LA-UR -76-541

Conf 760508--1

TITLE: USE OF SYMBOLIC AND NUMERIC METHODS IN AN ALGORITHM FOR THE APPROXIMATION OF MULTIVARIATE FUNCTIONS

AUTHOR(S): DAVID K. KAHANER
MARK B. WELLS

SUBMITTED TO: 1976 ACM SYMPOSIUM ON SYMBOLIC AND ALGEBRAIC COMPUTATION,
Yorktown Heights, NY, Aug. 10-12, 1976.

los alamos
scientific laboratory
of the University of California
LOS ALAMOS, NEW MEXICO 87544

Form No. 836
St. No. 2629
1/75

UNITED STATES
ENERGY RESEARCH AND
DEVELOPMENT ADMINISTRATION
CONTRACT W-7405-ENG. 36

MASTER

USE OF SYMBOLIC AND NUMERIC METHODS IN AN ALGORITHM
FOR THE APPROXIMATION OF MULTIVARIATE FUNCTIONS*

by

David K. Kahaner and Mark B. Wells

Abstract: The calculation of a polynomial interpolant over a simplex for a given function of n variables is discussed. Polynomial manipulation is required for constructing these interpolants and matrix manipulation is necessary for evaluating them. Use of an extensible language in which various manipulations could easily be expressed greatly facilitated development of the general approximation algorithm of which this calculation is a part.

*University of California, Los Alamos Scientific Laboratory, Los Alamos, New Mexico. Work performed under the auspices of the Division of Physical Research/Molecular Sciences of the Energy Research and Development Administration.

1. INTRODUCTION

The computer is a powerful tool for performing monotonous calculations correctly, be they numeric, symbolic, or other. Indeed, there are important problems that quite naturally require manipulations of various kinds of data during calculation of their solution. Thus, an important goal of computer science research today is the development and understanding of complex and useful algorithms that involve assorted manipulations. This work has historically been hampered by lack of appropriate programming languages for expression of such general algorithms. Fortran (say) is unsuited for most symbolic manipulation, while Reduce (say) is not designed for efficient numerical manipulation. An approach to language extensibility recently pursued by various language designers [1, 3, 6] shows considerable promise in resolving this dilemma.

In this paper, we discuss aspects of a particular algorithm, namely that of finding an approximation to a real function of n variables, that uses both symbolic and numeric manipulations. Development of this algorithm was motivated in part by an actual application; it is being used in a study of the optimal design of a geothermal energy extraction plant [4]. Our experience here with the extensible language Madcap [6], in which the approximation algorithm was developed, certainly corroborates the importance of very high level, general-purpose languages for the design of involved algorithms.

2. THE PROBLEM

One is given a function of n variables f (i.e., a procedure with n input parameters). Each function evaluation is the result of a time-consuming and inexact calculation. The function values themselves may be of interest, but more often they are useful for some other purpose such as locating local extrema, plotting projections to understand relations between variables, etc. Consequently, the specific points at which f is to be evaluated are not known in advance, but are determined by later use. Very often a grid is placed on the domain of interest, the function evaluated at each grid point, and some local approximation (often an interpolant and usually a polynomial) is used whenever function values are needed in order to save time.

If the input function is smooth and its domain small, a regular grid is usually sufficient. But if, as is normally the case, significant structure is present the grid spacing required for resolution is too small to be practical. The problem our algorithm solves is the automatic construction of an adequate continuous piecewise polynomial interpolant given a requested absolute accuracy ϵ. The decomposition of the domain (generally irregular) is done with the goal of using as few expensive function evaluations as possible.

3. THE METHOD

The approach taken follows closely that of an algorithm for adaptive numerical quadrature in n dimensions which has been developed by the authors [2]. The idea is progressively to subdivide n-space into smaller and smaller pieces until the approximation over each piece is sufficiently accurate. The strategy of subdivision adapts itself to the behavior of the function in that fine subdivision will occur only in regions of space where the function is difficult to approximate.

The basic unit of subdivision is the n-simplex, a triangle for $n = 2$, a tetrahedron for $n = 3$, etc. This cell was chosen for two basic reasons: (1) Simplexes use function evaluations efficiently, since if n-space is tessellated into n-simplexes then a mesh point is a vertex of $(n+1)!$ simplexes whereas for n-cubes the corresponding number is only 2^n. (2) Doubling the degree of polynomial approximation can make use of the same function evaluations as subdividing the simplex, since when a simplex is subdivided into 2^n subsimplexes, those subsimplexes into 2^n sub-subsimplexes, etc. to m levels, the total number of mesh points is $\binom{n + 2^m}{n}$ and this is precisely the number of terms in a polynomial of degree 2^m in n variables.

We cannot delve deeply into the algorithm here, we only discuss the construction and use of the polynomial interpolant over a simplex, since there both symbolic and numerical tools come into play.

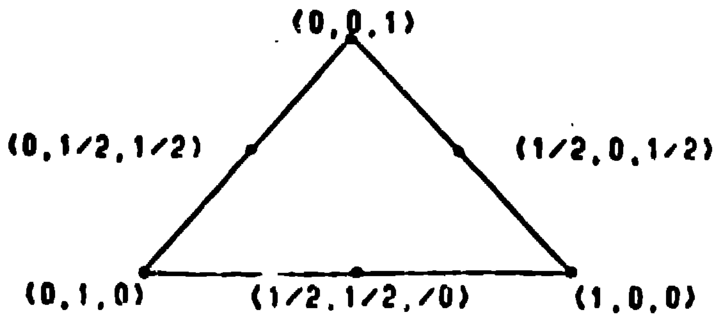
We assume f is defined over an n-simplex and wish to construct a dth degree polynomial interpolant. For purposes of illustration, we consider $n = 2$ and $d = 2$. The simplex is then a triangle and six $\left[= \binom{n + d}{n}\right]$ points are required to determine the polynomial since that is the number of monomials of degree ≤ 2 in two independent physical variables. An efficient method of constructing this approximation is based on a scheme of Silvester [5] which uses a barycentric coordinate system over the simplex and produces a polynomial in three [= n+1] barycentric variables. The method precomputes certain basic polynomials over a general simplex. The jth polynomial is constructed to evaluate to 1 at the jth of six selected points and to 0 at the others. A simple linear combination of these basic polynomials then yields the interpolant for a given specific simplex. The six points used are, in barycentric coordinates,

$$\left\langle c_0/d \; , \; c_1/d \; , \; c_2/d \right\rangle$$

where $c = \langle c_0, c_1, c_2 \rangle$ is a "composition" of 2[= d] into 3[= n+1] parts allowing zero parts, that is, the c_i are integers such that

$$0 \le c_i \le 2 \quad \text{and} \quad d = \sum_{0 \le i \le 2} c_i = 2 \quad .$$

[As mentioned earlier, when $d = 2^m$, these points are vertices of the sub-subcells to level m, hence Silvester's scheme merges nicely with our "adaptive" algorithm.] Figure 1 shows the triangle, its points of evaluation for a quadratic polynomial given in barycentric coordinates and the basic polynomials associated with those points given in barycentric variables.



$$Q_{(1,0,0)} = z_0(2z_0-1)$$
$$Q_{(0,1,0)} = z_1(2z_1-1)$$
$$Q_{(0,0,1)} = z_2(2z_2-1)$$
$$Q_{(1/2,1/2,0)} = 4z_0z_1$$
$$Q_{(1/2,0,1/2)} = 4z_0z_2$$
$$Q_{(0,1/2,1/2)} = 4z_1z_2$$

Figure 1: Polynomial Bases for n=2, d=2

4. SYMBOLIC COMPUTATION

Let $P = \langle c_0/d \; , \; c_1/d \; , \; \ldots, \; c_n/d \rangle$ be one of the points at which a basic polynomial is to be calculated, that is

$$\sum_{0 \le i \le n} \frac{c_i}{d} = 1 \quad \text{and} \quad d = \sum_{0 \le i \le n} c_i$$

for integral c_i . The polynomial is

$$Q = \prod_{0<i<n} R_i(z_i)$$

where

$$R_i(z_i) = \frac{1}{c_i!} \prod_{0<k<c_i} d \cdot z_i - k \quad .$$

Each R_i is a polynomial in the single distinct variable z_i and Q is a polynomial in the n + 1 variables z_i, $0 \leq i \leq n$. It is straightforward to verify that Q equals 1 at the point p and equals 0 at all other evaluation points.

Computer calculation of Q is also straightforward provided we can do polynomial arithmetic easily. The algorithm is expressed as a procedure in the Madcap language in Fig. 2. This program makes use of a "space" of polynomials in which polynomials are represented by tuples of coefficients and can be manipulated with various operators. This program uses the

«
$\underline{c}$: composition
$\underline{n} \leftarrow \#c-1;\ \underline{d} \leftarrow \Sigma_{0\leq i\leq n} c_i$ @ POLYNOMIAL
$\underline{z} \leftarrow$ variable↑POLYNOMIAL
$Q \leftarrow \Pi[\text{for } 0\leq \underline{i}\leq n:\ \pi[\text{for } 0\leq \underline{k}<c_i:\ d\cdot z-k)\cdot \frac{1}{c_i!}]$
»

Figure 2: Calculation of Interpolating Polynomial

dot multiply (·) and subtract (-) to form the linear polynomials and for constant multiplication, the small product (π) to form the univariate R's (the innermost product), and the big product (Π) to form the multivariate Q.

Now let $p_0, p_1, \ldots, p_{v-1}$ be the $\binom{n+d}{n}$ evaluation points and let $Q_0, Q_1, \ldots, Q_{v-1}$ be the corresponding basis polynomials (independent of f). The linear combination

$$A = Q_0 \cdot f(p_0) + Q_1 \cdot f(p_1) + \cdots + Q_{v-1} \cdot f(p_{v-1})$$

is then the polynomial approximation to f over a specific simplex in barycentric variables. [This formula shows the need for a plus (+) operator in our polynomial space. Other useful operators include subscript for picking off coefficients and juxtaposition for evaluating a polynomial at a point in n-space.] This polynomial can now be evaluated without repeating the time-consuming computation needed to construct it.

5. NUMERIC COMPUTATION

Numeric computation in the form of matrix arithmetic arises naturally in the evaluation of the polynomial interpolant, essentially for the conversion between points given in physical coordinates to points given in barycentric coordinates. Let (x_0,y_0), (x_1,y_1), (x_2,y_2) be vertices of a triangle given in physical coordinates, and $p = (x,y)$ be an arbitrary point in 2-space. The barycentric coordinates $\langle b_0,b_1,b_2\rangle$ of p relative to the triangle may be calculated from the relations

$$b_0x_0 + b_1x_1 + b_2x_2 = x$$

$$b_0y_0 + b_1y_1 + b_2y_2 = y$$

and

$$b_0 + b_1 + b_2 = 1 \quad .$$

In matrix form, this is

$$\langle b_0, b_1, b_2\rangle = \langle x, y, 1\rangle \cdot T^{-1}$$

where

$$T = \begin{pmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{pmatrix} \quad .$$

Thus, to evaluate the polynomial A at one of many points p, one first calculates the barycentric coordinates of p using matrix operations, and then substitutes these as values of the barycentric variables of A.

A program for calculating the point $= \langle b_0, b_1, \ldots, b_n \rangle$ given a physical point $X = \langle x_0, x_1, \ldots, x_{n-1} \rangle$ and the precomputed inverse to T appears in Fig. 3. This and other matrix

```
«
X: point; T.inverse: @MATRIX
v ← ⟨⟨x_i: 0≤i<n; 1⟩⟩ @ MATRIX
B ← v · T.inverse
»
```

Figure 3: Conversion from Physical to Barycentric Coordinates

manipulation programs make use of a space of matrices (including vectors) with representation via tuples of tuples and the operators of multiply (·) and invert (exponentiation to -1). Thus, numeric as well as symbolic manipulation is expressed with natural notation.

References

[1] Dahl, O. J., Myhrhaug, B., and Nygaard, K., *The SIMULA 67 Common Base Language*, Publication S-22, Norwegian Computing Center, Oslo, 1970.

[2] Kahaner, D. K. and Wells, M. B., "An Algorithm for n-dimensional Adaptive Quadrature Using Advanced Programming Techniques" (submitted for publication).

[3] Liskov, B. and Zilles, S., "Programming With Abstract Data Types," *Proceedings of ACM SIGPLAN Conference on Very High Level Languages*, SIGPLAN notices, Vol. 9, no. 4 (April 1974), pp. 50-59.

[4] Milora, S. L. and Tester, J. W., *Geothermal Energy as a Source of Electric Power*, MIT Press, 1976.

[5] Silvester, P., "Symmetric Quadrature Formulae for Simplexes," *Mathematics of Computation*, Vol. 24, no. 109, (January 1970), pp. 95-100.

[6] Wells, M. B. and Cornwell, F. L., "A Data Type Encapsulation Scheme Utilizing Base Language Operators," *Proceedings of ACM Conference on Data: Abstraction, Definition, and Structure*, Salt Lake City, March, 1976.