

Multivariable Interpolating Polynomials in Newton Forms

Richard D. Neidinger
Davidson College
Davidson, NC 28035
rineidinger@davidson.edu

May 24, 2010

1 Introduction

Unlike one-variable interpolation, multivariable polynomials through given points may or may not exist even when the number of unknown coefficients agree with the number of points. It all depends on the *nodes*, meaning the location of the points in the domain. In one variable, distinct nodes always guarantee a unique interpolating polynomial. In just two variables, the search for an interpolating polynomial of the form

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

shows the richness of the multivariable question. With six unknowns, we would hope to interpolate on six nodes and consider three 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 specified 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 either have no solutions or infinitely many. In (b), using a different polynomial basis (Newton form), having 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 either find a generalization of the nice Newton basis in (b) that facilitates unique interpolation or find a polynomial with value zero on all nodes, as in (a).

Our approach is constructive and seeks to extend as far as possible the successes of one variable interpolation. We will review one-variable theory including the efficient divided-difference algorithm for computing coefficients in the convenient Newton polynomial basis. This is a common topic introductory numerical analysis where Taylor polynomials and interpolating polynomials are

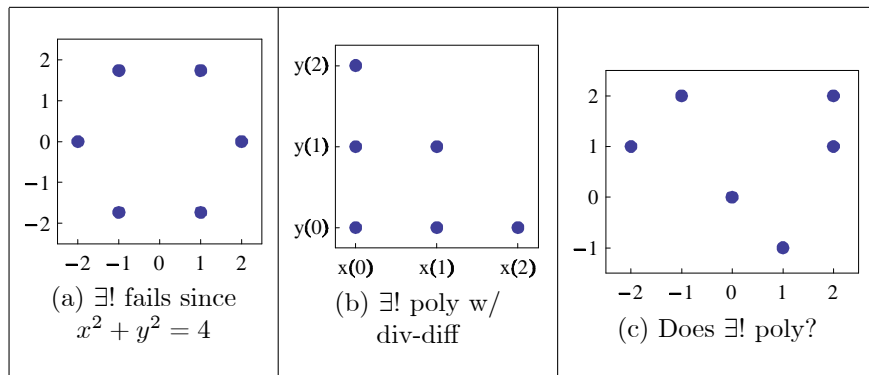


Figure 1: Impossible Circle, Easy Corner, and Generic Example

the two primary (and often interchangeable) tools to derive methods and error bounds. Our focus will be on algorithms, leaving error bounds to another project. In Sections 3 and 4, we study how the classic Newton polynomial form and divided-difference algorithm extend directly to some multivariable contexts. The contexts are described on a multivariable grid and include a corner of nodes, generalizing the example in Figure 1(b), and also include a box of nodes having interpolation within a tensor product space. 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. To design an efficient numerical method using multivariate interpolation, one could construct the appropriate structure of nodes, as is done in [6] which motivated this study.

In Sections 5 through 7, we drop restrictions on geometric structure and consider arbitrary nodes in \mathbb{R}^n . We use variations on Gaussian elimination to explain more recent algorithms ([8] and [7]) that form different generalizations of the Newton basis. The primary focus is to facilitate finding a unique interpolating polynomial in P_d^n , the space of n -variable polynomials of degree $\leq d$. We start with $\binom{n+d}{n}$ nodes, since such a polynomial has that number of unknown coefficients, as justified by a stars-and-bars combinatorial argument. (Each coefficient corresponds to some monomial of powers, totalling $\leq d$, in each of the n variables. Imagine $n+d$ slots where you choose any distinct locations for n dividers or bars and fill the rest with stars. This divides the d stars into $n+1$ groups of zero to d stars. The first n groups correspond to the powers of the variables in exactly one monomial.) Interpolating function values on nodes using the space P_d^n is often called the Lagrange interpolation problem [11], 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 [5].

An interesting result comes when the Gaussian algorithm of Section 5 fails to produce a full basis for P_d^n from the given nodes. Then, the algorithm will produce a polynomial in P_d^n that is a zero on all the nodes, showing that unique

interpolation within P_d^n is impossible, as in Figure 1(a). Geometrically, the nodes lie on the level surface where this polynomial is zero, called an algebraic hypersurface. Moreover, we can augment the algorithm as in Section 6 to produce a minimal degree polynomial subspace within which unique interpolation on these nodes is possible. The algorithm may be used on any nodes whatsoever to produce a polynomial subspace of minimal degree where unique interpolation is possible, although there can be many such minimal degree subspaces.

We encounter some interesting geometric structures in nodes and some curious subspaces of polynomials, but only begin to address the many interpolation questions that are asked along these lines. There are many different types of results about multivariable interpolation: describing other geometric structures in the nodes that guarantee unique interpolation; pairing polynomial subspaces with sets of nodes, and gauging how far nodes are or can be from certain subspaces; interpolating derivatives as well; other algorithms for other forms, etc. ([5, Section 6.10], [2]). We will maintain our constructive theme of finding a good basis for the polynomial space relative to given nodes for real function value interpolation. 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 .

2 One-variable interpolation

The one-variable cubic case will suffice to summarize the classic method and establish notation which nicely generalizes to multiple variables. Take any four distinct reals $x(0), x(1), x(2), x(3)$ with any desired function values f_0, f_1, f_2, f_3 , respectively. This notation will free us to later use x_i and y as multi-dimensional domain variables. The distinct domain values $x(i)$ are called *nodes* and need not be ordered or regularly spaced in any way. We seek a polynomial $p(x) = c_0 + c_1x + c_2x^2 + c_3x^3$ such that $p(x(i)) = f_i$.

To see that a unique solution must exist, we rewrite p in *Newton form*:

$$p(x) = a_0 + a_1(x - x(0)) + a_2(x - x(0))(x - x(1)) + a_3(x - x(0))(x - x(1))(x - x(2)).$$

This changes from the standard basis $\{1, x, x^2, x^3\}$ of P_3^1 to a basis of *classic Newton polynomials* corresponding to given nodes:

$$\begin{aligned} q_0(x) &= 1, \\ q_1(x) &= (x - x(0)), \\ q_2(x) &= (x - x(0))(x - x(1)), \\ q_3(x) &= (x - x(0))(x - x(1))(x - x(2)). \end{aligned}$$

In Newton form, the equations $p(x(i)) = f_i$ form a triangular linear system $\mathbf{a}N = \mathbf{f}$, where $\mathbf{a} = [a_0, a_1, a_2, a_3]$ and $\mathbf{f} = [f_0, f_1, f_2, f_3]$ are row vectors, and

matrix N is

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & (x(1) - x(0)) & (x(2) - x(0)) & (x(3) - x(0)) \\ 0 & 0 & (x(2) - x(0))(x(2) - x(1)) & (x(3) - x(0))(x(3) - x(1)) \\ 0 & 0 & 0 & (x(3) - x(0))(x(3) - x(1))(x(3) - x(2)) \end{bmatrix}.$$

Since the nodes are distinct, the diagonal entries are nonzero and there is a unique solution. Clearly, this holds for any $d + 1$ distinct nodes. While the transpose of this system might be more familiar, this perspective will be useful in later sections that form matrices with columns corresponding to nodes and rows corresponding to basis polynomials.

Although simple forward-substitution will solve the system, the following *divided-difference algorithm* is an even more efficient method. The obvious beginning solves $a_0 = f_0$ and then $a_1 = \frac{f_1 - f_0}{x(1) - x(0)}$. It is not so obvious that nesting similar divided-differences will give the desired coefficients, as proved in the most general setting in Section 4. In fact, the approach can start with the vector \mathbf{f} and successively overwrite entries until it produces the vector \mathbf{a} . We show these steps using the specific example nodes $x(0) = 0$, $x(1) = 1$, $x(2) = -1$, $x(3) = 0.5$ and corresponding function values $f_0 = 5$, $f_1 = 8$, $f_2 = 2$, $f_3 = 4.25$. We introduce a notation for divided-differences that begins with $f[i, i] = f_i$. Begin with the vector

$$\mathbf{f} = [f[0, 0], f[1, 1], f[2, 2], f[3, 3]] = [5, 8, 2, 4.25].$$

We replace entries with computation in this order:

$$\begin{aligned} f[2, 3] &= \frac{f[3, 3] - f[2, 2]}{x(3) - x(2)} = 1.5 \text{ overwrites the } f[3, 3] \text{ entry,} \\ f[1, 2] &= \frac{f[2, 2] - f[1, 1]}{x(2) - x(1)} = 3.0 \text{ overwrites the } f[2, 2] \text{ entry,} \\ f[0, 1] &= \frac{f[1, 1] - f[0, 0]}{x(1) - x(0)} = 3.0 \text{ overwrites the } f[1, 1] \text{ entry.} \end{aligned}$$

Now, the first two entries of the vector are a_0 and a_1 :

$$[f[0, 0], f[0, 1], f[1, 2], f[2, 3]] = [5, 3, 3, 1.5].$$

Next,

$$\begin{aligned} f[1, 3] &= \frac{f[2, 3] - f[1, 2]}{x(3) - x(1)} = 3 \text{ overwrites the } f[2, 3] \text{ entry,} \\ f[0, 2] &= \frac{f[1, 2] - f[0, 1]}{x(2) - x(0)} = 0 \text{ overwrites the } f[1, 2] \text{ entry,} \end{aligned}$$

leaving

$$[f[0, 0], f[0, 1], f[0, 2], f[1, 3]] = [5, 3, 0, 3].$$

Finally,

$$f[0, 3] = \frac{f[1, 3] - f[0, 2]}{x(3) - x(0)} = 6 \text{ overwrites the } f[1, 3] \text{ entry.}$$

leaving

$$\mathbf{a} = [f[0, 0], f[0, 1], f[0, 2], f[0, 3]] = [5, 3, 0, 6].$$

The result is the unique interpolating polynomial for these nodes and values:

$$\begin{aligned} p(x) &= 5 + 3(x - 0) + 0(x - 0)(x - 1) + 6(x - 0)(x - 1)(x + 1) \\ &= 5 + 3x + 6x(x^2 - 1) = 5 - 3x + 6x^3. \end{aligned}$$

We have taken the traditional divided-difference notation $f[x(0), x(1), x(2), x(3)]$ and abbreviated it to $f[0, 3]$, which facilitates multivariable generalization.

3 Nodes on a multivariate grid

For any finite set of nodes in \mathbb{R}^n , take each coordinate x_i and label all reals used in that coordinate $x_i(0), x_i(1), \dots, x_i(k_i)$; we call these *tick marks*. 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 Figures 1(b) and 2. 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 .

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. We can index a grid point as in $\mathbf{x}(0213) = (x_1(0), x_2(2), x_3(1), x_4(3))$, where the condensed notation is 0213 is used for the multi-index $\boldsymbol{\lambda} = (0, 2, 1, 3)$.

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. So $\boldsymbol{\lambda} = (0, 2, 1, 3)$ has order 6 and we say $\mathbf{x}(0213)$ is a node of order 6. Throughout, we use a linear ordering of the multi-indices (for a fixed dimension n) by first increasing by order $|\boldsymbol{\lambda}|$ and then, within the same order, listing from highest-to-lowest in the first coordinate that differs going left-to-right. We will use this linear ordering in any listing corresponding to multi-indices. However, the notation $\boldsymbol{\lambda} \leq \boldsymbol{\beta}$ will mean coordinate-wise, i.e., $\lambda_i \leq \beta_i$ for all $i = 1, \dots, n$.

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

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

Here $\mathbf{x} = (x_1, \dots, x_m, \dots, x_n)$ are free variables and each $x_m(i)$ goes through all tick marks with $i < \lambda_m$.

For example, the structured nodes in Figure 1(b) give rise to the classic Newton polynomial basis for quadratic polynomials:

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

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 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	$\Delta\Delta$

(2)

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 (2) define matrix N , corresponding to the nodes. Then $\mathbf{a}N = \mathbf{f}$ is the system with coefficients \mathbf{a} of the Newton basis for the desired function values \mathbf{f} at the nodes. Clearly, a unique solution exists and could be easily found. The special structure of this example is used in the divided-difference algorithm (next section) that directly manipulates the vector of values \mathbf{f} and ends with \mathbf{a} , using memory only the square root of the size of N .

To get the block structure in (2), 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. Each $q_{\boldsymbol{\lambda}}$ is a polynomial that

- (a) is zero at nodes of lower order,
- (b) is nonzero at node $\mathbf{x}(\boldsymbol{\lambda})$ and (3)
zero at other nodes of the same order, and
- (c) has degree $|\boldsymbol{\lambda}|$ and only one term of highest degree.

One consequence is 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.

Any nodes whatsoever lead to an invertible triangular system similar to (2) except that the nodes, and hence polynomials, may use higher orders without using all multi-indices of lower orders. For example, node set $\{\mathbf{x}(00), \mathbf{x}(10), \mathbf{x}(11)\}$

gives the 3×3 invertible triangular matrix using rows corresponding to q_{00}, q_{10}, q_{11} . In Section 5, 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 properties (3) as much as possible. In this section, we associate all nodes with grid points and use classic Newton polynomials.

Any set of nodes can be described by $\{\mathbf{x}(\boldsymbol{\lambda}) : \boldsymbol{\lambda} \in I\}$ for some tick marks and some $I \subseteq \mathbb{N}_0^n$. Define $q_{\boldsymbol{\lambda}}$ for each $\boldsymbol{\lambda} \in I$. For each grid point $\mathbf{x}(\boldsymbol{\mu})$ (whether or not it is a node), the definition of $q_{\boldsymbol{\lambda}}$ yields:

$$q_{\boldsymbol{\lambda}}(\mathbf{x}(\boldsymbol{\mu})) = 0 \text{ if and only if } \exists m \in \{1, \dots, n\} \text{ such that } \mu_m < \lambda_m. \quad (4)$$

Interpolation at the set of nodes, using a linear combination of these polynomials, results in the linear system:

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

The matrix of this system is $N = [q_{\boldsymbol{\lambda}}(\mathbf{x}(\boldsymbol{\mu}))]$, where we list whatever indices are in I according to the linear ordering. Diagonal entries $q_{\boldsymbol{\lambda}}(\mathbf{x}(\boldsymbol{\lambda})) \neq 0$ and, if $\boldsymbol{\mu}$ precedes $\boldsymbol{\lambda}$ in the linear ordering, then $q_{\boldsymbol{\lambda}}(\mathbf{x}(\boldsymbol{\mu})) = 0$. This means that N is an invertible triangular matrix, proving the following:

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

The catch is that $\text{Span}\{q_{\boldsymbol{\lambda}} : \boldsymbol{\lambda} \in I\}$ may or may not be useful. We first consider a curious application to the circle of nodes where unique interpolation by a quadratic polynomial is impossible. Then, we define three cases of node structures where interpolation using the corresponding span of q 's is ideal and the divided-difference algorithm will generalize.

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 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 (1) 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 5 to find a subspace of polynomials of minimal degree 3 that is correct for the circle of nodes. We now consider important special cases where the classic Newton polynomial basis is the right thing to do.

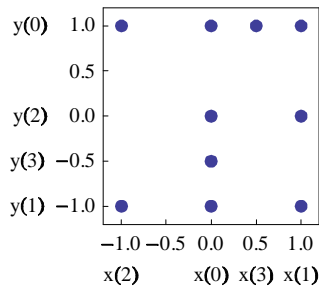


Figure 2: Example corner of nodes

Define a *corner* of nodes to be a set of nodes in \mathbb{R}^n that equal $\{\mathbf{x}(\boldsymbol{\lambda}) : |\boldsymbol{\lambda}| \leq d\}$ for some specification of tick marks. The multi-indices $I = \{\boldsymbol{\lambda} : |\boldsymbol{\lambda}| \leq d\}$ are literally in the corner of the nonnegative integer lattice in n dimensions. In contrast, the nodes can have any irregular tick marks. The example in Figure 1(b) is an obvious corner of nodes. A more generic example is given by the nodes in Figure 2. The special property of a corner of nodes is that the

$$\text{Span}\{q_{\boldsymbol{\lambda}} : |\boldsymbol{\lambda}| \leq d\} = P_d^n,$$

the space of n -variable polynomials of degree $\leq d$. For any function values on any corner of nodes in \mathbb{R}^n , there exists a unique interpolating polynomial of degree $\leq d$. This was exploited in the application to automatic differentiation [6] that motivated this study. Very roughly, the goal there was to find a multivariable Taylor polynomial of degree $\leq d$, given that values of the polynomial at points were easier to compute indirectly (using univariate directional derivatives). By computing values at a corner of nodes, the unique multivariable Taylor polynomial could be constructed. The numerical stability was improved by choosing the tick marks corresponding to alternating dyadic sequence $0, 1, -1, \frac{1}{2}, \frac{-1}{2}, \frac{1}{4}, \frac{-1}{4}, \frac{3}{4}, \frac{-3}{4}, \frac{1}{8}, \dots$

A *box* of nodes which is a set of nodes that equal $\{\mathbf{x}(\boldsymbol{\lambda}) : \boldsymbol{\alpha} \leq \boldsymbol{\lambda} \leq \boldsymbol{\beta}$ (coordinate-wise) $\}$ for some specification of tick marks and some fixed multi-indices $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. Here, the set of nodes is the entire set of grid points for these tick marks. If these were the only nodes being considered, we could start all tick marks at $\mathbf{0}$ instead of $\boldsymbol{\alpha}$, and directly use the definition of $q_{\boldsymbol{\lambda}}$ above. However, we need a version with the origin shifted to $\boldsymbol{\alpha}$ for handling sub-boxes in the proofs of the next section. The resulting generalization of Theorem 1 says that there is a unique interpolating polynomial in

$$\begin{aligned} & \text{Span} \left\{ \prod_{m=1}^n \prod_{i=\alpha_m}^{\lambda_m-1} (x_m - x_m(i)) : \boldsymbol{\alpha} \leq \boldsymbol{\lambda} \leq \boldsymbol{\beta} \right\} \\ &= \text{Span}\{x_1^{\mu_1} x_2^{\mu_2} \cdots x_n^{\mu_n} : \mathbf{0} \leq \boldsymbol{\mu} \leq \boldsymbol{\beta} - \boldsymbol{\alpha}\}. \end{aligned} \quad (5)$$

This is a tensor-product of polynomial spaces. All results of this section and the next are sometimes referred to simply as the tensor-product case in modern

surveys of interpolation [2]. We will repeatedly use the property that the unique interpolating polynomial for function values on a box 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 (5).

Finally, we define an *echelon* of nodes, the most general structure for which our divided-difference algorithm will work, encompassing both corners and boxes with origin $\mathbf{0}$ as special cases. An *echelon* of multi-indices is any index set $J \subseteq \mathbb{N}_0^n$ with the property that whenever $\beta \in J$ and $\mathbf{0} \leq \lambda \leq \beta$ (coordinate-wise), then $\lambda \in J$. An *echelon* of nodes is any set in \mathbb{R}^n that equals $\{\mathbf{x}(\lambda) : \lambda \in J\}$ for some specification of tick marks such that J is an echelon of multi-indices. The $\text{Span}\{q_\lambda : \lambda \in J\}$ is a sum of subspaces. For example, $J = \{00, 10, 20, 30, 40, 01, 02, 03, 11\}$ extends out each axis with a step in the middle at 11. For any corresponding nodes and Newton polynomials, an element of $\text{Span}\{q_\lambda : \lambda \in J\}$ can be written as $p(x) + q(y) + axy$ for some scalar a and one-variable polynomials p and q of degree ≤ 4 and 3 , respectively. The key property here is that we can decrement any coordinate in any node's index and get to another node in the set. This allows a generalization of divided-differences, where each difference is between a value at a multi-index and a value at a multi-index decremented in one coordinate.

4 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 $(x, y) = (-1, -1)$ where we want function value 16. The value at $(2, 1)$ will be changed in each

\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 $i + j \geq k$

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 in (a) by $\frac{16-10}{-1-1} = -3$, in (b) by $\frac{-3-1}{-1-0} = 4$, and in (c) by $\frac{4-0}{-1-1} = -2$. 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 (2). 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 describes the plane through points $(0, 1, 5)$, $(1, 1, 3)$, $(0, -1, -2)$. The quadric surface given by 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 (which does not seem to have appeared in print, although the idea is very old [10, p. 204]). Fix an echelon of nodes and values $\{(\mathbf{x}(\boldsymbol{\lambda}), f_{\boldsymbol{\lambda}}) : \boldsymbol{\lambda} \in J\}$ over some echelon of multi-indices J , as defined at the end of the previous section. We seek the coefficients of the unique interpolating polynomial in $\text{Span}\{q_{\boldsymbol{\lambda}} : \boldsymbol{\lambda} \in J\}$. First, we focus on sub-boxes of nodes where $\boldsymbol{\alpha} \leq \boldsymbol{\lambda} \leq \boldsymbol{\beta}$ for some $\boldsymbol{\beta} \in J$. For this box of nodes, there is a unique interpolating polynomial in $\text{Span}\{x_1^{\mu_1} x_2^{\mu_2} \cdots x_n^{\mu_n} : \mathbf{0} \leq \boldsymbol{\mu} \leq \boldsymbol{\beta} - \boldsymbol{\alpha}\}$.

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 (5). 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. The multi-index notation facilitates this exposition.

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 $p(\mathbf{x}(\boldsymbol{\lambda})) = p_-(\mathbf{x}(\boldsymbol{\lambda})) + (1)(p_+(\mathbf{x}(\boldsymbol{\lambda})) - p_-(\mathbf{x}(\boldsymbol{\lambda}))) =$

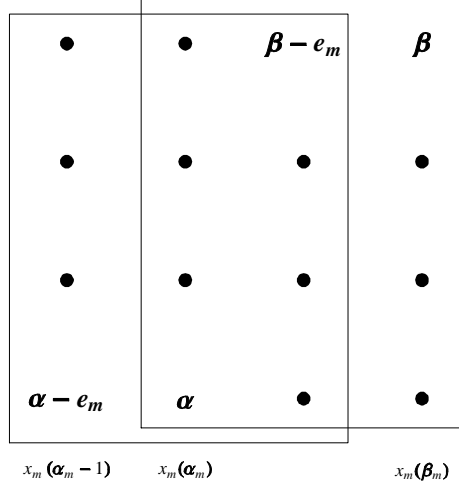


Figure 3: Multi-indices in proof of divided-difference

$p_+(\mathbf{x}(\boldsymbol{\lambda})) = f_{\boldsymbol{\lambda}}$. We conclude that $p(\mathbf{x})$ is the interpolating polynomial over the union and, by definition, the coefficient of the highest power is $f[\boldsymbol{\alpha} - \mathbf{e}_m, \boldsymbol{\beta}]$. This highest power is $x_1^{\mu_1} x_2^{\mu_2} \cdots x_n^{\mu_n}$ for $\boldsymbol{\mu} = \boldsymbol{\beta} - \boldsymbol{\alpha} + \mathbf{e}_m$, which occurs in (7) 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 $f[\boldsymbol{\alpha}, \boldsymbol{\beta}]$ and in $p_-(\mathbf{x})$ is $f[\boldsymbol{\alpha} - \mathbf{e}_m, \boldsymbol{\beta} - \mathbf{e}_m]$. 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 in the right side of (7) is 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 $\boldsymbol{\beta} \in J$, initialize memory location $A[\boldsymbol{\beta}] = f[\boldsymbol{\beta}, \boldsymbol{\beta}] = f_{\boldsymbol{\beta}}$. This location will be changed $|\boldsymbol{\beta}|$ times by Theorem 4 to reach $f[\mathbf{0}, \boldsymbol{\beta}]$. We consistently choose the direction of decrements so that, before each pass through the array, there exist $\boldsymbol{\alpha}$ and \mathbf{e}_m such that $A[\boldsymbol{\beta}] = f[\boldsymbol{\alpha}, \boldsymbol{\beta}]$ and $A[\boldsymbol{\beta} - \mathbf{e}_m] = f[\boldsymbol{\alpha} - \mathbf{e}_m, \boldsymbol{\beta} - \mathbf{e}_m]$. The divided difference will then compute $A[\boldsymbol{\beta}] = f[\boldsymbol{\alpha} - \mathbf{e}_m, \boldsymbol{\beta}]$. One method, corresponding to the "left before below" choice in Table 1, is to simply decrement the coordinates of $\boldsymbol{\beta}$ 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, $\boldsymbol{\beta} = (2, 0, 4, 1)$ will have been decremented 4 times to $\boldsymbol{\alpha} = (0, 0, 2, 1)$ and will compute further to $\boldsymbol{\alpha} - \mathbf{e}_m = (0, 0, 1, 1)$, lowering the $m = 3$ coordinate to $\alpha_m - 1 = 1$. Since $\boldsymbol{\beta} - \mathbf{e}_m = (2, 0, 3, 1)$, the 5th application of Theorem 4 at $\boldsymbol{\beta} = (2, 0, 4, 1)$, 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 linear ordering.

5 Polynomials for arbitrary nodes

We now return to the general question: given any set of $\binom{n+d}{n}$ nodes in \mathbb{R}^n , does there exist a unique interpolating polynomial of degree $\leq d$ for values on these nodes? We will find either a positive answer with a constructive algorithm or a negative answer with concise evidence. For any corner of nodes, we know the answer is yes, with a classic Newton polynomial basis that makes it easy to compute the interpolating polynomial. For more general nodes, as in Figure 1(c), the approach of Section 3 fails to provide a useful polynomial subspace. So, we now abandon this way of indexing our nodes and develop a different generalization of Newton polynomials that recovers some of the properties.

Let's start from the standard basis with $n = 2$ and $d = 2$. 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 multi-index notation for coefficients conveniently matches the exponent powers. We also choose to index the six distinct nodes using multi-index notation,

$$\mathbf{x}_{00} = (a_1, b_1), \mathbf{x}_{10} = (a_2, b_2), \mathbf{x}_{01} = (a_3, b_3), \mathbf{x}_{20} = (a_4, b_4), \mathbf{x}_{11} = (a_5, b_5), \mathbf{x}_{02} = (a_6, b_6),$$

although the coordinates have no particular structure (unlike our $\mathbf{x}(00)$ notation earlier). The multi-index does arbitrarily associate each node $\mathbf{x}_{\boldsymbol{\mu}}$ with some order $|\boldsymbol{\mu}|$ and will help us to associate a generalized Newton polynomial.

The matrix formulation of this problem is $\mathbf{c}V = \mathbf{f}$, for coefficients $\mathbf{c} = [c_{00}, c_{10}, c_{01}, c_{20}, c_{11}, c_{02}]$ and function values $\mathbf{f} = [f_{00}, f_{10}, f_{01}, f_{20}, f_{11}, f_{02}]$, and

where the matrix is a generalization of a one-dimensional Vandermonde matrix:

$$V = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \\ b_1 & b_2 & b_3 & b_4 & b_5 & b_6 \\ a_1^2 & a_2^2 & a_3^2 & a_4^2 & a_5^2 & a_6^2 \\ a_1 b_1 & a_2 b_2 & a_3 b_3 & a_4 b_4 & a_5 b_5 & a_6 b_6 \\ b_1^2 & b_2^2 & b_3^2 & b_4^2 & b_5^2 & b_6^2 \end{bmatrix}. \quad (8)$$

Unlike the one-variable case, V may or may not be invertible, which is equivalent to existence and uniqueness for the interpolation problem. An efficient way to proceed would be Gaussian elimination (usually performed on V^T) to either find a unique solution or show it is impossible. From a numerical methods perspective, Gaussian elimination produces an LU -decomposition of V^T that converts the problem to simple triangular systems. In a similar way, we will perform Gaussian elimination steps directly on V and find an upper-triangular W and lower-triangular matrix R , where R defines polynomials in a generalized Newton basis, and where solving just $\mathbf{a}W = \mathbf{f}$ will give the coefficients of the interpolating polynomial in this basis.

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:

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	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

(9)

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, listed with increasing degree, as labeled in the example. Initially, the rows also correspond to these monomials. We block all rows and columns according to the order of the associated multi-index.

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 (9). 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

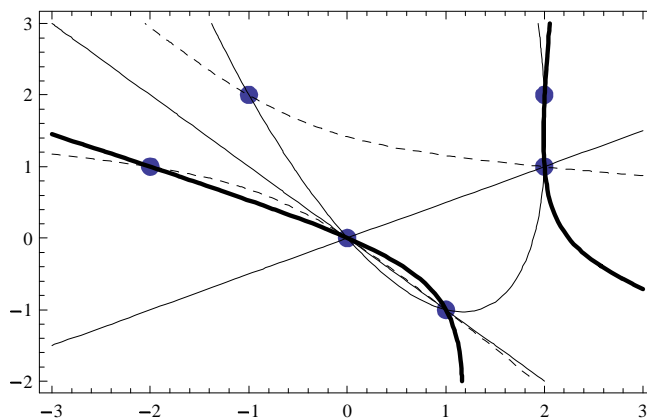


Figure 4: Zero contours of Newton-Sauer polynomials

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. 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 (9), 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	$\frac{3}{13}$	$\frac{5}{26}$	$-\frac{4}{39}$	$-\frac{3}{13}$	$\frac{1}{78}$
0	0	0	0	0	1	0	$\frac{13}{52}$	$\frac{13}{52}$	$\frac{39}{52}$	$\frac{13}{13}$	$\frac{39}{52}$

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 (3), 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 produced in [8]. Our row-operations with column exchanges are equivalent to the steps in [8, 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 . (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 needed 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 a particularly easy triangular system, though W is not quite as simple as the classic (2) that was solved by divided-differences. Here, forward-substitution (in row or block forms) may be used and results in $\mathbf{a} = [5, 1, 2, 1, 5, 7]$. 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:

$$\begin{array}{c|ccc|ccc|ccc}
 & \text{circle of nodes} & & 1 & x & y & x^2 & xy & y^2 \\
 \hline
 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
 0 & 1 & 0 & \frac{-1}{2} & 1 & \frac{3}{2} & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\
 0 & 0 & 1 & \frac{1}{2} & 1 & \frac{1}{2} & \frac{1}{2} & 0 & \frac{\sqrt{3}}{6} & 0 & 0 & 0 \\
 \hline
 0 & 0 & 0 & 1 & 0 & 1 & \frac{-1}{3} & 0 & 0 & \frac{1}{3} & 0 & 0 \\
 0 & 0 & 0 & 0 & 1 & 1 & \frac{1}{6} & \frac{1}{4} & \frac{\sqrt{3}}{12} & \frac{1}{12} & \frac{\sqrt{3}}{12} & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & 0 & 1 & 0 & 1
 \end{array} . \tag{10}$$

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 \mathbf{f}_μ only affects coefficient \mathbf{c}_μ and coefficients of higher order.) If unsuccessful, we find a row of zeros on the left and a nonzero polynomial of degree $\leq d$ that is zero on all the nodes, showing that unique interpolation is impossible. Geometrically, the nodes lie on the level surface where this polynomial is zero, called an algebraic hypersurface.

Theorem 6 *Given a set of $\binom{n+d}{n}$ 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.*

6 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, 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

$$\frac{\text{circle of nodes} \quad \parallel \quad 1 \quad x \quad y \quad x^2 \quad xy \quad x^3}{-1 \mid 1 \quad -1 \mid -8 \quad 1 \mid 8 \parallel 0 \mid 0 \quad 0 \mid 0 \quad 0 \quad 0 \mid 1}.$$

The desired identity blocks in W are shrunken on each deletion, so they corresponding to the degree of the monomials involved. Thus, our previous five rows of (10) remain unchanged and we do row additions into the last row to result in

$$\frac{\text{circle of nodes} \quad \parallel \quad 1 \quad x \quad y \quad x^2 \quad xy \quad x^3}{0 \mid 0 \quad 0 \mid 0 \quad 0 \mid 1 \parallel \frac{-1}{6} \mid \frac{-1}{12} \quad 0 \mid \frac{1}{6} \quad 0 \quad 0 \mid \frac{1}{12}}.$$

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

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

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 [9], 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.

7 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-divided difference algorithm. Even for a corner of nodes corresponding to the classic Newton polynomials, the algorithm of Section 5 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 non-zero so that we may divide by it. For multivariable diagonal blocks, this suggests either an identity matrix as in Section 5 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 (3) 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 (9) 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	$\frac{10}{3}$	$\frac{16}{3}$	$\frac{1}{3}$	0	$\frac{2}{3}$	$\frac{1}{3}$	0	0	1

(11)

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 (3), 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 $d = 3$:

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

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 5 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. 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 $\mathbf{a}_k = \mathbf{a}_k - \mathbf{a}_i U_{i,k}$ 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 (11) and initial $\mathbf{f} = [[5], [6, 7], [8, 9, 10]]$, we get

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

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 exactly the classic Newton polynomials and all U_{i_k} are the "tick-mark" differences as in (2). 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.

8 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 $m = \binom{n+d}{n}$ 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.

Summary 7 *There exists a unique polynomial of degree $\leq d$ for any values on $d + 1$ distinct nodes in \mathbb{R} , as seen using a Newton polynomial basis and as efficiently computed using a divided-difference algorithm. Even with the correct number of nodes in \mathbb{R}^n , interpolation may or may not be possible within P_d^n , the space of n -variable polynomials of degree $\leq d$. A straightforward generalization of the Newton basis and divided-difference algorithm is shown to apply to a geometric structure of nodes, including a corner of nodes having unique interpolation within P_d^n , and including a box of nodes for interpolation within a tensor product space. Without assuming any geometric structure, an algorithm based on Gaussian elimination either produces a generalization of a Newton basis that facilitates unique interpolation or produces a polynomial that is zero on all nodes, showing that unique interpolation is impossible. The algorithm may also be used on any nodes whatsoever to produce a polynomial subspace of minimal degree where unique interpolation is possible. A variation of this algorithm using matrix block operations produces another generalization of the Newton basis, which exactly agrees with the classic one for a corner of nodes, and computes an interpolating polynomial using formulas similar to divided-differences.*