ROCKY MOUNTAIN
JOURNAL OF MATHEMATICS
Volume 14, Number 1, Winter 1984

C^1 SURFACE INTERPOLATION FOR SCATTERED DATA ON A SPHERE

CHARLES L. LAWSON

ABSTRACT. This paper describes an algorithm for constructing a smooth computable function f defined over the surface of a sphere and interpolating a set of n data values u_i associated with n locations p_i on the surface of the sphere. The interpolation function f will be continuous and have continuous first partial derivatives. The locations p_i are not required to lie on any type of regular grid.

1. Introduction. The problem of constructively defining a smooth surface that interpolates data defined at scattered points in the plane has been treated in different ways by a number of authors. For surveys of this work up to 1977 see [2] and [9].

We consider here the analogous problem for data defined at scattered points over the surface of a sphere. When data are defined over only a portion of the surface of a sphere, it may be most reasonable to map that portion of the spherical surface to a planar region, using a C^1 mapping function, and treat the problem by an algorithm designed for the planar domain problem. However, when the data are scattered over the whole surface, and it is desired to produce a C^1 interpolation function defined over the entire surface, it seems necessary, or at least very desirable, to deal with the problem directly in the spherical setting. In particular, there is no C^1 function that will map the entire surface of a sphere to a bounded planar region.

2. The problem. Let S denote the surface of the unit sphere in three-space. Given points p_i, $i = 1, \ldots, n$, the problem is to construct a computable function f defined and having C^1 continuity over S and satisfying the interpolation conditions $f(p_i) = u_i$ for $i = 1, \ldots, n$.

2.1. Relevant properties of C^1 functions on S. A function, f, defined on S is differentiable at a point p_0 in S if and only if there exists a three-vector g_0 satisfying

The development described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

$$\lim_{\substack{\|dp\|\to 0 \\ p_0+dp\in S}} \frac{f(p_0+dp)-(f(p_0)+g_0^T dp)}{\|dp\|} = 0. \tag{1}$$

Let T_0 denote the tangent plane to the sphere at the point p_0. Since the perturbed points $p_0 + dp$ in (1) are required to lie in S, the normalized perturbation vectors $dp/\|dp\|$ approach the plane T_0 as $\|dp\|$ approaches zero. It follows that if a vector g_0 satisfies (1), then so also does any vector of the form $g_0 + h$ where h is orthogonal to the tangent plane T_0, i.e., where h is a multiple of the vector from the origin to p_0.

To resolve this nonuniqueness of vectors g_0 satisfying (1) we will standardize on the shortest such vector. This vector is distinguished among vectors g_0 satisfying (1) by the property of being orthogonal to the position vector from the origin to p_0, or equivalently by the property that the point $p_0 + g_0$ lies in the tangent plane T_0. We will call this vector g_0 the gradient vector of f at p_0.

Note that the fact that f has a restricted domain, namely S, is an essential part of this definition. For example if f is the restriction to S of some function $\hat{f}$ defined in an open neighborhood of three-space containing p_0, it is entirely possible that $\hat{f}$ may be differentiable at p_0 and have a unique gradient vector g that is different from the (minimal length) gradient vector g_0 of f. In such a case, however, g_0 will be the orthogonal projection of g onto the 2-D subspace parallel to the tangent plane T_0.

Let U be a region of S containing p_0 and not extending more than $\pi/2$ radians away from p_0 in any direction. Let k be the one-to-one mapping of points of U to their orthogonal projections in T_0. Let U_0 be the region in T_0 to which U is mapped by k. Define the function f_0 on U_0 by $f_0(t) = f(k^{-1}(t))$. Note that the point p_0 is in both the domains of f and f_0. If f is differentiable at p_0 with gradient vector g_0, then also f_0 is differentiable at p_0 with gradient vector g_0. We will make use of this local equivalence of f and f_0 later in deriving an algorithm for estimating the gradient of f from discrete data.

We will say a function defined on S is in the class C^1 if there is a continuous 3-D vector-valued function g defined on S such that for each point $p_0 \in S$, $g(p_0)$ is orthogonal to the vector from the origin to p_0 and satisfies the condition ascribed to g_0 in (1).

3. Major steps of the solution method. The approach to be described has the same major steps as the method for the analogous planar problem given in [6]. These steps are

1. build a triangular grid on S having the given points p_i as vertices,
2. estimate the gradient vector g_i at each point p_i, and
3. evaluate the interpolation function f at an arbitrary point p in S by
   (a) Looking up p in the grid to find the triangle containing p and

(b) Computing $f(p)$ by an interpolation method using the given function values u_i and the estimated gradient vectors g_i at the three vertices of the enclosing triangle.

3.1. Data structures. In the algorithms to be described the points p_i will be represented by their Cartesian coordinates. It will be convenient in the following to let the same symbol denote either a point or the 3-D vector from the origin to the point. In particular, points in S are represented by vectors of unit Euclidean length. Each triangle will have an index number and will be represented by a set of six pointers identifying the three adjacent triangles and the three vertex points. This is exactly the same data structure as was used in [6].

If triangle t has vertices whose indices are A, B, and C in counterclockwise order, and whose adjacent triangle indices are a, b, and c with triangle a opposite vertex A, b opposite B, and c opposite C, the six pointers representing triangle t would be stored in one of the following three permutations:

$$a, b, c, B, C, A$$
$$b, c, a, C, A, B$$
$$c, a, b, A, B, C.$$

All access to these pointers is done via three very short subroutines. Thus the actual storage mode for these pointers is "hidden" from the rest of the program. By appropriate programming of these three subroutines the pointers can be packed to save storage.

The array storage requirements of this algorithm are thus $3n$ locations for the vectors p_i, $i = 1, \ldots, n$, n locations for the data values u_i, $i = 1, \ldots, n$, $12n$ locations for the triangle pointers (This is based on six pointers per triangle and at most $2n$-4 triangles. This storage requirement can easily be reduced by packing), $3n$ locations for the gradient vectors g_i, $i = 1, \ldots, n$, and n locations for a permutation vector used only while building the grid (This storage could be overlaid by the gradient vector array or could be eliminated entirely by minor changes in the program design).

3.2. Determinantal tests and grid look-up. Let p_1, p_2, and p_3 be three-vectors having unit Euclidean length. Let $\det(p_1, p_2, p_3)$ denote the determinant of the 3×3 matrix whose column vectors are p_1, p_2, p_3 in that order. If $\Delta = \det(p_1, p_2, p_3) \neq 0$, then no two of the vectors form an angle of zero or π, and the three vectors do not all lie in a single plane through the origin. In this case a proper spherical triangle can be formed by connecting each of the three pairs of points by the shorter arc of the great circle in S determined by that pair of points. Thus each arc will have

length less than π. This triangle divides S into two regions. The smaller region is to be regarded as the interior of the triangle. If $\Delta > 0$, an observer traversing the edges of the triangle with the interior of the triangle to the left will visit the vertices in the order p_1, p_2, p_3. If $\Delta < 0$, the ordering would be reversed. We will always order the vertices of triangles so that $\Delta > 0$.

Let p_1, p_2, p_3, be vertices of a proper triangle t in S with $\Delta > 0$. Regarding q as a variable three-vector in S, note that the quantity $s_1 = \det(q, p_2, p_3)$ is proportional to the distance of q from the plane determined by the vectors p_2 and p_3 with the sign of s_1 being positive if q is on the same side of the (p_2, p_3)-plane as p_1 and negative if q is on the opposite side. Thus a point $q \in S$ is inside the triangle t if and only if the three quantities

$$
\begin{aligned}
s_1 &= \det(q, p_2, p_3) \\
s_2 &= \det(p_1, q, p_3) \\
s_3 &= \det(p_1, p_2, q)
\end{aligned} \tag{2}
$$

are all nonnegative.

Our algorithm for finding a triangle containing a given point q consists in computing the quantities s_1, s_2 and s_3 for some triangle t and then either accepting t as the containing triangle if all $s_i \geqq 0$ or else moving to the neighboring triangle across the edge opposite vertex p_i if s_i is the first of the test quantities found to be negative. If there is no neighboring triangle across this edge the search stops, returning this information. Otherwise the search continues by computing the test quantities in the neighboring triangle.

Rounding errors in computing a 3×3 determinant causing inconsistent sign determination could conceivably lead to cycling in the look-up process or to the construction of topologically impossible edges in the grid construction. (Grid construction will be described in §3.3). Consider for example four points $p_1, \ldots, p_4$ that lie in order along an arc of a great circle, the arc having length less than π. The true mathematical value of the determinant of the 3×3 matrix formed using any three of these vectors is zero. Using finite precision coordinates and finite precision floating point arithmetic these determinants will generally not be computed as zero. A nonzero result does not in itself cause a serious problem but the possibility of inconsistency in the evaluation of related determinants can.

To illustrate the hazard suppose that with $p_1, \ldots, p_4$ as above the computed value of $\det(p_1, p_2, p_3)$ is positive and (p_1, p_2, p_3) is accepted as a triangle in the grid. Then suppose p_4 is tested for inclusion in this triangle. It is possible that all of the determinants $\det(p_4, p_2, p_3)$, $\det(p_1, p_4, p_3)$, and $\det(p_1, p_2, p_4)$ might evaluate nonnegative. This would lead to the erroneous conclusion that p_4 is contained in the triangle (p_1, p_2, p_3) and

various topologically incorrect edges would be constructed to incorporate p_4 into the grid. Using a tolerance ε such that all results between $-\varepsilon$ and ε are treated as zero does not solve the problem. We have had good luck using double precision evaluation of the determinants and strict zero tests. We have also had success with single precision determinant evaluation if we randomized the order in which the points p_i were considered for inclusion in the grid.

One way to assure consistency while sacrificing some accuracy would be to truncate all coordinate values to a small enough number of bits to permit the determinant evaluation to be done exactly. For example, on a machine carrying fourteen hexadecimal digits of significance in a double precision number, one might round all coordinates to the 2^{-17} bit. The smallest nonzero bit that could occur in the product of three such numbers would be the 2^{-51} bit. The coordinates do not exceed one in magnitude; so the same is true of their products. These products and the sum of up to six such products can be held exactly in a normalized floating point number carring fourteen hexadecimal digits. Thus determinants of 3×3 matrices could be computed exactly.

3.3. Constructing the triangular grid. The convex hull of a finite set of points in the plane is the smallest convex polygon containing the entire point set. We need an analogous notion, which we will call the spherical convex hull, for points on the surface S of the unit sphere. Let P be a finite set of points in S. If there is no plane that strictly separates the origin from P, we will say the whole surface S is the spherical convex hull of P. Alternatively, if there is a plane strictly separating the origin from P, let C be the smallest convex cone with its vertex at the origin and containing the set P. The intersection of C with S will be called the spherical convex hull of P. This region will lie strictly within some hemisphere of S.

A triangular grid with n vertices and covering all of S will have $2n$-4 triangles. A grid that covers a spherically convex proper subset of S and has n vertices and b boundary edges will have $2n$-b-2 triangles. Note that $2n$ can always be used as an upper bound on the number of triangles. Our method of constructing a triangular grid using a given finite point set P in S as vertices will be a sequential process that alters a grid covering the spherical convex hull of some set of k points of P to obtain a grid covering the spherical convex hull of these k points plus one more.

Algorithms of this type can be divided into (at least) three subtypes.

(a) First find the boundary points of the (spherical) convex hull of P and construct a triangular grid for these points. Then in the remaining sequential part of the algorithm each new point is known to lie in some triangle of the current grid.

(b) Preprocess the points of P into an ordering that assures that each

new point will be stricty outside the (spherical) convex hull of the preceding points.

(c) Do no prepreprocessing and be prepared for each new point to be either inside or outside the (spherical) convex hull of the preceding points.

With subtypes (a) and (b) one is adding extra code and execution time for a preprocessing stage in the hope of permitting the subsequent sequential phase to be simpler and execute faster. We have at different times developed algorithms for the planar problem representing each of these subtypes. The algorithm of [6] is of subtype (b). My present inclination is to prefer subtype (c) as I think it permits the total program to be simpler and probably is not significantly slower if in fact it is any slower. More specifically it does not require storage for a separate data structure to keep track of boundary points as was the case in [6].

Our approach then is to form one initial triangle and then loop through the remaining $n - 3$ points adding one at a time and modifying the triangular grid at each stage to cover the spherical convex hull of all the points so far considered. Each new point may be either inside or outside the grid so far constructed. In the class of problems for which this method is primarily intended, i.e., problems in which the data are scattered quite generally over all of S, a stage will be reached at which the spherical convex hull is all of S. Thereafter all additional points will necessarily lie inside the grid so far constructed since the grid will cover all of S. The user can cause this full coverage of S to happen early by arranging that the first four points to be processed are located such that the tetrahedron with these four points as vertices contains the origin as a strictly interior point. The triangular grid based on these four points will cover all of S.

Initially the algorithm seeks three points with which to construct the first triangle. The first vector p_1 is accepted unconditionally. The remaining vectors are scanned for the first one whose inner product with p_1 lies between cos 179° and cos 1°, i.e., between -0.99985 and 0.99985. Pointers are swapped to relabel this vector as p_2. The remaining vectors are scanned to find one whose determinant along with p_1 and p_2 exceeds 0.001 in magnitude. Such a vector is relabeled as p_3. The vectors p_2 and p_3 are then swapped if necessary to assure that $\det(p_1, p_2, p_3)$ is positive. This completes the construction of the first triangle.

We may now assume a grid based on $k - 1$ points has been constructed and the next point, p_k, is to be introduced. A look-up is done using the method described in §3.2. This look-up either finds a triangle t containing p_k, or else finds a triangle t such that p_k is outside this triangle relative to a side of the triangle beyond which there is no adjacent triangle. In the first case, the single triangle t having vertex points p_A, p_B, p_C will be replaced by three triangles having vertex points (p_k, p_B, p_C), (p_A, p_k, p_C),

and (p_A, p_B, p_k) respectively. The algorithm then does a grid improvement phase to be described subsequently. In the second possible outcome of the look-up process, the point p_k is strictly outside the spherical convex hull of the preceding $k - 1$ points, and in particular it is outside an edge of triangle t that constitutes a portion of the boundary of the spherical convex hull. In this case one new triangle will be formed by connecting p_k to the two ends of the edge of t that gave a negative s_i value in the look-up testing (See §3.2).

The algorithm next scans the current grid boundary points in both directions from the new triangle and connects p_k to all other boundary points that result in the creation of proper spherical triangles (See §3.2). The algorithm then does grid improvement.

3.3.1. Grid improvement. When two adjacent spherical triangles form a strictly convex spherical quadrilateral, there arises the possibility of replacing these two triangles by the two that occur when the quadrilateral is partitioned by its other diagonal. One must estalish a criterion for choosing between the two possible dissections of a quadrilateral. This issue was discussed for the planar case in [6] where it was shown that three differently stated criteria were mathematically equivalent. In the spherical setting a fourth criterion with considerable intuitive appeal can be formulated and it is easily seen to be equivalent to the "circle test" of [6].

Let p_1, p_2, p_3, and p_4 be the vertices, in counterclockwise order, of a spherical quadrilateral in S. Assume all four of the potential triangles $(p_1 p_2 p_3)$, $(p_2 p_3 p_4)$, $(p_3 p_4 p_1)$, and $(p_4 p_1 p_2)$ would be proper spherical triangles. One choice would be to connect points p_1 and p_3 forming triangles $(p_1 p_2 p_3)$ and $(p_3 p_4 p_1)$ while the other choice would be to connect points p_2 and p_4 forming triangles $(p_2 p_3 p_4)$ and $(p_4 p_1 p_2)$. Consider the $3D$ polyhedron underlying the spherical triangular grid. If the four points under consideraton are not coplanar then one choice will give underlying planar triangular faces that could be faces of a convex polyhedron and the other choice will not. Therefore our new criterion is a preference to make the underlying $3D$ polyhedron convex.

Another way to describe this criterion is to consider the unique line L from the origin that intersects both of the lines $p_1 p_3$ and $p_2 p_4$. If p_1, p_2, p_3, and p_4 are not coplanar the two lines will intersect L at two distinct points. We construct the one of these two lines that intersects L furthest from the origin. We implement this test by computing $d = \det(p_2 - p_1, p_3 - p_1, p_4 - p_1)$ and constructing the line $p_2 p_4$ if $d > 0$ and constructing $p_1 p_3$ if $d < 0$. Either line can be used if $d = 0$.

After a new point, say p_k, is connected into the grid, each edge that is opposite p_k in some triangle is a candidate for swapping. Thus if there is

a triangle $p_k p_2 p_4$ and an adjacent triangle $p_2 p_3 p_4$, the edge $p_2 p_4$ will be replaced by the edge $p_k p_3$ if $\det(p_2 - p_k, p_3 - p_k, p_4 - p_k)$ is negative. When an edge is swapped, the edges opposite p_k in the two newly formed triangles become candidates for swapping.

3.4. Estimation of gradient vectors. We assume a triangular grid has been constructed in S covering the spherical convex hull of the points $p_1, \ldots, p_n$ and having the points $p_1, \ldots, p_n$ as vertices. We also assume the data values $u_1, \ldots, u_n$ (See §2) are available. It is required to estimate a $3D$ gradient vector g_i at each point p_i. See §2.1 for the characterization of gradient vectors for this problem.

Let p_i be a point at which a gradient vector g_i is to be estimated. Our general idea is to do a least squares quadratic fit to data near the point p_i and then use the gradient vector of this fitted quadratic polynomial as the gradient vector at p_i. We use a six-term quadratic polynomial in two variables forcing interpolation to the value u_i at p_i. Thus we need at least five neighboring points, and prefer more than five to obtain a local smoothing effect on the gradient vector. Let Q denote the set of points to be used for the fit. We first place all the immediate neighbors of p_i into Q. If the number of immediate neighbors is from six through sixteen and if the matrix for the least squares problem passes a conditioning test then this set Q is used for the fit. If the number of points exceeds sixteen, excess points are discarded. If the number is less than six, more nearby points beyond the immediate neighbors of p_i are introduced. If the matrix condition test is not passed, more points, up to sixteen, are added. If the condition test still fails with sixteen points, the least squares system is damped to bias the solution toward small values of the coefficients of the three second order polynomial terms.

The fitting is set up in a local coordinate system determined by p_i. A 3×3 rotation matrix R is determined that transforms the position vector of p_i to the vector $(0, 0, 1)$. Thus the "north pole" of the rotated coordinate system is at p_i. The same coordinate transformation is applied to all vectors in the fitting set Q. Generally these transformed vectors, having some proximity to p_i, will all lie in the "northern hemisphere" of the rotated coordinate system, i.e., their z coordinates will be positive. If any transformed vector (x, y, z) has $z < 0$, we arbitrarily replace it by $(x/s, y/s, 0)$ where $s = \text{sqrt}(x^2 + y^2)$. This last step is just an expedient to do something definite in a poor situation. Data must be very sparse or poorly distributed to result in any points of Q being in the "southern hemisphere" of the rotated coordinate system.

We ignore the z coordinates of these transformed vectors, using only their x and y coordinates in the fitting. This can be interpreted as projecting the points p_j of Q orthogonally onto the plane T that is tangent

to the sphere at the "north pole", i.e., at p_i. The polynomial model for the fit is

$$c_1x + c_2y + c_3x^2 + c_4xy + c_5y^2 = u - u_i.$$

The coefficients $c_1, \ldots, c_5$ of this polynomial are determined by a least squares computation. The two-vector (c_1, c_2) is the gradient vector at p_i of the fitted polynomial relative to the (x, y)-coordinate system in the tangent plane T. Using the observations at the end of §2.1 we take the three-vector $(c_1, c_2, 0)$ to be the gradient vector at p_i of the (as yet unknown) interpolating function defined over the surface of the sphere. The inverse of the rotation matrix R is then applied to $(c_1, c_2, 0)$ to obtain the representation of the gradient vector g_i in the original coordinate system.

3.5. Interpolation in a single triangle. In the planar case described in [6] we preferred the nine-parameter Clough-Tocher cubic macroelement [3] as our interpolation method primarily for the following two reasons.

(a) It is more economical to evaluate than any other C^1 interpolation method of which we are aware. Beginning with the rectangular coordinates of q and of the vertices, and the function values and $2D$ gradient vectors at the vertices, our evaluation of this interpolant uses 55 multiplications, 65 additions, and 4 divisions.

(b) The interpolant at any point is simply a cubic polynomial in the Cartesian coordinates (or in the barycentric coordinates), and thus it is easy to derive and implement an evaluation of the gradient of the interpolated surface if this should be desired.

Unfortunately, the Clough-Tocher method depends strongly on properties of polynomials in Cartesian coordinates over a planar region and does not seem to generalize for use over a spherical triangle.

We will describe two methods for C^1 interpolation over planar triangles that do generalize to spherical triangles. We first describe aspects that are common to both of these methods. The given data consists of a function value and a $2D$ gradient vector at each vertex. Values along an edge are computed by Hermite cubic interpolation and the tangential derivative at any point on an edge is computed as the derivative of this Hermite cubic interpolation polynomial. The normal derivative at any point on an edge is computed by linear interpolation using the derivatives normal to the same edge at the two ends of the edge. For q on the boundary t^* of a triangle t let $F(q)$ denote the value and $G(q)$ denote the gradient vector defined by these interpolation methods along the boundary.

The interpolant for interior points $q \in t$ will be constructed as a convex combination of partial interpolants:

$$f(q) = \sum_{i=1}^{3} w_i(q) f_i(q), \qquad \sum_{i=1}^{3} w_i(q) = 1. \tag{3}$$

Each partial interpolant f_i will agree with F for all q on the boundary t^*, and will satisfy the gradient conditions $\nabla f_i(q) = G(q)$ for some, but not all, points of t^*. The weight functions w_i will be constructed to be nonnegative throughout t, and to be zero at all boundary points q at which $\nabla f_i(q) \neq G(q)$. Interpolation methods of the form of (3) are analyzed in detail in [4]. It is shown in Corollary 2.5 of [4] that if the f_i's and w_i's are C^1 throughout t and satisfy the conditions just enumerated, the function f of (3) is also C^1 throughout t and satisfies $f(q) = F(q)$ and $\nabla f(q) = G(q)$ for all $q \in t^*$. We sketch here the proof of this proposition as given in [4].

Satisfaction of $f(q) = F(q)$ is clear since each f_i satisfies this condition and the f_i's are combined using weight functions that sum to one. To verify the satisfaction of the gradient condition on the boundary, expand ∇f as

$$\nabla f = \sum_{i=1}^{3} w_i \nabla f_i + \sum_{i=1}^{3} f_i \nabla w_i. \tag{4}$$

Since all f_i's agree with F on the boundary, the second summation in (4) satisfies

$$\sum_{i=1}^{3} f_i \nabla w_i = F \sum_{i=1}^{3} \nabla w_i = F \nabla \sum_{i=1}^{3} w_i = 0,$$

for $q \in t^*$, where the last equation uses the observation that the condition $\sum w_i = 1$ of (3) implies $\nabla \sum w_i = 0$ throughout t. The first summation of (4) evaluates to $G(q)$ for $q \in t^*$ since the w_i's sum to one, and each w_i is nonzero at a boundary point q only if $\nabla f_i(q) = G(q)$.

3.5.1. Planar Method 1. For any point q in the triangle t let f_i in (3) be defined by Hermite cubic interpolation along the line through q parallel to the edge opposite vertex p_i. This function f_i has been called the BBG interpolant or BBG projector due to its use in [1]. See also [2, pp. 92–101]. Function and derivative values for this interpolation are derived from the edge functions F and G defined above. The function $f_i(q)$ defined in this way is C^1 over triangle t, and f_i and its gradient match F and G respectively on all edges, except that the normal derivative of f_i on the relative interior of the edge opposite p_i will generally not be consistent with G.

Thus letting f_i be the BBG interpolant, it will suffice, according to Corollary 2.5 of [4], to require that w_i have the value zero on the edge opposite p_i and be nonzero elsewhere on the boundary of t. This is conveniently assured by letting w_i be the barycentric coordinate of q that has the value zero on the edge opposite p_i and one at p_i. Thus (3) specializes to

$$f(q) = b_1 f_1(q) + b_2 f_2(q) + b_3 f_3(q), \tag{5}$$

where the b_i are the barycentric coordinates of q relative to the triangle t and the f_i's are BBG interpolants, each requiring two linear interpolations and three Hermite cubic interpolations for its evaluation.

3.5.2. Planar Method 2. For any point q in the triangle t let f_i in (3) be defined by Hermite cubic interpolation along a line from vertex p_i through q to the opposite edge. This interpolant has been called a side-vertex or radial interpolant. A number of ways of using side-vertex interpolants to construct interpolation methods are discussed in [8], however the method we are using is not treated there. (After reading the initial version of this paper, Nielson has shown in a personal communication that Planar Method 2 can be derived as a specialization of Theorem 3.3 of [8]). Each partial interpolant f_i is C^1 throughout t and matches the required function value F at all boundary points. The gradient ∇f matches G on the edge opposite p_i, but the normal derivative will generally not be consistent with G on the relative interiors of the two edges adjacent to vertex p_i. with Corollary 2.5 in mind, it seems appropriate for the weight function w_i to contain the product $b_{i+1}b_{i+2}$ as a factor, since this product is zero along the two edges adjacent to p_i, and nonzero in the relative interior of the edge opposite p_i. Here the b_i's denote barycentric coordinates of q, and the subscripts are to be evaluated modulo 3 to one of the values 1, 2, or 3.

Introducing a normalizing factor to assure $\sum w_i = 1$, we define

$$w_i = \begin{cases} b_{i+1}\, b_{i+2}/(b_{i+1}\, b_{i+2} + b_{i+2}\, b_i + b_i\, b_{i+1}) & \text{for } q \neq p_{i+1} \text{ or } p_{i+2} \\ 0 & \text{for } q = p_{i+1} \\ 1 & \text{for } q = p_{i+2}. \end{cases} \tag{6}$$

Note that the rational expression in the first line of the above definition has non-removable singularities at vertices p_{i+1} and p_{i+2}. The assignment of the values 0 and 1 at p_{i+1} and p_{i+2}, respectively, is a technicality to preserve the property $\sum w_i = 1$ throughout t. Using these weight functions (3) specializes to

$$f(q) = \begin{cases} \dfrac{(b_2 b_3 f_1 + b_3 b_1 f_2 + b_1 b_2 f_3)}{(b_2 b_3 + b_3 b_1 + b_1 b_2)} & \text{for } q \neq p_1, p_2, \text{ or } p_3 \\ u_i & \text{for } q = p_i. \end{cases} \tag{7}$$

Here each partial interpolant f_i requires one linear interpolation and two Hermite cubic interpolations along lines. Corollary 2.5 of [4] establishes the C^1 continuity of f, except possibly at the three vertices, since the weight functions of (6) do not satisfy the hypotheses of the corollary at these

three points. Each w_j is discontinuous at two vertices and lacks differentiability at all three vertices. Thus it is necessary to prove that f is continuous, differentiable, and continuously differentiable at the vertices.

This proof has been recorded in detail in an internal memorandum [7]. It is shown that each of the side-vertex partial interpolants f_j has a representation in a neighborhood of each vertex p_i of the form

$$f_j(p_i + h) = u_i + g_i^T h + 0\ \|h\|^2$$

for $2D$ vectors h that are sufficiently small and such that $p_i + h \in t$. The proofs of continuity and differentiability of f then follow directly. To demonstrate that ∇f is continuous at a vertex p_i we write, for $h \neq 0$,

$$\begin{aligned}\nabla f(p_i+h)-g_i &= \textstyle\sum_j f_j(p_i+h)\ \nabla w_j(p_i+h)+\sum_j w_j(p_i+h)\ \nabla f_j(p_i+h)-g_i\\ &= (u_i+g_i^T h)\ \textstyle\sum_j \nabla w_j(p_i+h)+\sum_j[\nabla w_j(p_i+h)\ 0\ \|h\|^2]\\ &\quad+\textstyle\sum_j\,[w_j(p_i+h)\ (g_i+0\ \|h\|)]-g_i\\ &= \textstyle\sum_j[\nabla w_j(p_i+h)\ 0\ \|h\|^2]+\sum_j[w_j(p_i+h)\ 0\ \|h\|]\\ &= \textstyle\sum_j \nabla w_j(p_i+h)\ 0\ \|h\|^2\ +\ 0\ \|h\|.\end{aligned}$$

It is then shown that the last expression above approaches zero as $\|h\| \to 0$ by verifying that each $\nabla w_j(p_i + h)$ grows no faster than $\|h\|^{-1}$ as $\|h\| \to 0$. In particular it is shown that

$$\nabla w_j(p_i + h) = \begin{cases} O(1) & \text{when } j = i \\ O(\|h\|^{-1}) & \text{when } j \neq i. \end{cases}$$

3.5.3. Generalization of Planar Methods 1 and 2 for spherical triangles. The key in generalizing these two planar methods for use with a grid of spherical triangles on the surface S of the unit sphere is to replace all of the linear and Hermite cubic interpolations along line segments by the same type of interpolations along arcs of great circles in S. Let t denote a proper spherical triangle with vertex position vectors p_1, p_2, and p_3, and let q be a point of S contained in t. Let t' denote the underlying planar triangle having the same vertices as t, and let q' be the central projection of q into the plane of triangle t', i.e., q' is the point in the plane of t' intersected by the line from the center of the sphere to q.

When the look-up procedure of §3.2 finds that a given point q in S is in triangle t, it also returns the three nonnegative numbers s_1, s_2, and s_3 of (2). We call these numbers unnormalized barycentric coordinates since the (normalized) barycentric coordinates of q' relative to the planar triangle t' can be computed as

$$b_i = s_i/(s_1 + s_2 + s_3), \qquad i = 1, 2, 3.$$

The intersection points between certain lines through q' and edges of t'

needed for either of the two planar interpolation methods are easily represented in terms of the b_i's and p_i's. Thus the intersection between edge p_ip_{i+1} with the line through q' parallel to edge $p_{i+1}p_{i+2}$ has position vector $b_ip_i + (1 - b_i)p_{i+1}$ while the intersection between edge $p_{i+1}p_{i+2}$ with the line from vertex p_i through q has the position vector $(b_{i+1}p_{i+1} + b_{i+2}p_{i+2})/(b_{i+1} + b_{i+2})$. These intersection points can then be centrally projected to S by normalizing their position vectors to have unit Euclidean length. All of the linear and cubic interpolations called for in the planar methods are then done with respect to arc length along great circle arcs in S obtained by central projection of the corresponding line segments in the planar triangle t'.

Recall that gradient data at each vertex p_1, p_2, and p_3 is represented as a three-vector orthogonal to the position vector of the vertex. Gradient information generated at auxiliary points in either interpolation method is also represented as a three-vector orthogonal to the associated position vector. The verification that each of these two spherical triangle interpolation methods defines a C^1 function over S can be carried out in the same way the C^1 property of the planar methods is proved.

4. Software implementing these algorithms. Subroutines were written for these algorithms in 1979 using the JPL SFTRAN3 structured Fortran language which is preprocessed to Federal (ANSI) Standard Fortran 77. The time for grid construction for n points was proportional to $n^{1.25}$ for test cases in the range from 25 to 500 points. The RMS error in test cases using simple mathematical functions to generate data over relatively uniform triangular grids of various densities was proportional to $h^{3.4}$ in test cases having maximum edge length in the grid ranging from 63° down to 9°. A count of the number of arithmetic operations required to do a single interpolation in a triangle gives the figures listed in Table 1. The planar Clough-Tocher method is included for comparison. For all methods the computation starts with Cartesian coordinatdes for q, p_1, p_2, and p_3 and function values and gradient vectors at p_1, p_2, and p_3. The weights used to combine the counts are arbitrary but plausible. They are normalized to cause an addition plus a multiplication to sum to one for consistency with operation counts measured in "Flops".

As a test of the C^1 continuity of Method 1 we reprogrammed the code for that method using a "U-arithmetic" package developed at JPL in 1971 based on the ideas of [**10**]. (This is like the method of [**5**] without the benefit of a preprocessor.) In this approach the program computes a $3D$ gradient vector and a 3×3 Hessian matrix for every intermediate quantity, and thus also for the final interpolated value. All derivative computations use mathematically correct formulas, i.e., not differencing. We found it necessary to reorder some computations to avoid severe

	Clough-Tocher Planar Method	Spherical Method 1	Spherical Method 2	Factors for weighted total
Add/Subtract	65	371	352	0.4
Multiply	55	699	450	0.6
Divide	4	81	57	1.2
Sqrt		24	15	3.0
Atan		18	12	5.0
Weighted Total (Flops)	63.8	827.0	584.2	

TABLE 1. Operation counts for a single interpolation in a triangle

artificial numerical instabilities in the derivative computations. After this reordering the results were consistent with C^1 continuity.

We did not try a U-arithmetic version of Method 2. I would expect severe difficulties with this since the singularities of the w_i's at certain vertices (See §3.5.2) imply that some first partial derivatives of the w_i's can be arbitrarily large in a small neighborhood of a vertex. Mathematically these cancel out, but numerically there would be large rounding errors.

Numerical differencing tests for C^1 continuity were run on the implementations of both Method 1 and Method 2. The set of ten points listed in Table 2 was used as the grid nodal points for the tests. The grid generation subroutine was run producing sixteen triangles as shown in

(East) Longitude (Degrees)	Latitude (Degrees)
0	0
40	10
5	35
−35	20
−25	−30
20	−25
240	10
180	40
155	−10
180	−20

TABLE 2. Nodal points for test cases.

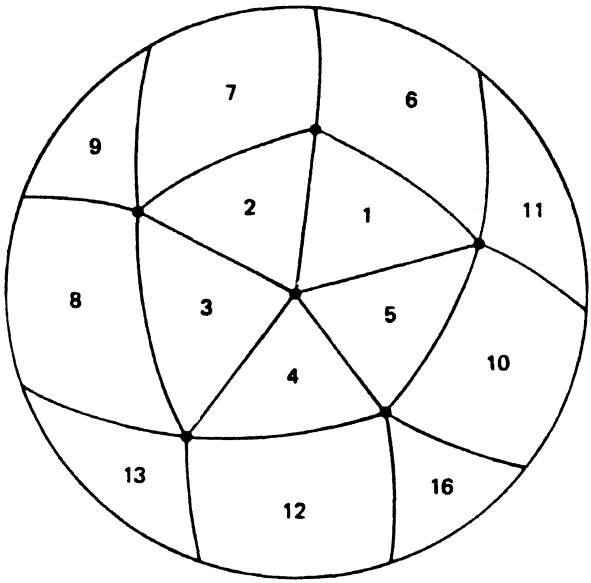

FIGURE 1. Front surface of sphere, centered on the zero longitude meridian.

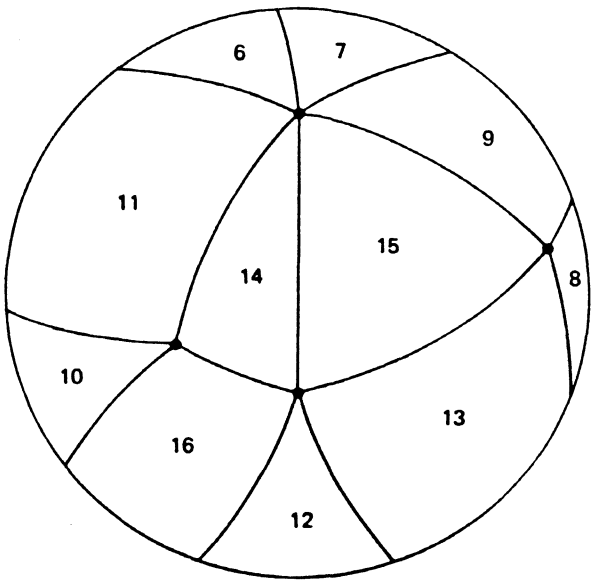

FIGURE 2. Back surface of sphere, centered on the 180° longitude meridian.

Figures 1 and 2. The numbers in these two figures are identification numbers for the triangles.

The first six points of Table 2 appear as triangle vertices in Figure 1 while the last four points are vertices in Figure 2. The vertex points are positioned correctly as orthogonally projected onto the (y, z)—plane. The triangle edges were drawn with convenient drafting tools and do not portray precise curvatures.

Longitude and latitude are related to rectangular coordinates by the equations

$$x = \cos(\text{longitude})\ \cos(\text{latitude})$$
$$y = \sin(\text{longitude})\ \cos(\text{latitude})$$
$$z = \sin(\text{latitude})$$

In Case 1 of the difference testing we used interpolation at thirty points as illustrated in Figure 3. This figure corresponds to a small central region of Figure 1.

There is a primary set of fifteen interpolation points on the equator running from -0.07 radians to 0.07 radians of longtiude in steps of 0.01 radians. This set of points falls in Triangles 3 and 5 and includes the vertex common to these two triangles. These points are used to compute first and second differences in the longitude direction. The secondary set of fifteen interpolation points lying 0.01 radians of latitude above the primary set is used for computing a first difference in the latitude direction. Note that some of these points fall in Triangles 2 and 1 as well as 3 and 5.

For Case 2 we used a similar set of thirty points shifted 180° in longi-

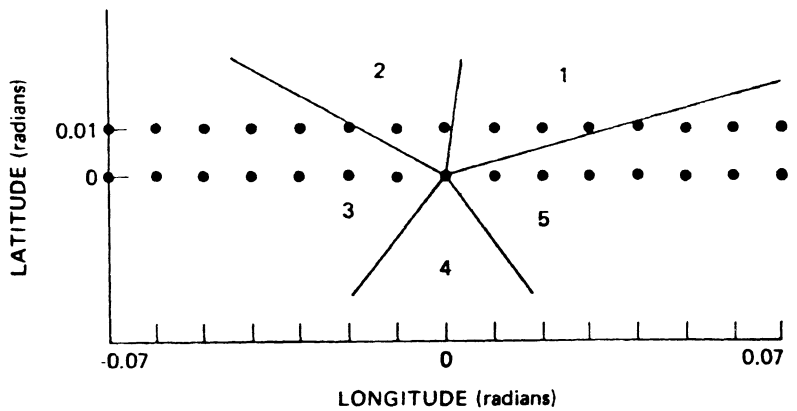

FIGURE 3. Interpolation points for Case 1.

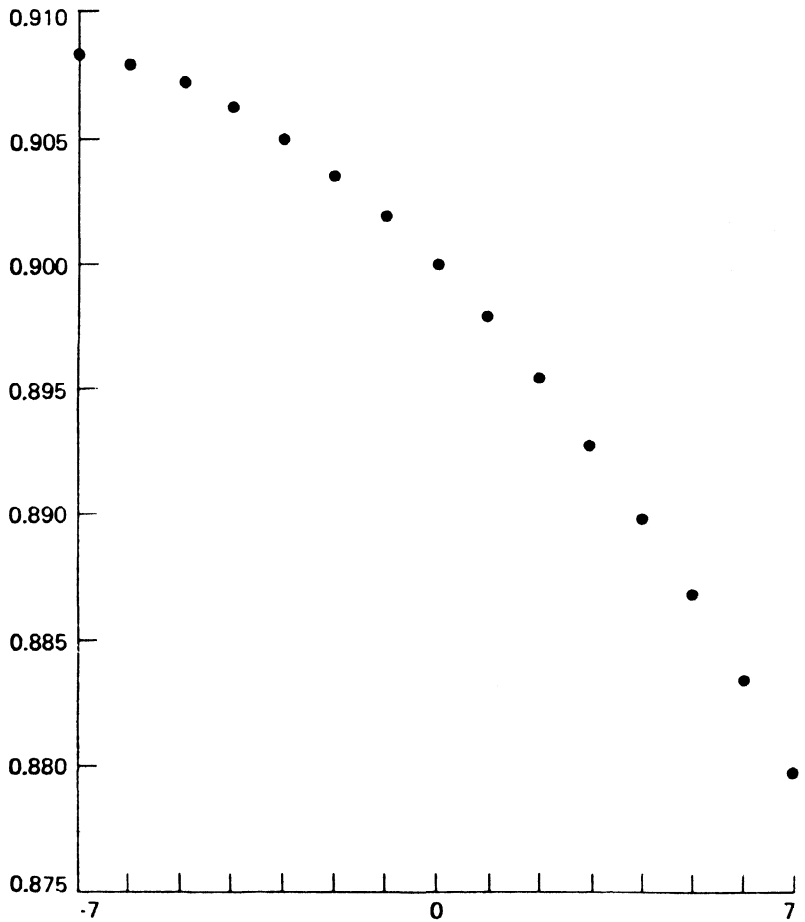

FIGURE 4a. Case 1. Interpolated value.

tude. Thus for Case 2 the interpolation points lie in Triangles 14 and 15 with the central primary and secondary points lying on the edge common to these two triangles.

For the test reported here the function to be interpolated was the cubic polynomial

$$f(x, y, z) = (9x^3 - 2x^2y + 3xy^2 - 4y^3 + 2z^3 - xyz)/10.$$

Values of this polynomial and its gradient vector, projected into the local tangent plane, were associated with each point of Table 2. For Method 1 results from these tests are shown in Figures. 4*a-f* and 5*a-f*. The abscissa

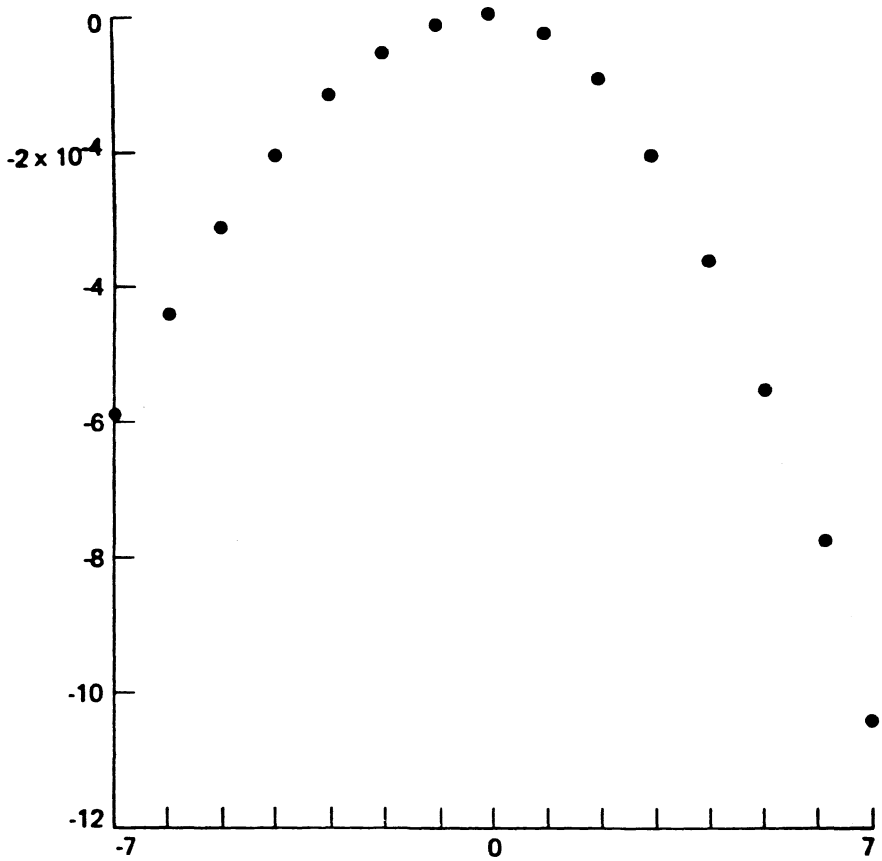

FIGURE 4b. Case 1. Interpolated value minus true value.

in these figures and in Figures. 6–7 is in units of 0.01 radians from the central longitude for the case. The central longitude is zero for Case 1 and 180° for Case 2. These tests were run on a Univac 1100/81 computer having relative precision of 27 bits, or about 8.1 decimal places.

These results appear to be consistent with the expectation that the interpolation method maintains C^1 continuity when crossing a vertex (Case 1) or an edge (Case 2), and interpolates the value and gradient data at the vertex common to triangles 3 and 5 in Case 1. Thus in Figures. 4c, 4e, 5c, and 5e, the first differences may have different behavior to the left and right of the center abscissa (0 in Case 1 and 180° in Case 2) but the left and right subsets appear to approach a common value at the

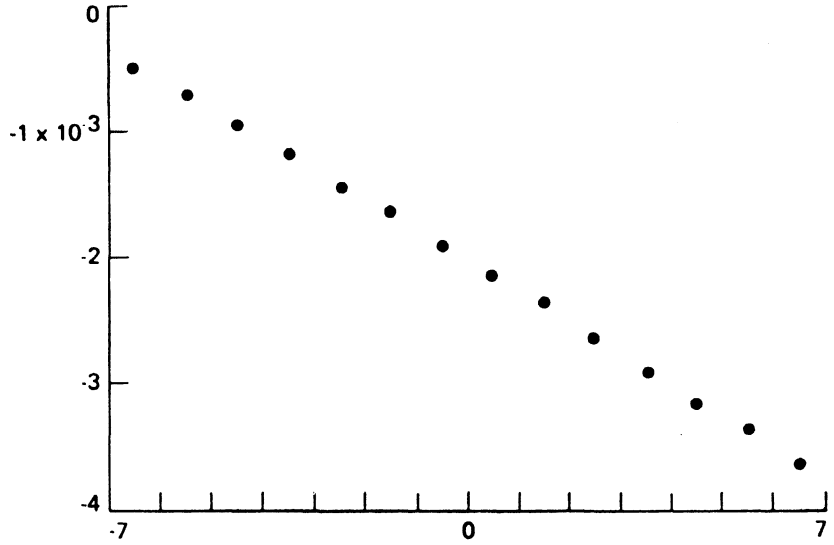

FIGURE 4c. Case 1. First difference in longitude.

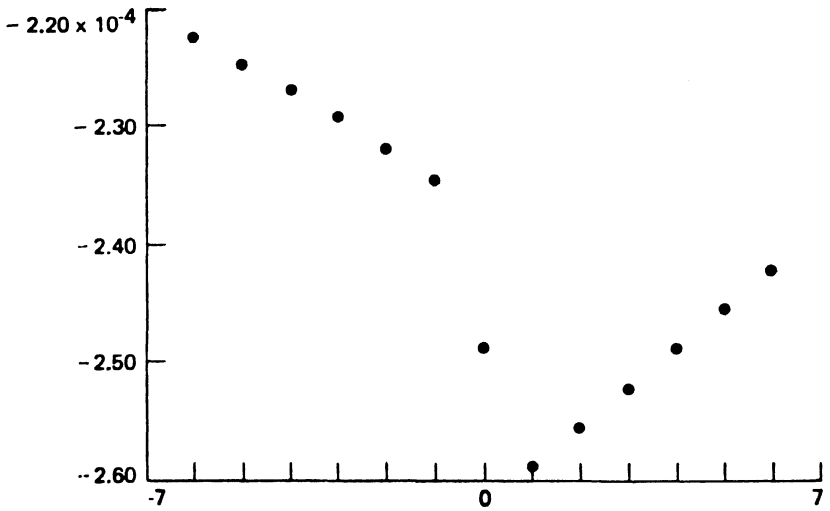

FIGURE 4d. Case 1. Second difference in longitude.

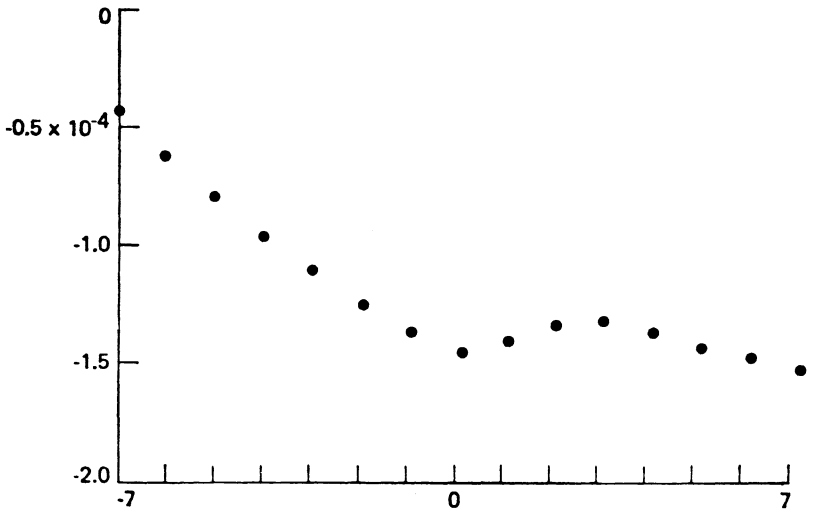

FIGURE 4e. Case. 1. First difference in latitude.

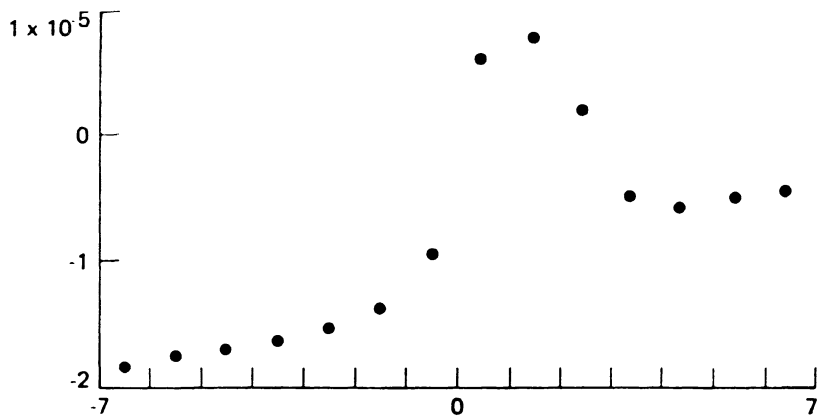

FIGURE 4f. Case 1. First difference in longitude of first difference in latitude.

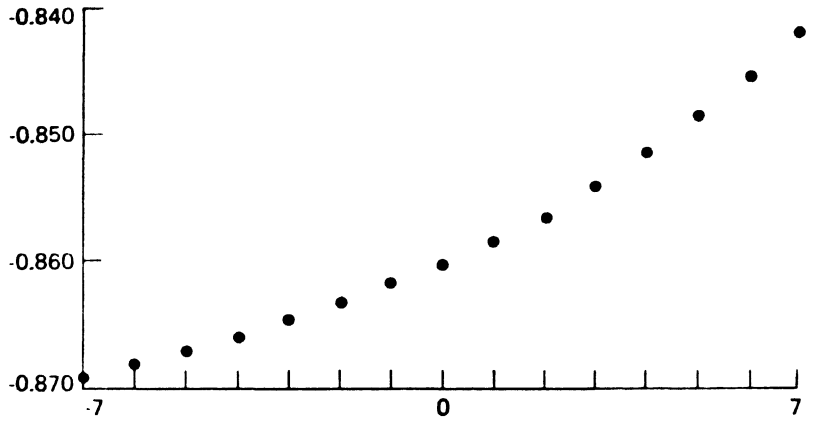

FIGURE 5a. Case 2. Interpolated values.

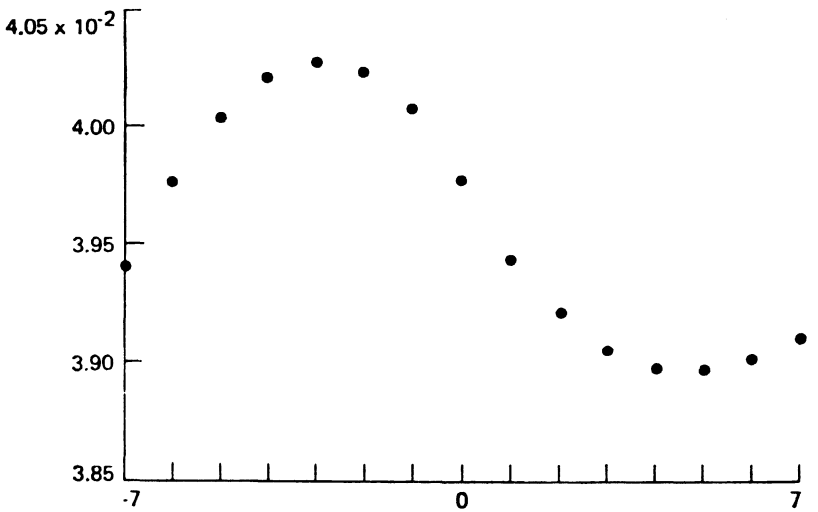

FIGURE 5b. Case 2. Interpolated value minus true value.

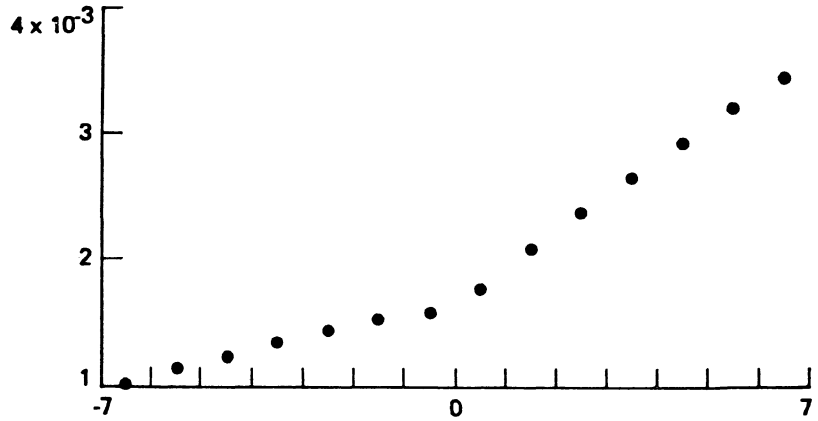

FIGURE 5c. Case 2. First difference in longitude.

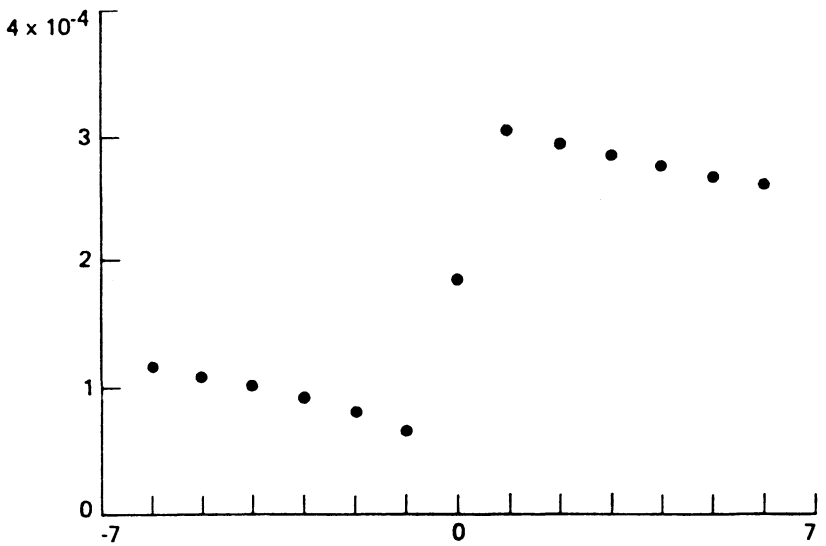

FIGURE 5d. Case 2. Second difference in longitude.

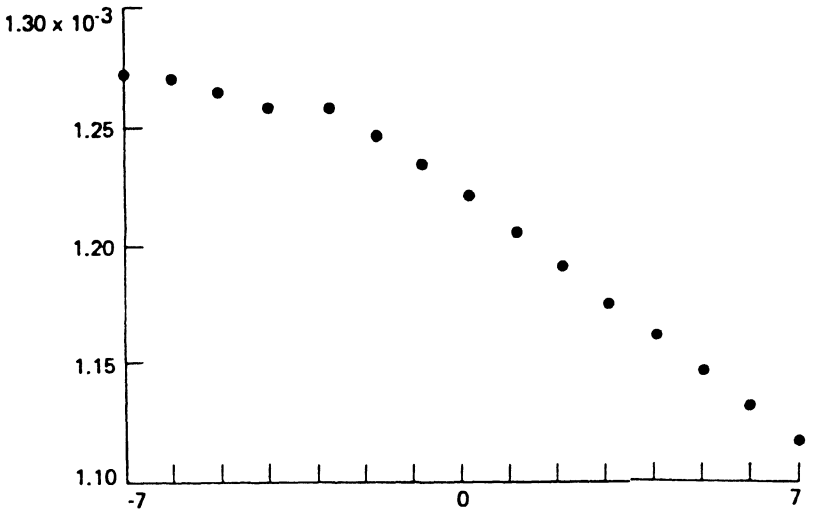

FIGURE 5e. Case 2. First difference in latitude.

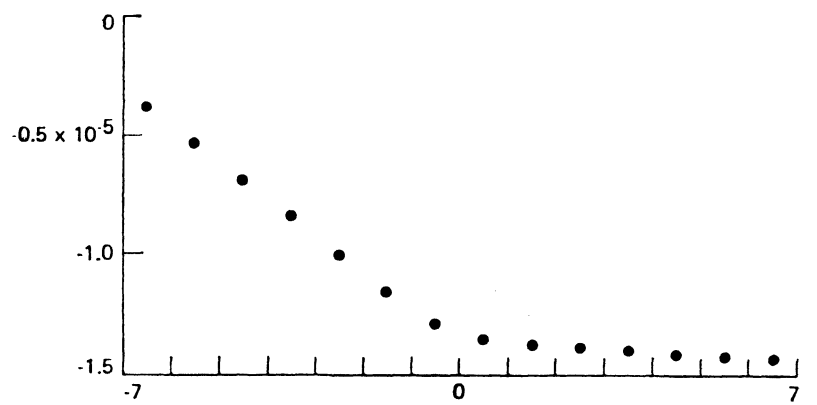

FIGURE 5f. Case 2. First difference in longitude of first differences in latitudes.

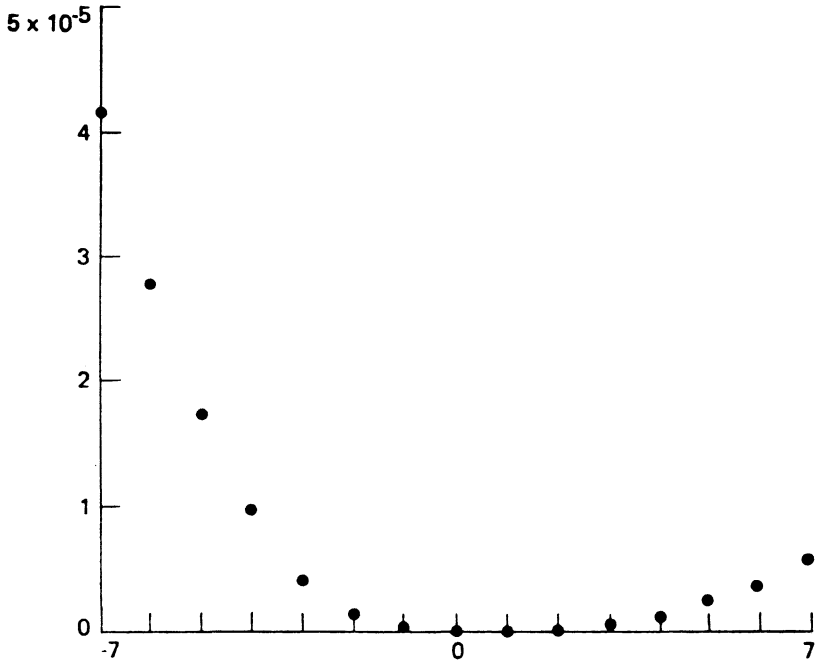

FIGURE 6. Case 1. Method 1 minus Method 2.

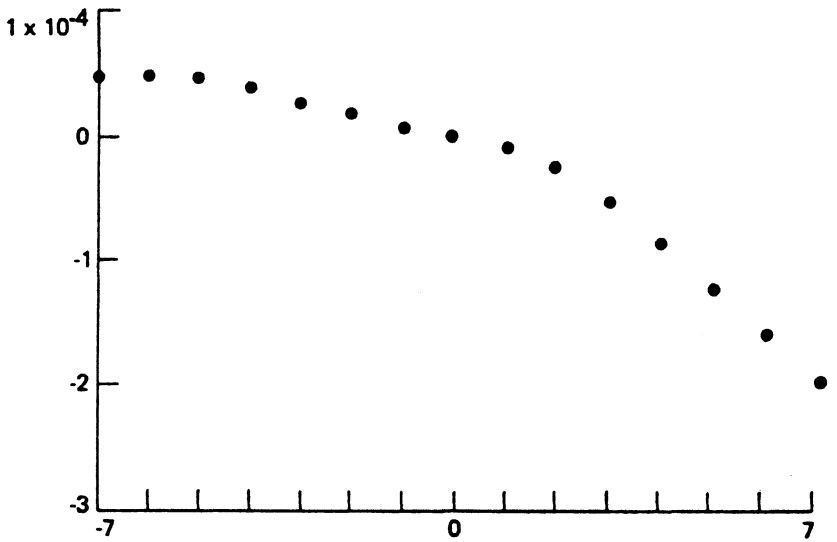

FIGURE 7. Case 2. Method 1 minus Method 2.

center abscissa. Graphs of results using Method 2 are almost identical in general appearance to those for Method 1 and thus are not included here. Differences between interpolated values obtained by Method 1 and Method 2 are shown in Figures. 6 and 7.

5. An application. In February, 1982, this software was used at JPL in the study of gravity variation over the surface of the planet Venus. Data was available at many, but not all points, of a rectangular longitude-latitude grid. The missing data occurred in irregularly shaped regions determined by geometrical constraints of the observation and communication instruments. Using 2450 points at which data was present our program built a spherical triangular grid consisting of 4896 triangles. Missing data in the rectangular grid was then filled in by interpolation in the triangular grid. In the course of this work the scientists gained new insights regarding their data and we found and repaired a weak spot in our program. See the discussion of determinant evaluation in §3.2.

6. Conclusions and remarks. The efficiency of the grid building procedure, execution time in test cases being observed to be proportional to $n^{1.25}$, is quite satisfactory. C^1 interpolation in a spherical triangle requires nine to thirteen times as many Flops as C^1 interpolation in a planar triangle. Modifications giving small reductions in the operation counts are known, but it would be interesting if an entirely different approach could be found that might be more intrinsically related to the topology of the spherical surface and require significantly fewer Flops.

Method 1 is more time-consuming than Method 2 by a factor of about 3 to 2 since Method 1 uses nine cubic interpolations along arcs compared with six for Method 2. Analytic computation of gradients for interpolated values would probably be more stable using Method 1 than Method 2 because of the singularities in the w_i's of Method 2. It would be interesting to make visual comparisons of surfaces generated by these two methods, but we have not had the resources to make such comparisons.

The programs appear to be robust and reliable. The use of the SFTRAN3 structured Fortran language has been extremely helpful in keeping the code understandable.

It should be noted that the use of the surface of a sphere as the domain is just a mathematical construct for dealing with the set of all directions in three-space from an origin point. Thus the methods of this paper are applicable to the representation of any bounded two-dimensional C^1 surface in three-space that is "starlike" in the sense that there is some origin point from which a ray in any direction intersects the surface in at most one point and the ray is not tangent to the surface at that point. Other two-dimensional manifolds besides the plane and the spherical surface that may deserve investigation for scattered data interpolation

include the surface of a cylinder or a torus. On a cylinder one may wish to admit triangles having two vertices at the same data point while on the torus one may admit triangles having all three vertices at the same data point.

References

1. R. E. Barnhill, G. Birkhoff and W. J. Gordon, *Smooth Interpolation in Triangles*, J. Approx. Theory **8** (1973), 114–128.

2. R. E. Barnhill, *Representation and Approximation of Surfaces*, Mathematical Software III, ed. J. R. Rice, Academic Press, 1977, 69–120.

3. R. W. Clough and J. L. Tocher, *Finite element stiffness matrices for analysis of plates in bending*, Proc. Conf. Matrix Methods in Struct. Mech., Air Force Inst. of Tech., Wright-Patterson A.F.B., Ohio, 1965.

4. G. J. Herron, *Triangular and Multisided Patch Schemes*, Ph.D. Theisis, University of Utah, 1979.

5. G. Kedem, *Automatic Differentiation of Computer Programs*, TOMS **6** (1980), 150–165.

6. C. L. Lawson, *Software for C^1 Surface Interpolation*, Mathematical Software III, ed. J. R. Rice, Academic Press, 1977, 161–194.

7. ———, *The C^1 continuity of a side-vertex interpolator*, JPL Internal Memorandum, Computing Memorandum No. 366–491, 1982.

8. G. M. Nielson, *The Side-Vertex Method for Interpolation in Triangles*, Jour. Approx. Theory **25** (1979), 318–336.

9. L. L. Schumaker, *Fitting Surfaces to Scattered Data*, Approximation Theory II, ed. G. G. Lorentz, C. K. Chui, and L. L. Schumaker, Academic Press, 1976, 203–268.

10. R. E. Wengert, *A Simple Automatic Derivative Evaluation Program*, CACM **1** (1964), 463–464.

Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109