-triangular is another reasonable choice. The choices represent a trade-off between which properties of the classic polynomials (7) 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 $[V \,|\, I]$ matrix (12) 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

$$\begin{array}{c|cc|ccc||c|cc|ccc} \multicolumn{6}{c||}{\text{nodes}} & 1 & x & y & x^2 & xy & y^2 \\ \hline\hline 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 2 & 2 & -1 & -2 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 2 & 2 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & \frac{-2}{3} & \frac{4}{3} & \frac{20}{3} & 0 & \frac{-5}{3} & \frac{-2}{3} & 1 & 0 & 0 \\ 0 & 0 & 0 & \frac{2}{3} & \frac{-13}{3} & \frac{-8}{3} & 0 & \frac{-1}{3} & \frac{-4}{3} & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{10}{3} & \frac{16}{3} & \frac{8}{3} & 0 & \frac{-2}{3} & \frac{1}{3} & 0 & 0 & 1 \end{array} \quad . \tag{14}$$

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*

$$1,\ x,\ y,\ \frac{-1}{3}(5x + 2y) + x^2,\ \frac{-1}{3}(x + 4y) + xy,\ \frac{-1}{3}(2x - y) + y^2. \tag{15}$$

Compared to properties of the classic polynomials (7), property (b) concerning values at nodes of the same order is essentially lost in favor of regaining property (c): each Newton–Olver polynomial has degree $|\lambda|$ and the only term of that degree is $1\mathbf{x}^{\lambda}$.

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 $[V \,|\, I]$ matrix and reduce to a block-upper-triangular $[A \,|\, S]$ with invertible matrices on the diagonal, where $m_k = \binom{n+k-1}{k}$ is the size of the block for order k. Name the resulting blocks, as done here for $d = 3$:

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

This end is achieved by block Gaussian elimination, in which every step is pivoting on block A_{kk}: assume all rows above and columns to left of A_{kk} are done, group all remaining parts of the augmented matrix into $\left[\begin{smallmatrix} A_{kk} & B \\ C & D \end{smallmatrix}\right]$, and replace D by $D - CA_{kk}^{-1}B$. From D, define the next pivoting block of size m_{k+1} and repeat. The algorithm assumes that nodes are poised and ordered so that they are poised in block, so each A_{kk} is invertible. Otherwise, it may be more practical to switch to the row operations of section 6 in order to find the necessary column exchanges or find the hypersurface containing the nodes.

The Newton–Olver name reflects that this process is equivalent to the recursive algorithm described in [14, Theorem 5]. Their notation $\Omega_k^i(X_j)$ denotes row block i and column block j on the kth 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. In fact, [14] includes binomial coefficients with the monomials in the original generalization of the Vandermonde matrix V, 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 A holds values of the Newton–Olver basis at the nodes. Let the desired node values $\mathbf{f}$ and coefficients $\mathbf{a}$ be row vectors and solve $\mathbf{a}A = \mathbf{f}$. 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 $\mathbf{a}_k = \mathbf{a}_k - \mathbf{a}_i A_{i\,k}$ for $i = 0, \dots, k-1$ and then "divide" $\mathbf{a}_k = \mathbf{a}_k A_{kk}^{-1}$. The equivalent algorithm in [14, equation (75)] is called "a full multivariate version of the classical recursive divided difference scheme." For the generic example in (14) and initial $\mathbf{f} = [[5], [6, 7], [8, 9, 10]]$, we obtain

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

Thus, the unique interpolating polynomial using the basis in **??** is

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

For a corner of nodes in multi-index ordering, the Newton–Olver polynomials are exactly the classic Newton polynomials and all $A_{i\,k}$ are the tick-mark differences as in the $[q_\lambda(\mathbf{x}(\mu))]$ matrix for the classic Newton system. Again, uniqueness demands this. If the corner has a classic Newton $q_\lambda(\mathbf{x})$ and the Newton–Olver polynomial 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.

9. Conclusion.

The exhibited algorithms are accessible after only a first course in numerical analysis. They enable practical application and open the door to further study of multivariate interpolation, which is much more involved than univariate. The multivariate extension of the classic Newton polynomials and divided-difference algorithm is a very efficient tool when nodes may be chosen in a corner, box, or lower set. For arbitrary nodes, the matrix algorithms give satisfying generalizations. For multivariate basis polynomials of degree k, Newton–Sauer polynomials can have all nonzero coefficients, while Newton–Olver polynomials are simpler with only one term of degree k, specifically, $1\mathbf{x}^{\lambda}$. When applied to a corner of nodes in increasing order of node index, both algorithms will recover the classic Newton polynomials, though the Newton–Sauer normalizes to value one at the associated node. When the nodes do not allow a full basis for polynomials of degree $\leq d$, the Newton–Sauer process explains why (giving an algebraic hypersurface) and extends to a minimal degree interpolation space.

The divided-difference algorithm and the two matrix algorithms represent very different levels of work. Given an interpolation problem, our first phase is finding the appropriate Newton basis and the second phase is solving for the coefficients of an interpolating polynomial. For a corner of nodes in n dimensions up through order d, the first phase takes no work, and we just write down the classic multivariate Newton basis. The divided-difference algorithm is an extremely efficient second phase using an array of $m = \binom{n+d}{n}$ 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 one triangular system requiring roughly m^2 operations, versus two such LU solves for coefficients of the standard monomial basis. In fact, the second phase triangular system has blocks on the diagonal: Newton–Sauer has particularly simple identity blocks; Newton–Olver has only nonsingular blocks requiring more smaller scale solves. Of course, only the second phase need be repeated for different values on the same nodes.

Though not the focus of this paper, the numerical error in results and applications of these algorithms deserves further exposition and study. The algorithms of this paper give exact interpolating polynomial coefficients using rational arithmetic. However, floating point arithmetic can lead to significant roundoff error, especially in interpolation problems which can be ill-conditioned. The effect of node placement can be studied through the condition number of the corresponding matrix, specifically G in the proof of Theorem 2 and V in section 6. In an application using a corner of nodes in [13], condition numbers help explain the loss of some significance in computing multivariate interpolating polynomial coefficients of degree 9 and total loss of accuracy in some coefficients of degree 25. Numerical experiments on arbitrary nodes in [18] used an algorithm equivalent to section 6 on hundreds of random nodes in two variables in $[0,1]^2$, and found mean error around 10^{-7} and largest error around 10^{-3} in (reproducing) node values of a smooth function with maximum 1. This Newton–Sauer basis was significantly more accurate than using a Lagrange basis. The experiments also show that smoother functions produce much better numerical results. The column exchanges used to produce the Newton–Sauer basis can also be used to reduce roundoff error. Since the 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 [9, section 4.3]. This can help, especially in computationally tough problems where all nodes are close to an algebraic hypersurface. A different application issue is truncation error when using a multivariate interpolating polynomial, even with exact coefficients, to approximate f at a point other than an interpolated node. A remainder formula for any lower set of nodes is shown in [11] and simpler remainder forms, for more structured nodes, are shown in [21] along with a remainder formula for the most general case as in section 6.

The algorithms of this paper could be a basis for further exploration of applications, limitations, and improvements of multivariate interpolation, using classic Newton polynomial forms on structured nodes or Newton-like forms on any nodes.

REFERENCES

[1] J. Carnicer and T. Sauer, *Leibniz rules for multivariate divided differences*, J. Approx. Theory, 181 (2014), pp. 43–53, https://doi.org/10.1016/j.jat.2014.02.002. (Cited on p. 372)

[2] J. Czekansky and T. Sauer, *The multivariate Horner scheme revisited*, BIT, 55 (2015), pp. 1043–1056, https://doi.org/10.1007/s10543-014-0533-x. (Cited on p. 365)

[3] C. de Boor and A. Ron, *Computational aspects of polynomial interpolation in several variables*, Math. Comp., 58 (1992), pp. 705–727, https://doi.org/10.2307/2153210. (Cited on p. 370)

[4] N. Dyn and M. S. Floater, *Multivariate polynomial interpolation on lower sets*, J. Approx. Theory, 177 (2014), pp. 34–42, https://doi.org/10.1016/j.jat.2013.09.008. (Cited on pp. 362, 365)

[5] M. Gasca and T. Sauer, *On the history of multivariate polynomial interpolation*, J. Comput. Appl. Math., 122 (2000), pp. 23–35, https://doi.org/10.1016/S0377-0427(00)00353-8. (Cited on p. 362)

[6] M. Gasca and T. Sauer, *Polynomial interpolation in several variables*, Adv. Comput. Math., 12 (2000), pp. 377–410, https://doi.org/10.1023/A:1018981505752. (Cited on pp. 362, 363)

[7] J. Harris, *Interpolation*, in Current Developments in Algebraic Geometry, Math. Sci. Res. Inst. Publ. 59, Cambridge University Press, Cambridge, UK, 2012, pp. 165–176. (Cited on p. 363)

[8] E. Isaacson and H. B. Keller, *Analysis of Numerical Methods*, Dover, New York, 1994; corrected reprint of the 1966 original. (Cited on pp. 362, 370)

[9] D. R. Kincaid and E. W. Cheney, *Numerical Analysis: Mathematics of Scientific Computing*, 3rd ed., AMS, Providence, RI, 2009. (Cited on pp. 363, 379)

[10] K. S. Kunz, *Numerical Analysis*, McGraw-Hill, New York, 1957. (Cited on p. 362)

[11] Z. Li, S. Zhang, T. Dong, and Y. Gong, *Error formulas for Lagrange projectors determined by Cartesian sets*, J. Syst. Sci. Complex., 31 (2018), pp. 1090–1102, https://doi.org/10.1007/s11424-017-6159-8. (Cited on p. 379)

[12] R. D. Neidinger, *Directions for computing truncated multivariate Taylor series*, Math. Comp., 74 (2005), pp. 321–340, https://doi.org/10.1090/S0025-5718-04-01657-6. (Cited on pp. 362, 367)

[13] R. D. Neidinger and B. Altman, *Comparing high-order multivariate AD methods*, Optim. Methods Softw., 33 (2018), pp. 995–1009, https://doi.org/10.1080/10556788.2018.1472256. (Cited on pp. 362, 368, 379)

[14] P. J. Olver, *On multivariate interpolation*, Stud. Appl. Math., 116 (2006), pp. 201–240, https://doi.org/10.1111/j.1467-9590.2006.00335.x. (Cited on pp. 363, 377, 378)

[15] J. M. Peña and T. Sauer, *On the multivariate Horner scheme*, SIAM J. Numer. Anal., 37 (2000), pp. 1186–1197, https://doi.org/10.1137/S0036142997324150. (Cited on p. 365)

[16] G. M. Phillips, *Interpolation and Approximation by Polynomials*, CMS Books Math./Ouvrages Math. SMC 14, Springer-Verlag, New York, 2003, https://doi.org/10.1007/b97417. (Cited on p. 362)

[17] C. Rabut, *Multivariate divided differences with simple knots*, SIAM J. Numer. Anal., 38 (2000), pp. 1294–1311, https://doi.org/10.1137/S0036142999351042. (Cited on p. 372)

[18] T. Sauer, *Computational aspects of multivariate polynomial interpolation*, Adv. Comput. Math., 3 (1995), pp. 219–237, https://doi.org/10.1007/BF02432000. (Cited on pp. 363, 372, 374, 379)

[19] T. Sauer, *Polynomial interpolation of minimal degree*, Numer. Math., 78 (1997), pp. 59–85, https://doi.org/10.1007/s002110050304. (Cited on pp. 363, 376)

[20] T. Sauer, *Lagrange interpolation on subgrids of tensor product grids*, Math. Comp., 73 (2004), pp. 181–190, https://doi.org/10.1090/S0025-5718-03-01557-6. (Cited on p. 365)

[21] T. Sauer and Y. Xu, *A case study in multivariate Lagrange interpolation*, in Approximation Theory, Wavelets and Applications (Maratea, 1994), NATO Adv. Sci. Inst. Ser. C Math. Phys. Sci. 454, Kluwer Academic, Dordrecht, the Netherlands, 1995, pp. 443–452. (Cited on p. 380)

[22] T. Sauer and Y. Xu, *On multivariate Lagrange interpolation*, Math. Comp., 64 (1995), pp. 1147–1170, https://doi.org/10.2307/2153487. (Cited on pp. 363, 374)

[23] J. F. Steffensen, *Interpolation*, 2nd ed., Chelsea, New York, 1950. (Cited on p. 362)

[24] D. N. Varsamis and N. P. Karampetakis, *On the Newton bivariate polynomial interpolation with applications*, Multidimens. Syst. Signal Process., 25 (2014), pp. 179–209, https://doi.org/10.1007/s11045-012-0198-z. (Cited on p. 366)

[25] H. Werner, *Remarks on Newton type multivariate interpolation for subsets of grids*, Computing, 25 (1980), pp. 181–191, https://doi.org/10.1007/BF02259644. (Cited on pp. 365, 368)