UNIVERSITY OF WISCONSIN-MADISON
COMPUTER SCIENCES DEPARTMENT

Computational aspects of multivariate polynomial interpolation: Indexing the coefficients

Carl de Boor[1]

January 1999

ABSTRACT

An algorithm is derived for generating the information needed to pass efficiently between multiindices of neighboring degrees, of use in the construction and evaluation of interpolating polynomials and in the construction of good bases for polynomial ideals.

short title: Indexing the coefficients of multivariate polynomials

AMS (MOS) Subject Classifications: primary 41A05, 41A10, 41A63, 65D05, 65D15

Key Words: polynomials, multivariate, evaluation, differentiation

Authors' affiliation and address:
Computer Sciences Department
University of Wisconsin-Madison
1210 West Dayton St.
Madison WI 53706

[1] supported by the United States Army under Contracts No. DAAH04-95-1-0089 and DAAG55-98-1-0443, and by the National Science Foundation under Grant No. DMS-9626319.

Computational aspects of multivariate polynomial interpolation: Indexing the coefficients

Carl de Boor

1. Introduction

In dealing with d-variate polynomials, in power form or in the related Bernstein-Bézier form, one has to store the coefficients in some linear order, hence needs a total ordering of the set $\mathbb{Z}_+^d$ of multi-indices with d entries, i.e., an invertible map

$$N : \mathbb{Z}_+^d \to \mathbb{N}.$$

With the standard abbreviation

$$|\alpha| := \|\alpha\|_1 := \sum_i |\alpha(i)|, \qquad \alpha \in \mathbb{Z}_+^d,$$

we follow custom and impose on such N the condition

$$|\alpha| < |\beta| \quad \Longrightarrow \quad N(\alpha) < N(\beta). \tag{1}$$

Beyond that, there seem to be no further conditions, other than the more detailed condition

$$N(\alpha) < N(\beta) \quad \Longrightarrow \quad N(\alpha+\gamma) < N(\beta+\gamma), \qquad \text{all } \ \gamma \in \mathbb{Z}_+^d$$

(making it a 'monomial ordering'; see, e.g., [CLO: p53ff]), and the nebulous one that N should make calculations 'easy'.

One such calculation is the evaluation of the (normalized) power form,

$$p(x) = \sum_{\alpha\in\mathbb{Z}_+^d} c(\alpha) x^\alpha \tbinom{|\alpha|}{\alpha}, \qquad \text{with } \ \tbinom{|\alpha|}{\alpha} := |\alpha|!/\prod_i \alpha(i)!,$$

using (symmetric) Nested Multiplication aka Horner's method. With

$$\iota_i := (\underbrace{0,\ldots,0}_{i-1 \text{ terms}}, 1, 0, \ldots)$$

the ith coordinate vector, and assuming the polynomial to have (total) degree $\le k$, the calculation

$$v(\alpha) \;\leftarrow\; c(\alpha), \quad |\alpha| = k,$$

$\texttt{for}\ \ j = k{:}{-1}{:}1$

$$v(\alpha) \;\leftarrow\; c(\alpha) + \sum_{i=1}^{d} v(\alpha+\iota_i)x(i), \quad |\alpha| = j-1,$$

produces, in $v(0)$, the number $p(x)$, as one easily verifies, using the observation that, in the resulting explicit expression for $v(0)$, the coefficient $c(\alpha)$ appears indeed only with the factor x^a and exactly as many times as there are sequences $(0 = \beta_0, \beta_1, \ldots, \beta_{|\alpha|} = \alpha)$ with $\beta_{j+1} - \beta_j \in \{\iota_1, \ldots, \iota_d\}$, i.e., exactly $\binom{|\alpha|}{\alpha}$ times.

Assuming $c(\alpha)$ to be the $N(\alpha)$th entry in the vector used to store the coefficients and $v(\alpha)$ stored analogously, one needs in this calculation, for given $N(\alpha)$, easy access to the vector $(N(\alpha + \iota_i) : i = 1{:}d)$. Such easy access is also needed when multiplying a homogeneous polynomial by the d monomials, $()^{\iota_i}$, all i, of use in the construction of the least interpolant [BR] and of good bases for polynomial ideals. It is the purpose of this note to describe such easy access for the graded lexicographic (**grlex**) ordering and thereby fill the gap left in this regard in [BR].

2. Notation

Here and below, $x(i)$ denotes the ith component of $x \in \mathbb{R}^d$, all i. Also, `MATLAB`'s colon notation is used, according to which, for any integers a and b,

$$a{:}b \;:=\; (a, a+1, \ldots, b),$$

with the sequence empty if $a > b$. If, more generally, a, s, b are real numbers, then

$$a{:}s{:}b = (a, a+s, a+2s, \ldots, a+ms),$$

with m the natural number for which $a + ms$ lies between a and b, while $a + (m+1)s$ does not. Also, if needed for clarity, a sequence is, at times, delimited by brackets rather than parentheses (as is done in `MATLAB`). For example,

$$[i{:}d, 1{:}(i-1)] := (i, i+1, \ldots, d, 1, 2, \ldots, i-1).$$

Such use of brackets is particularly helpful when using `MATLAB`'s convenient notation $A(b, c)$ for the matrix whose (i, j) entry is the number $A(b(i), c(j))$, all i, j. To be sure, $A(b, c)$ is of order $\#b \times \#c$, with $\#a$ the number of entries in a sequence a. Also, if b and/or c here are replaced by `:`, then all the rows and/or all the columns of A are to be taken (in order). This `MATLAB` notation is particularly handy when dealing with permutations. Let `a` be an n-vector and let `p` be (an n-vector containing) a permutation of order n. Then `a(p)` is the n-vector that results from permuting the entries of `a` according to the permutation (contained in) `p`. For that reason, the `MATLAB` statement

```
a(p) = b;
```

puts into `a` the entries of `b` permuted according to the *inverse* of the permutation (contained in) `p`.

Finally, we use $\#A$ (rather than $|A|$) to denote the cardinality of the set A.

3. A formula for $N(\alpha)$

Let

$$A_j := A_j(d) := \{\alpha \in \mathbb{Z}_+^d : |\alpha| = j\},$$

and

$$A_{\le k} := A_{\le k}(d) := \bigcup_{j \le k} A_j(d).$$

Then condition (1) is equivalent to having

$$N(\alpha) \le \# A_{\le |\alpha|}(d), \qquad \alpha \in \mathbb{Z}_+^d.$$

Equivalently, N is of the form

$$N(\alpha) =: \# A_{<|\alpha|}(d) + n(\alpha),$$

with

$$n(A_j) = \{1, 2, \ldots, \# A_j\}, \qquad j = 0, 1, 2, \ldots.$$

The 'standard' choice for n on A_j is the reverse lexicographic ordering. For certain reasons, I prefer here to use the lexicographic ordering instead. For this choice, consider now the following:

Problem. *Give a formula for $n(\alpha)$.*

For its solution, let

$$\mathtt{A}_j = \mathtt{A}_{j,d}$$

be the matrix whose rows contain the elements of $A_j(d)$ in lexicographic order. E.g., for $d = 3$, the first four $\mathtt{A}_j$ are:

$$\mathtt{A}_0 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}, \quad \mathtt{A}_1 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad \mathtt{A}_2 = \begin{bmatrix} 0 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 2 & 0 & 0 \end{bmatrix}, \quad \mathtt{A}_3 = \begin{bmatrix} 0 & 0 & 3 \\ 0 & 1 & 2 \\ 0 & 2 & 1 \\ 0 & 3 & 0 \\ 1 & 0 & 2 \\ 1 & 1 & 1 \\ 1 & 2 & 0 \\ 2 & 0 & 1 \\ 2 & 1 & 0 \\ 3 & 0 & 0 \end{bmatrix}.$$

Now note that the rows of $\mathtt{A}_j$ that have i as their first entry have the elements of $A_{j-i}(d-1)$ in lexicographic order in their remaining $d-1$ entries. In MATLAB terms,

$$\mathtt{A}_{j,d}(\mathtt{find(}A_{j,d}(:,1)\mathtt{==}i\mathtt{)}, 2{:}d) \ \text{ equals } \ \mathtt{A}_{j-i,d-1}, \qquad i = 0{:}j.$$

This says that the last $d-1$ columns of $\mathtt{A}_{j,d}$ contain, in order, in their rows the elements of $A_j(d-1), A_{j-1}(d-1), \ldots, A_0(d-1)$, or, in MATLAB terms,

$$\mathtt{A}(:,2{:}d) \ \text{ equals } \ [\mathtt{A}_{j,d-1}; \mathtt{A}_{j-1,d-1}; \ldots; \mathtt{A}_{0,d-1}].$$

With $\alpha(2{:}d) := (\alpha(2), \ldots, \alpha(d))$, it follows that

$$n(\alpha) = \sum_{0 \le i < \alpha(1)} \#A_{|\alpha|-i}(d-1) \ + \ n(\alpha(2{:}d)).$$

Therefore, by induction,

$$n(\alpha) = \sum_{r=1}^{d} \sum_{0 \le i < \alpha(r)} \#A_{|\alpha(r:d)|-i}(d-r).$$

Since

$$\#A_j(d) = \tbinom{j+d-1}{j},$$

we could now express this double sum in terms of binomial coefficients. But we refrain from doing so since the resulting formula is of limited use: it provides $n(\alpha)$ as a function of α, while we really would like $n(\alpha + \iota_i)$ as a function of $N(\alpha)$.

4. The map $N(\alpha) \mapsto n(\alpha + \iota_i)$

In the terms introduced, the practical problem of interest mentioned at the outset is the following:

Problem. *Describe the map*

$$N(\alpha) \mapsto (n(\alpha + \iota_i) : i = 1{:}d).$$

We propose to solve this problem here in the sense of providing an algorithm for the calculation of the first few (however many) columns of the d-rowed matrix $\mathtt{M}$ for which

$$\mathtt{M}(i, N(\alpha)) = n(\alpha + \iota_i), \qquad i = 1{:}d, \ \alpha \in \mathbb{Z}_+^d. \tag{2}$$

For this, we now describe a process of generating the matrix $\mathtt{A}_{j+1} = \mathtt{A}_{j+1,d}$ (whose rows contain the elements of $A_{j+1}(d)$ in lexicographic order) from the matrix $\mathtt{A}_j$. This time, we block the matrix $\mathtt{A}_j$ in a different way. We notice that

$$\mathtt{A}_j(1{:}\#A_j(i), (d-i+1){:}d) \ \text{ equals } \ \mathtt{A}_{j,i}, \qquad i = 1{:}d.$$

Further, by adding 1 to all the entries of column $d+1-i$ of the corresponding segment $\mathtt{A}_j(1{:}\#A_j(i), :)$ of $\mathtt{A}_j$ (i.e., the leftmost column in this block that is not entirely zero), we obtain a segment of $\mathtt{A}_{j+1}$. Hence, if we now concatenate these modified matrices in this order, we obtain a matrix containing $\sum_{i=1}^{d} \#A_j(i) = \#A_{j+1}(d)$ distinct elements of $A_{j+1}(d)$ in lexicographic order, hence this must be $\mathtt{A}_{j+1}$.

This has two consequences of interest here.

(i) If $\mathtt{m}_j$ is the last column of $\mathtt{A}_j$, then the last column of $\mathtt{A}_{j+1}$ is the sequence

$$\mathtt{m}_{j+1} = (j+1, \mathtt{m}_j(1{:}\#A_j(2)), \mathtt{m}_j(1{:}\#A_j(3)), \ldots, \mathtt{m}_j). \tag{3}$$

(ii) Since adding 1 to the first column of $\mathtt{A}_j$, as we do in the last step of the above process, corresponds to adding ι_1 to each of the rows of $\mathtt{A}_j$, we know now that, for $|\alpha| = j$ and with

$$\mathtt{M}_j(:, n(\alpha)) := \mathtt{M}(:, N(\alpha)), \qquad |\alpha| = j,$$

necessarily

$$\mathtt{M}_j(1,:) = (\#A_{j+1}(d) - \#A_j(d)) \ + \ (1{:}\#A_j(d)). \tag{4}$$

From this, we obtain $\mathtt{M}_j(i,:)$ for $i > 1$, by considering the effect of rotating the columns of $\mathtt{A}_j$, i.e., by considering the matrix $\mathtt{A}_j(:, [2{:}d, 1])$ obtained from $\mathtt{A}_j$ by moving the first column all the way to the right, to become the last column. The rows of this new matrix still comprise all the elements of $A_j(d)$, but in some new order. Hence there exists exactly one permutation, call it q_j, that restores lexicographic order, i.e., for which

$$\mathtt{A}_j(q_j, [2{:}d, 1]) = \mathtt{A}_j.$$

Therefore, by induction,

$$\mathtt{A}_j(q_j^{i-1}, [i{:}d, 1{:}(i-1)]) = \mathtt{A}_j, \qquad i = 1{:}d, \tag{5}$$

with q_j^k the kth power of the permutation q_j, all k. Hence, also

$$\mathtt{A}_{j+1}(q_{j+1}^{i-1}, [i{:}d, 1{:}(i-1)]) = \mathtt{A}_{j+1}, \qquad i = 1{:}d.$$

Therefore, by (4), on adding ι_1 to row r of the matrix $\mathtt{A}_j(q_j^{i-1}, [i{:}d, 1{:}(i-1)]) = \mathtt{A}_j$, we obtain row $\mathtt{M}_j(1,r)$ of the matrix $\mathtt{A}_{j+1}(q_{j+1}^{i-1}, [i{:}d, 1{:}(i-1)]) = \mathtt{A}_{j+1}$. In other words, by adding ι_i to row $q_j^{i-1}(r)$ of $\mathtt{A}_j$, we obtain row $q_{j+1}^{i-1}(\mathtt{M}_j(1,r))$ of $\mathtt{A}_{j+1}$. In formula,

$$\mathtt{M}_j(i, q_j^{i-1}) = q_{j+1}^{i-1}(\mathtt{M}_j(i,:)), \qquad i = 1{:}d. \tag{6}$$

It remains to construct q_j. For this, assume that p_j is the permutation that puts the last column of $\mathtt{A}_j =: \mathtt{Aj}$ into increasing order with the least number of interchanges, as would be (contained in) the vector `p` generated by the `MATLAB` statement

```
[ignore,p] = sort(Aj(:,d))
```

and consider the matrix $\mathtt{B} := \mathtt{A}_j(p_j,:)$. Then the last column of B is in increasing order. Further, since p_j fails to reorder any two rows with the same entry in column d, each of the $j+1$ submatrices

$$\mathtt{B}(\mathtt{find}(\mathtt{B}(:,d)\mathtt{==}i),[1:(d-1)]), \qquad i=0:j,$$

is in lexicographic order (given that $\mathtt{A}_j$ is in lexicographic order, by assumption). It follows that $\mathtt{B}(:,[d,1:(d-1)])$ is in lexicographic order. Since the rows of B comprise the elements of $\mathtt{A}_j$, it follows that

$$\mathtt{A}_j(p_j,[d,1:(d-1)]) \;=\; \mathtt{A}_j,$$

or, equivalently,

(7) $$\mathtt{A}_j(p_j^{-1},[2:d,1]) \;=\; \mathtt{A}_j,$$

with p_j^{-1} the inverse of the permutation p_j. In other words,

$$q_j = p_j^{-1}.$$

In particular, with $\mathtt{p}= p_j$, for any vector b of length $\#A_j(d)$, the vector $\mathtt{b}(q_j)$ can be obtained by the MATLAB statement

```
b(p) = b
```

i.e., without having to generate the vector $\mathtt{q}= q_j$ explicitly (as could be done by the command `[ignore,q] = sort(p)`).

We have proved the following

Proposition. *For $a \in \mathbb{Z}_+^d$, let $N(\alpha) =: \#A_{|\alpha|-1}(d) + n(\alpha)$ be its position in the grlex order, i.e., the unique linear order $N : \mathbb{Z}_+^d \to \mathbb{N}$ that satisfies (1) and is lexicographic within each set $A_j(d) := \{\alpha \in \mathbb{Z}_+^d : |\alpha| = j\}$, $j = 0, 1, \ldots$.*

Then

$$n(\alpha+\iota_i) = \mathtt{M}_{|\alpha|}(i,n(\alpha)), \qquad \alpha\in\mathbb{Z}_+^d,\ i=1:d,$$

with (see (4))

$$\mathtt{M}_j(1,:) = (\#A_{j+1}(d) - \#A_j(d)) \;+\; (1:\#A_j(d)),$$

and (see (6))

$$\mathtt{M}_j(i,q_j^{i-1}) = q_{j+1}^{i-1}(\mathtt{M}_j(i,:)), \qquad i=2:d,$$

and q_j the inverse of the permutation p_j that puts the vector $\mathtt{m}_j$ into increasing order with the least number of interchanges, and the vector $\mathtt{m}_j$ obtainable inductively by $\mathtt{m}_0 = (0)$ and (see (3))

$$\mathtt{m}_{j+1} = (j+1, \mathtt{m}_j(1:\#A_j(2)), \mathtt{m}_j(1:\#A_j(3)), \ldots, \mathtt{m}_j), \qquad j=0,1,\ldots.$$

The matrix $\mathtt{A}_j$ can be obtained from the vector $\mathtt{m}_j$ and the permutation p_j by the assignments:

$$\begin{aligned} \mathtt{A}_j(:,d) &\;\leftarrow\; \mathtt{m}_j, \\ \mathtt{A}_j(p_j,i-1) &\;\leftarrow\; \mathtt{A}_j(:,i), \quad i=d:-1:2. \end{aligned}$$

5. Algorithm for generating $\mathtt{M}_j$ inductively

The resulting algorithm for generating $\mathtt{M}_j$ inductively is quite simple, at least in MATLAB.

Assume the following items available:

(i) The vector $\mathtt{m} := \mathtt{m}_j$ containing the last column of the matrix $\mathtt{A}_j$.

(ii) The vector $\mathtt{p} := p_j$ containing the permutation that puts $\mathtt{m}_j$ into increasing order (as could have been obtained from `m` by the MATLAB command `[ignore,p] = sort(m)`).

(iii) The vector `blaises` containing the sequence

$$(\#A_j(i) : i = 1{:}d) = (\tbinom{j+i-1}{j} : i = 1{:}d).$$

Then the following MATLAB script will produce $\mathtt{Mj} := \mathtt{M}_j$ and update `m`, `p`, and `blaises` in the process.

```
d = length(blaises); mp1 = blaises(2);
for i=2:d
   mp1 = [mp1 m(1:blaises(i))];
end

[ignore,pp1] = sort(mp1);
nAjd = blaises(d); blaises = cumsum(blaises);
rangej = (blaises(d)-nAjd) + [1:nAjd];

Mj = zeros(d,nAjd); next = 1:blaises(d);
for i=1:d
   Mj(i,:) = next(rangej); Mj(:,p) = Mj; next(pp1) = next;
end
```

6. The normalized shifted power form

The above algorithm is used in [B], a package of m-files for multivariate polynomial work, to generate the appropriate initial segment of the matrix `M` as a part of the (normalized shifted) power form of a d-variate polynomial.

If the polynomial is of degree k, then $\mathtt{M}(:, 1{:}\#A_{<k}(d))$ is so generated and carried along with the coefficient array `coefs`, of order $df \times \#A_{\le k}$, with df the dimension of the target space of the polynomial. (Even if one only deals with scalar-valued polynomials, it is very convenient in the multivariate context to be able to accommodate vector-valued polynomials, as one is likely to have to deal with gradients, Hessians, and the like.) In addition, the vector `dimPjd`, containing the sequence

$$(\#A_{\le j}(d) : j = -1{:}k),$$

is part of that form, as is the vector `center`, containing a point in $\mathbb{R}^d$ chosen somewhere in the middle of the domain on which the polynomial p represented is to be considered.

In these terms,

$$p(x) = \sum_{|\alpha| \le k} \texttt{coefs}(:, N(\alpha)) \binom{|\alpha|}{\alpha} (x - \texttt{center})^\alpha.$$

Hence, for given `x`, the value $p(\texttt{x})$ can be obtained, as `v(:,1)`, from `v` generated by the following `MATLAB` script:

```
v = coefs; xmc = x - center;
for j=k:-1:1
   rangej = dimPjd(j)+1:dimPjd(j+1);
   for i=1:d
      v(:,rangej) = v(:,rangej) + xmc(i)*v(:,dimPjd(j+1)+M(i,rangej));
   end
end
```

To be sure, the inner loop can be avoided by use of `MATLAB` commands `reshape` and `sum` or `permute`; see `mpval` in [B].

As another example, consider the construction of the directional derivative of the above polynomial, in the direction y. We have

$$D_y p(x) = \sum_{|\alpha| < k} \texttt{dcoefs}(:, N(\alpha)) \binom{|\alpha|}{\alpha} (x - \texttt{center})^\alpha,$$

with the coefficient array `dcoefs` given by

$$\texttt{dcoefs}(:, N(\alpha)) = (|\alpha| + 1) \sum_{i=1}^{d} y(i) \texttt{coefs}(:, N(\alpha + \iota_i)), \qquad \alpha \in \mathbb{Z}_+^d, \tag{8}$$

hence obtained in the following `MATLAB` script:

```
dcoefs = zeros(df,dimPjd(k+1));
for j=1:k
   rangej = dimPjd(j)+1:dimPjd(j+1);
   for i=1:d
      dcoefs(:,rangej) = dcoefs(:,rangej) + ...
                j*y(i)*coefs(:,dimPjd(j+1)+M(i,rangej));
   end
end
```

Again, the inner loop can be avoided by use of `reshape` and `sum` or `permute`; see `mpder` and `mpdir` in [B].

7. The plain (shifted) power form

To be sure, the *symmetric* nested multiplication algorithm used above is not the most efficient way to evaluate a polynomial in power form. Since all but one of its coefficients are to be multiplied by some power of x, with the resulting factors in general different from coefficient to coefficient, the minimum number of adds and of multiplies needed is each at least as big as one less than the number of terms in the form. Moreover, that lower bound is attained by plain (i.e., asymmetric) nested multiplication,

$$
\begin{array}{l}
v(\alpha) \leftarrow c(\alpha), \quad |\alpha| \le k, \\
\texttt{for } j = k{:}{-1}{:}1 \\
\quad \texttt{for } i = 1{:}d \\
\qquad \texttt{for } \alpha(1) = \cdots = \alpha(i-1) = 0 < \alpha(i), |\alpha| = j, \\
\qquad\quad v(\alpha - \iota_i) \leftarrow v(\alpha - \iota_i) + x(i)v(\alpha)
\end{array}
$$

which produces, in $v(0)$, the number

$$p(x) := \sum_{|\alpha| \le k} c(\alpha) x^\alpha, \tag{9}$$

i.e., the value at x of p from its plain power form. Note that this calculation is easy for the ordering N since, as used earlier, $\mathtt{A}_{j+1}$ is the concatenation, in order, of the segments

$$\mathtt{A}_j(1{:}\#A_j(i), (d-i+1){:}d)$$

after 1 has been added to all the entries of its $(d+1-i)$th column, $i = 1{:}d$, hence the inner assignment becomes

$$
\begin{aligned}
v(\dim \Pi_{j-2}(d) + [1{:}\#A_{j-1}(i)]) \quad \leftarrow \quad & v(\dim \Pi_{j-2}(d) + [1{:}\#A_{j-1}(i)]) + \\
& x(d+1-i) v(\dim \Pi_{j-1}(d) + [\#A_j(i-1){+}1 : \#A_j(i)]).
\end{aligned}
$$

The sequences

$$\mathtt{blaise}(:, j+1) := (\#A_j(i) : i = 1{:}d), \quad j = 0{:}k$$

needed here would be carried along as part of the plain (shifted) power form.

The normalized (shifted) power form would be at a disadvantage here since, for it, the initial assignment here would have to be changed to

$$v(\alpha) \leftarrow c(\alpha) \binom{|\alpha|}{\alpha}, \quad |\alpha| \le k,$$

thus doubling the multiplications needed and requiring the vector

$$\mathtt{b} := (\tbinom{|\alpha|}{\alpha} : |\alpha| \le k).$$

To be sure, that vector `b` is easily generated with the aid of the matrix `M` and the vector `dimPjd` used earlier, as in the following `MATLAB` script:

```
b = zeros(1,dimPjd(end)); b(1) = 1;
for j=1:k
   range = dimPjd(j)+1:dimPjd(j+1); bj1 = b(range);
   for i=1:d
      temp = dimPjd(j+1)+M(i,range);
      b(temp) = b(temp) + bj1;
   end
end
```

But, why bother with the normalized power form at all, when the plain power form is cheaper to evaluate? Aside from the satisfaction of having results that are independent of the particular ordering of the independent variables (though some ordering will occur in any calculation on a serial computer), the symmetry becomes essential when working with piecewise polynomials, using the closely related Bernstein-Bézier form.

Indeed, in this form, a polynomial of degree $\le k$ is written

$$p(x) = \sum_{|\alpha|=k} \xi(x)^\alpha \binom{|\alpha|}{\alpha} c(\alpha),$$

with $\xi(x)$ the affine or barycentric coordinates of x with respect to some affinely independent $(d+1)$-set V, i.e.,

$$x = \sum_{v\in V} \xi_v(x)v, \quad \text{with} \quad \sum_{v\in V} \xi_v(x) = 1,$$

and, correspondingly, $\alpha = (\alpha(v) : v \in V) \in \mathbb{Z}_+^{d+1}$. In other words, it is in the *normalized* power form of a polynomial in $d+1$ variables.

The normalization is also handy, as we saw earlier, in the construction of a directional derivative. For the plain power form, the corresponding formula is

$$D_y p(x) = \sum_{|\alpha|<k} \texttt{dcoefs}(:, N(\alpha))(x - \texttt{center})^\alpha,$$

with the coefficient array `dcoefs` given by

$$\texttt{dcoefs}(:, N(\alpha)) = \sum_{i=1}^{d} (\alpha(i)+1)y(i)\texttt{coefs}(:, N(\alpha+\iota_i)), \qquad \alpha \in \mathbb{Z}_+^d. \tag{10}$$

Offhand, we now need, in addition to `M`, the inverse map, $N(\alpha) \mapsto \alpha$, e.g., the matrices $\mathtt{A}_j$, $j = 1{:}(k-1)$, and the Proposition suggests a particular way for their generation. The alternative is to carry along the vector `b`, in order to normalize, differentiate the normalized form, then unnormalize. This requires (roughly) two additional multiplications per coefficient, but replaces the d multiplications per coefficient in (10) by one multiplication per coefficient in (8).

8. List of m-files presently available in [B]

Here, at the editors' request, is a brief listing of the m-files for work with multivariate polynomials presently available in [B]. By the time you read this, the actual list may well be more extensive. No claim is made of particular efficiency.

`mp2fm(mp)` converts between the normalized and the plain power forms.

`mpapi(t,f,tol,center)` constructs the least interpolant (in normalized power form, optionally centered at the specified `center`) to the given data $(\mathtt{t}(:,\mathtt{i}), \mathtt{f}(:,\mathtt{i}))$, all `i`, with `tol` an optional value for the relative tolerance used during Gauss elimination (by degrees) to determine whether a proposed pivot element is zero.

`mpbrk(mp,part)` supplies parts of the multivariate polynomial in `mp`.

`mpcov(mp,A)` makes the linear change of variables $x \mapsto \mathtt{A}x$.

`mpder(mp,`X`)` provides the derivative $(\prod_{x\in X} D_x)p$ of the multivariate polynomial p in `mp`.

`mpdir(mp,directions)` provides the directional derivative in all the directions specified by columns of the matrix `directions`. For example, if p is the `d`-variate `m`-vector-valued polynomial described by `mp`, and `x` is some point in its domain, then

$$\mathtt{reshape}(\mathtt{mpval}(\mathtt{mpdir}(\mathtt{mp}, \mathtt{eye}(\mathtt{d})), \mathtt{x}), \mathtt{m}, \mathtt{d})$$

is the Jacobian of that function at that point.

`mpmak(coefs,d,k,center,blaise,mm)` puts together the normalized power form (centered at `center`, default is the origin) of the `d`-variate `k`-th degree polynomial whose normalized power coefficients are given by the array `coefs`. Supplying the combinatorial and indexing information `blaise` and `mm` obviates the need to generate that information.

`mpnext(blaises,mp)` is used in the construction of polynomial forms.

`mpshft(mp,newcenter)` shifts the polynomial form to the given `newcenter`.

`mptest` is a script file containing various tests of these m-files.

`mpval(p,x)` provides the value(s) at (the points provided by the columns of) `x` for the polynomial specified by `p`.

`dcube(d)` provides the corners of the `d`-cube, needed in `mptest`.

Thomas Grandine of Boeing has written a `C` version of least interpolation that covers arbitrary interpolation conditions. His code suite, called `MVP`, is available, via email or over the web, from `netlib`.

9. One other application

The map $N(\alpha) \mapsto n(\alpha + \iota_i)$ is also handy for generating the polynomials

$$p_i : x \mapsto x(i)p(x), \qquad i = 1{:}d,$$

from a given polynomial p, as is needed in the construction of good generating sets for polynomial ideals. Indeed, with

$$p = \sum_\alpha c(\alpha) x^\alpha,$$

we have

$$p_i = \sum_\alpha c(\alpha) x^{\alpha + \iota_i} = \sum_\beta c_i(\beta) x^\beta,$$

with

$$c_i(\beta) := \begin{cases} c(\beta - \iota_i), & \beta - \iota_i \in \mathbb{Z}_+^d, \\ 0, & \text{otherwise.} \end{cases}$$

Hence, if `coefs` is the sequence containing the coefficients c according to the grlex ordering, and the matrix `M` from (2), as well as the vector `blaise(d,:)` containing the sequence $(\#A_{j-1}(d) : j = 1{:}k+1)$ mentioned earlier, are available, then

```
cb = cumsum(blaise(d,:));
addon = zeros(1,cb(end));
addon(1+cb(1:end-1)) = blaise(d,1:end-1);
addon = cumsum(addon);
coefsi(M(i,:)+addon) = coefs;
```

provides the correspondingly ordered sequence of coefficients for p_i (assuming that `coefsi` has been initialized as a zero vector of the appropriate length).

In fact, it might be worthwhile to carry the vector `addon` as part of the power form since it gives the map

$$N(\alpha) \mapsto N(\alpha + \iota_i) = \texttt{addon}(N(\alpha)) + \texttt{M}(i, N(\alpha)).$$

References

[B] C. de Boor, A package of m-files for work with multivariate polynomials, available by anonymous ftp from `ftp.cs.wisc.edu/Approx`

[BR] C. de Boor and A. Ron, Computational aspects of polynomial interpolation in several variables. Math. Comp. 58(1992), 705–727.

[CLO] D. Cox, J. Little, and D. O'Shea, Ideals, Varieties, and Algorithms. Springer-Verlag, Heidelberg, 1992.