ROBUST CONTOUR TRACING

David P. Dobkin
Silvio V. F. Levy
William P. Thurston
Allan R. Wilks

CS-TR-054-86

Revised November 1987

Robust Contour Tracing

DAVID P. DOBKIN
SILVIO V. F. LEVY
WILLIAM P. THURSTON[1]
Princeton University

ALLAN R. WILKS
AT&T Bell Laboratories
Murray Hill, New Jersey 07974

November 17, 1987

Abstract. We present a robust method for tracing a curve that is represented as the contour of a function in Euclidean space of any dimension. The method proceeds locally by following the intersections of the contour with the facets of a triangulation of space.

The algorithm is robust in the presence of high curvature of the contour, and gives reasonable results when the curve is self-intersecting. It accumulates essentially no round-off error, and has a well-defined integer test for detecting a loop. In developing the algorithm we explore the nature of a particular class of triangulations of Euclidean space, namely, those generated by reflections.

Keywords: contour tracing, simplicial continuation, Coxeter triangulations.

1. Problem

We consider the problem of tracing the contours of a map between Euclidean spaces. If $f : \mathbf{R}^n \to \mathbf{R}^k$ $(k < n)$ is, say, smooth, the implicit function theorem (together with Sard's theorem) says that, for most points y in the image of f, the *contour* $C = f^{-1}(y)$ is a smooth manifold of dimension $n - k$. If $k = n$, again for most points y in the image of f, C is a collection of isolated points, and finding C is the same as finding the *roots* of $y = f(x)$; although the solution to this problem has a vast literature, it seems to be of little help in solving the problem for $k < n$.

The problem of finding the contour C breaks into two parts: locating the various components of C (that is, finding at least one point in each component), then tracing around each component, using the fact that the components are connected. Later we state more precisely our definition of what it means to trace C; the general idea is that we want to compute a collection of "representative" points in C. In this paper we will focus on the tracing problem, although we discuss very briefly (section 8) the problem of locating contours. In any case, many applications supply their own starting points.

[1]Research of the first three authors was supported in part by the National Science Foundation under Grant DCR85-05517.

We will assume that we are given a ***program*** to compute $f(x)$, for any x in some region of interest. This is to be distinguished from another situation that often arises, in which we are given a collection of pairs $\big(x_i, f(x_i)\big)$, for $i = 1, \ldots, N$, with the points x_i scattered throughout the region of interest. In principle the two situations are nearly equivalent: given a program to compute f, we can produce the values of f at $x_1, \ldots, x_N$; or given those values we can design a program that computes an approximation to f. In practice, however, if we have a program that calculates f for any input we will not precalculate f at a set of values, because many of these values will not be used in tracing the desired contour, especially in higher dimensions. Thus the two situations tend to lead, in practice, to different sorts of algorithms for tracing contours.

Most of the large literature on contour tracing deals with the data case. For an excellent review with extensive bibliography, see [Sabin 1985]. Arnon [1983] discusses the case of curves given by functions f on the plane, when f is algebraic.

Our decision to focus on functions rather than data does not, in general, create too great a restriction. A prime example is in estimating the probability density of some n-dimensional distribution from which we have a number of independent and identically distributed observations. Most existing procedures to contour a density estimate proceed by evaluating the estimate on a grid, then building the contours by a complete pass across the grid. But viewing the density estimate as a function (which is often, but not always, practical, depending upon the particular estimate), we may apply the methods of this paper to find the contours directly, while only evaluating the estimate at points in a neighborhood of the contour.

We will focus our attention on the case $k = n - 1$, which leads to one-dimensional contours, but our results can be extended to higher-dimensional contours; we take up this subject briefly in section 8.

The algorithm we will describe is related to a class of algorithms first introduced by Scarf [1967] to deal with certain fixed point problems in economics. These algorithms, based on a lemma due to Sperner [1928], basically follow a contour curve (typically of a homotopy) as it passes close to, but not necessarily through, a sequence of simplices. For more details, see the books by Todd [1976] and Karamardian [1977], the survey paper by Allgower and Georg [1980] with its extensive bibliography, and the recent article by Allgower and Schmidt [1985].

Our work is a departure from this line in several ways. First, we trace contours through the precise set of simplices they intersect; this translates into a gain in accuracy, especially as the dimension increases. Second, we make use of well-known geometrical results [Coxeter 1973] to generalize the several specific triangulations that have been in common use in the fixed point literature; in particular, we explore and compare all monohedral triangulations generated by reflections, in arbitrary dimension. Finally, the algorithm described here seems conceptually simpler, and is correspondingly easier to implement. A fairly general implementation of it is the subject of a paper currently under preparation.

Many interesting problems can be stated in the context of contour tracing. A few of them, drawn from computer graphics, computational geometry, pure mathematics and statistics, are discussed in the next section. These examples are not meant to be exhaustive, but rather to illustrate some directions in which the general method can be applied. Still in section 2, we outline two possible approaches to the tracing problem, and indicate why we chose one over the other.

Section 3 is devoted to an easily understood instance of the algorithm, and to setting the stage for the more general case. In section 4 we generalize the triangulation, and in section 5 we present further computational details, including a theoretical analysis of the relative performance of the various triangulations. Section 6 contains the full algorithm, adapted to degenerate cases. Finally, in section 7 we give examples of the algorithm in action, while in section 8 we suggest several possible extensions of the method.

2. Examples and Approaches

We begin by describing a broad range of sample problems to which contour tracing has been applied.

Example 1: curve tracing

Many simple curves may be represented as the contour of some suitable function. For example, let $f : \mathbf{R}^2 \to \mathbf{R}^1$ be the function defined by $f(x,y) = x^2 + y^2$. The graph of f is a paraboloid and contours of f, for y positive, are circles centered at the origin. The contour of f containing the starting point $(r,0)$ is just the circle $f(x,y) = r^2$.

Example 2: shadow computation

Let Σ_1 and Σ_2 be spheres with centers $\mathbf{c}_1 = (x_1^0, y_1^0, z_1^0)$ and $\mathbf{c}_2 = (x_2^0, y_2^0, z_2^0)$, and radii r_1 and r_2. Assume that a light source is located at (lx, ly, lz) and that it is known that Σ_1 casts a shadow on Σ_2. We can study the outline of this shadow by considering a contour of the function $f : \mathbf{R}^6 \to \mathbf{R}^5$ with components $f = (f_1, f_2, f_3, f_4, f_5)$, where

$$f_i(x_1, y_1, z_1, x_2, y_2, z_2) = (x_i - x_i^0)^2 + (y_i - y_i^0)^2 + (z_i - z_i^0)^2, \quad \text{for } i = 1, 2,$$

is the square of the distance from the point $\mathbf{p}_i = (x_i, y_i, z_i)$ to the center of sphere Σ_i;

$$f_3(\mathbf{p}_1, \mathbf{p}_2) = (x_1 - x_1^0)(x_1 - lx) + (y_1 - y_1^0)(y_1 - ly) + (z_1 - z_1^0)(z_1 - lz)$$

vanishes if and only if $\mathbf{p}_1$ lies on the horizon of the first sphere (the vector from the $\mathbf{p}_1$ to $\mathbf{c}_1$ is perpendicular to the vector from $\mathbf{p}_1$ to the light source); and

$$f_4(\mathbf{p}_1, \mathbf{p}_2) = (x_1 - lx)(y_2 - ly) + (x_2 - lx)(y_1 - ly),$$
$$f_5(\mathbf{p}_1, \mathbf{p}_2) = (x_1 - lx)(y_3 - ly) + (x_3 - lx)(y_1 - ly)$$

vanish if and only if $\mathbf{p}_1$ and $\mathbf{p}_2$ are collinear with the light source. The contour corresponding to $f(\mathbf{p}_1, \mathbf{p}_2) = (r_1^2, r_2^2, 0, 0, 0)$ gives all pairs of points such that the first belongs to the first sphere and the second belongs to the second sphere and lies in the shadow of the first. The shadow outline itself is (a part of) the $\mathbf{p}_2$-component of this contour. Notice that, although we need to keep track of the $\mathbf{p}_1$-component as well, this component is irrevelant to the computed shadow.

Example 3: object intersection

A torus in $\mathbf{R}^3$ is the set of points that are at a fixed distance from a given circle. (The distance from a point to a circle is the smallest distance to any point on the circle.) If we denote by $T(\mathbf{p}; \mathbf{c}, r, \mathbf{n})$ the square of the distance from a point $\mathbf{p}$ to the circle with center $\mathbf{c}$, radius r and lying in the plane normal to $\mathbf{n}$, we have

$$T(\mathbf{p}; \mathbf{c}, r, \mathbf{n}) = \big(r - \|\mathbf{p} - \mathbf{c} - a\mathbf{n}\|\big)^2 - (\mathbf{p} \cdot \mathbf{n})^2.$$

The torus of thickness d about this circle is the contour $T^{-1}(d^2)$. To find the intersection of the torus of thickness d_1 around the circle $(\mathbf{c}_1, r_1, \mathbf{n}_1)$ with the torus of thickness d_2 around the circle $(\mathbf{c}_2, r_2, \mathbf{n}_2)$, define $f : \mathbf{R}^3 \to \mathbf{R}^2$ by

$$f(\mathbf{p}) = \big(T(\mathbf{p}; \mathbf{c}_1, r_1, \mathbf{n}_1), T(\mathbf{p}; \mathbf{c}_2, r_2, \mathbf{n}_2)\big).$$

Then the intersection is the set of curves $f^{-1}(d_1^2, d_2^2)$.

Example 4: density estimation, confidence regions

A common activity in statistical data analysis is to estimate a probability density, given a sample of data points $x_1, \ldots, x_n \in \mathbf{R}^d$. One possibility is the *balloon density estimate*, which is defined at $x \in \mathbf{R}^d$, to be proportional to the reciprocal of the volume of the sphere centered at x that contains exactly k of the data points, for some fixed k [Tukey 1981]. This function from $\mathbf{R}^d$ to $\mathbf{R}$ has $(d-1)$-manifolds as contours; to study these, pictures can be made of level curves, by fixing the values of $d-2$ of the dimensions and tracing the contours of the resulting functions from $\mathbf{R}^2$ to $\mathbf{R}$.

In hypothesis testing, confidence regions can be considered as the regions bounded by contours of the log likelihood function, so their computation can be attacked as a contour tracing problem.

Some of the problems above can be solved by methods other than contour tracing: for example, Bresenham's algorithm can be used to scan-convert circles [Foley and van Dam 1982, p. 442], ray tracing to compute shadows, and there are other approaches to the problem of intersections of surfaces (see, for example, [Geisow 1983] and [Requicha and Voelcker 1983]). However, the paradigm of contour tracing is a useful one in which to cast these diverse problems. In our research, we have used contour tracing in numerous other situations; initially the idea arose in connection with two problems which we had previously considered to be unrelated, namely, the calculation of Julia and Mandelbrot sets [Blanchard 1984] and the fast display of torus knots. The first of these problems is discussed in section 7. For the latter, other considerations are more significant than the contour tracing and the methodology will be described elsewhere.

We now describe in more detail the concept of contour tracing. Let $f : U \to \mathbf{R}^{n-1}$ be a C^1 function on an open, connected subset U of $\mathbf{R}^n$, and assume that we are given x_0 and y_0 such that $y_0 = f(x_0)$. The *contour* of f passing through x_0 is the inverse image $C = f^{-1}(y_0)$, or sometimes the connected component of it containing x_0. Assuming for the moment that y_0 is a *regular value* of f, that is, $df(x)$ is of rank $n-1$ whenever $f(x) = y_0$, the implicit function theorem says that C is a disjoint union of smooth curves in U, each homeomorphic to either the reals $\mathbf{R}$ or to the circle S^1.

Our problem is find the connected component C_0 of C that contains x_0. More specifically, we would like to generate a finite sequence of points in C_0 by starting at x_0, and stepping along the curve until we either come back to x_0 or we reach the boundary of U. In the latter case, we start again at x_0 and trace in the opposite direction until we again hit the boundary. These are the only possibilities.

The first decision to be made is, how big should the steps along the curve be? A natural criterion for the step size is that every point of the curve should be within ε of one of the computed points, for some preassigned $\varepsilon > 0$. In addition, we might like the computed points to be closer together when the curve is changing more rapidly, in order that the details of the changes can be followed. In fact, we might vary the step size at each step, according to some estimate of the local curvature of C_0 at the current position. In what follows, we will not take this dynamic approach; we will assume that points that are at most ε apart on the curve are sufficient. This is certainly a reasonable assumption if C_0 is being graphically presented as, say, the linear spline connecting the computed points, and if ε is taken to be equal to the resolution of the plotting device; in this case the true picture of C_0 would be indistinguishable from the computed picture. We will talk a bit more about variable step sizes in section 8.

The basic problem is then to find the next point $x_{i+1} \in C_0$, given the current point $x_i \in C_0$, and perhaps previous points. Two general approaches can be broadly characterized as "infinitesimal" and "local" (but see [Geisow 1983] for a classification of methods that is somewhat different from ours).

The infinitesimal approach is easiest to describe for $n = 2$ (figure 1(a)). To find x_{i+1}, we use the derivative of f at x_i to determine the tangent line to the curve. We take a small step away from x_i along this line, then do a one-dimensional search in the direction orthogonal to the tangent to "rediscover" the curve. The main shortcomings of this method are: (a) it is possible to lose the curve in a region where it has high curvature; (b) derivatives may not be available for f, so an approximation must be used, and this can be a source of numerical difficulties; (c) each step requires several function evaluations to rediscover the curve—as many as 5 or 6 in dimension two—and in many applications a single function evaluation can be significantly expensive; (d) the procedure is difficult to generalize to higher dimensions satisfactorily. See [Allgower and Georg 1980] for a brief review of some standard infinitesimal methods and how they deal with these problems.

The idea behind the local approach is to use, rather than the derivative of f at x_i, the values of f in a neighborhood of x_i. At the outset, f is conceptually replaced with a function $\tilde{f}$, which is equal to f at points in some grid, and is defined by interpolation at other points. (Of course, the interpolation depends on the grid.) The simplest possible interpolation is affine (linear), and in this case the contour $\tilde{C} = \tilde{f}^{-1}(y_0)$ is piecewise linear, and can be traced exactly. In general, $\tilde{C}$ approximates C to the resolution of the grid, but it fails to reflect sudden variations in f that take place in a scale smaller than this resolution. Furthermore, $\tilde{C}$ and C may be topologically different; for example, if C has two components that come very close at some point, $\tilde{C}$ may join these two components at a node or a "neck".

On the other hand, the local method described in this paper does not suffer from disadvantages (a)–(d) above: it never loses the curve, does not involve derivatives, requires only one function evaluation per step, and generalizes easily to all dimensions. It also gives reasonable results even in cases when y_0 is a critical value of f, whereas the infinitesimal approach requires special care in these cases.

We can summarize the two approaches by saying that the local method traces the exact contour of an approximation to f, whereas the infinitesimal method traces an approximate contour of the exact f. Figure 1 illustrates the two approaches. In figure 1(a) the current position is x_n, on the contour $f^{-1}(y)$, where $f : \mathbf{R}^2 \to \mathbf{R}$ is some smooth function. We take a step from x_i to $x_i + \varepsilon f'(x_i)$, then do a one-dimensional search in the direction perpendicular to the direction of the step (along the dotted line). In Figure 1(b), f has been (conceptually) replaced by the affine approximation $\tilde{f}$, and the contours of $\tilde{f}$ are piecewise linear curves, as shown by the heavy polygonal line (the dots show where the contour changes triangles, and possibly direction). Since $\tilde{f}$ is determined by its values at the vertices of the current triangle, f is only evaluated at vertices of triangles through which the contour passes; clearly, this involves one new function evaluation per triangle.

Figure 1 about here

In sections 3 through 6 the letter f will stand for the affine approximation of some original, not necessarily differentiable, function from $U \subset \mathbf{R}^n$ into $\mathbf{R}^{n-1}$. Without loss of generality, we assume that the contour that interests us is the inverse image $f^{-1}(0, \ldots, 0)$ of the origin. We assume further that we know a point x_0 in this contour, and we denote by C the connected component of the contour that contains x_0.

3. The Non-Degenerate Case

In order to motivate the algorithm, we start with its simplest possible case: tracing a contour in two dimensions. We leave the triangulation unspecified, but assume that our function $f : \mathbf{R}^2 \to \mathbf{R}$ never vanishes at the vertices of the triangulation, and is never constant on a triangle. This will avoid degenerate behavior, whose treatment we postpone until section 6.

The procedure is iterative. At each step we consider the restriction of the function to one triangle only, which we have just "entered" through an edge for which f is positive at one endpoint and negative at the other (cf. Figure 1(b)). Let the vertices of this triangle be (s_0, s_1, s_2), and assume we entered it through the edge (s_0, s_1), where $f(s_0) > 0$ and $f(s_1) < 0$. Two cases can occur: if $f(s_2) > 0$, the contour must exit the triangle through the edge (s_1, s_2), and if $f(s_2) < 0$, the contour exits through the edge (s_0, s_2). In either case we can find the exit point by interpolation. Then we fetch the third vertex of the triangle that lies on the other side of the exiting edge, which becomes the entering edge of the new triangle. We're ready to restart the cycle.

The iteration ends when we return to the initial triangle or exit the search domain U. As for its beginning, if our initial point x_0 does not happen to lie on an edge of the triangulation, we can also use interpolation to replace it by one that does.

To discuss the case of arbitrary dimension, we start by fixing some nomenclature. Recall that an n-dimensional *simplex* (or *n-simplex*) σ in an n-dimensional affine space is a bounded intersection of $n+1$ closed, affine half-spaces. The hyperplanes determining these half-spaces, taken n at a time, intersect at the *vertices* of σ; there are $n+1$ vertices, they are in general position (that is, there is no hyperplane containing all of them), and their convex hull is σ.

A *k-dimensional face* (or *k-face*) of σ is the convex hull of any k vertices of σ. If the vertices of σ are $s_0, \ldots, s_n \in \mathbf{R}^n$, we call Φ_A, where $A \subset \{0, 1, \ldots, n\}$, the face that is the convex hull of $\{s_j : j \notin A\}$; in particular, $s_i = \Phi_{\{0,\ldots,i-1,i+1,\ldots,n\}}$. *Facets* are $(n-1)$-dimensional faces; $\Phi_{\{i\}}$ is sometimes called facet i, that is, facets are labeled with the index of the opposite vertex. An *edge* is a one-dimensional face and a vertex is a zero-dimensional face. Notice that $\sigma = \Phi_\varnothing$ is an n-dimensional face of itself.

A k-face of σ is clearly a k-simplex in the k-dimensional space in which it lies, and we also say it is a k-simplex in the big space. A face's *interior* is the set of affine combinations, with strictly positive coefficients, of its vertices; its *boundary* is what's left.

A *triangulation* of $\mathbf{R}^n$ is a collection of n-simplices that cover the whole space and whose interiors are disjoint. Here is a simple example of a triangulation in $\mathbf{R}^n$, for n arbitrary: Consider the collection of hypercubes whose vertices have integer coordinates and whose edges have length one and are parallel to the coordinate axes. We subdivide the unit hypercube $[0,1]^n$ into n-simplices by observing that, for every permutation $\pi = (i_1, i_2, \ldots, i_n)$ of the integers $\{1, 2, ..., n\}$, the set of points $(x^1, \ldots, x^n)$ that satisfy the constraints

$$x^{i_1} \le x^{i_2} \le \cdots \le x^{i_n}$$

is an n-simplex, and that these $n!$ simplices triangulate the unit hypercube. The vertices of such a simplex are

$$\begin{aligned}
s_0 &= (0,\dots,0) \\
s_1 &= (0,\dots,1,\dots,0) && \text{(1 in the } i_n\text{-th position),} \\
s_2 &= (0,\dots,1,\dots,1,\dots,0) && \text{(1 in the } i_n\text{-th and } i_{n-1}\text{-th position)} \\
&\dots \\
s_n &= (1,\dots,1);
\end{aligned}$$

notice that s_0 and s_n are shared by all simplices, and the edge they span is a main diagonal of the unit hypercube.

We complete the triangulation of $\mathbf{R}^n$ by translating the triangulation of the unit hypercube to all other hypercubes. Each simplex in this triangulation can be uniquely referenced by the lowest-coordinate corner of the hypercube it's contained in, together with a permutation of $\{1,\dots,n\}$. There is, however, a simpler way to reference each simplex, namely, by the coordinates of its center of mass (which, when multiplied by $n+1$, are integers). It is important to have a simple way to address simplices because, as we shall see later, we must keep track of already-visited ones.

With these examples in mind, we now describe in detail a simple version of the algorithm. Imagine that we have an arbitrary triangulation of $\mathbf{R}^n$. (There is no need to compute the positions of all the simplices of this triangulation; all we will need is a means of finding the simplices that abut a given simplex. Finding a triangulation for which this is easy turns out to be an interesting problem in its own right, and we take it up in the next section.) We will denote by C^σ the (non-empty) intersection of the contour C with a simplex σ of this triangulation.

For this simple version we'll want to avoid two types of situations. The first occurs when C^σ, which normally is a line segment, has dimension greater than one. The second is when C^σ, which normally goes in and out of σ through the interior of facets, happens to intersect also the interior of a face of dimension less than $n-1$. The first type of degeneracy implies the second; both are ruled out for the nonce. In spite of this limitation, this version of the algorithm works surprisingly well—the examples in section 7 were computed using it.

As in the two-dimensional example above, we work by keeping track of three pieces of information, which together constitute the *current state* $(\sigma, k, \mathbf{b})$ and reflect the situation as we are passing into a new simplex:

(1) the *current simplex* σ with vertices $s_0,\dots,s_n \in \mathbf{R}^n$;
(2) the index k of the *current facet* through which we're entering σ (remember that facets are indexed by the subscript of the opposite vertex); and
(3) the *barycentric coordinates* $\mathbf{b}_{\text{in}} = (b^0_{\text{in}},\dots,b^n_{\text{in}})$, of the *current position*.

The current position is the point in the current facet where C enters σ. The barycentric coordinates $\mathbf{b} = (b^0,\dots,b^n)$ of a point $x \in \mathbf{R}^n$ (with respect to an n-simplex σ) are defined by

$$S_\sigma \mathbf{b} = \begin{pmatrix} 1 \\ x \end{pmatrix}, \qquad \text{where} \quad S_\sigma = \begin{pmatrix} 1 & \cdots & 1 \\ s_0 & \cdots & s_n \end{pmatrix}.$$

Since the current position lies on the current facet, the k-th barycentric coordinate b_{in}^{k} is zero.

As we go along the contour, then, our task is to compute the next state $(\sigma', k', \mathbf{b}'_{\text{in}})$ from the current one. Notice that if a point is on a face common to two simplices, its barycentric coordinates with respect to the two are the same, if all the common vertices have the same indices in both simplices (otherwise they are the same up to reordering). In particular, this will be the case if all the vertices of the triangulation can be simultaneously assigned an index in $\{0, 1, \ldots, n\}$ in such a way that, for each simplex, the $n+1$ vertices get all possible $n+1$ indices. From now on we assume the triangulation satisfies this property.

Under this assumption, finding $\mathbf{b}'_{\text{in}}$ is the same as finding the barycentric coordinates $\mathbf{b}_{\text{out}}$ of the point where C exits σ, and finding k' is finding the facet to which $\mathbf{b}_{\text{out}}$ belongs. (We will freely use the barycentric coordinates of a point to stand for the point itself, because at any one time we only work with one system of barycentric coordinates.) In order to find $\mathbf{b}_{\text{out}}$ it will be useful to have the notion of a *barycentric direction*, that is, a non-zero $(n+1)$-tuple $\mathbf{v}$ whose elements have sum zero. We call $\mathbf{v}$ a direction because by adding scalar multiples of it to a barycentric position we generate further barycentric positions that represent points on a line in $\mathbf{R}^n$. Now provided that $\mathbf{v}$, when anchored at $\mathbf{b}_{\text{in}}$, points into σ, the intersection C^σ may be written as $\{\mathbf{b}_{\mathbf{v}}(t) = \mathbf{b}_{\text{in}} + t\mathbf{v} : t \in [0, \tau]\}$ for some barycentric direction $\mathbf{v}$ and some positive $\tau \in \mathbf{R}$. In this notation we can easily write $\mathbf{b}_{\text{out}} = \mathbf{b}_{\mathbf{v}}(\tau)$, and the problem is to find $\mathbf{v}$ and τ (as it turns out, k' will be found at the same time).

To find $\mathbf{v}$ we observe that the value of f at a point x of σ with barycentric coordinates $\mathbf{b} = S_\sigma^{-1}\binom{1}{x}$ is given by the equality

$$F_\sigma \mathbf{b} = \begin{pmatrix} 1 \\ f(x) \end{pmatrix}, \qquad \text{where} \quad F_\sigma = \begin{pmatrix} 1 & \cdots & 1 \\ f(s_0) & \cdots & f(s_n) \end{pmatrix},$$

because F_σ represents the unique affine transformation that gives the right values for $x = s_0, \ldots, s_n$. Thus C^σ is the set of points whose barycentric coordinates $\mathbf{b} = (b^0, \ldots, b^n)$ satisfy

$$F_\sigma \mathbf{b} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \qquad \text{and} \qquad b^i \geq 0 \quad \text{for all } i.$$

Here the equality has at least one solution $\mathbf{b}_{\text{in}}$, so the dimension of its set of solutions is $n+1$ minus the rank of F_σ. (Thus the first type of degeneracy mentioned above occurs when F_σ does not have full rank n.) The general solution of the equality can be written $\mathbf{b}_{\text{in}} + t\mathbf{v}$, where $\mathbf{v}$ is a non-trivial solution of the homogeneous equation

$$(*) \qquad F_\sigma \mathbf{v} = 0.$$

Finding $\mathbf{v}$ is therefore just a standard linear algebra problem.

To ensure that a barycentric vector anchored at $\mathbf{b}_{\text{in}}$ and pointing in the direction $\mathbf{v}$ points into σ, we simply look at the k-th coordinate v^k of $\mathbf{v}$ and multiply $\mathbf{v}$ by -1 if this coordinate is negative. We observe for later use that a vector anchored in σ, and in the direction $\mathbf{v}$, points away from facet i if v^i is positive, points towards facet i if v^i is negative and is parallel to facet i if v^i is zero. If v^k were zero C would run along facet k and would intersect a lower-dimensional face, so we would have the second kind of degeneracy mentioned above. Furthermore, since facet k is shared with the previous simplex (unless we're just starting the process), this would imply that in that simplex there was a degeneracy too.

It is now a simple matter to find the exiting facet in which $\mathbf{b}_{\text{out}}$ lies. In general, $\mathbf{b}_{\mathbf{v}}(t)$ lies on an extended facet of σ whenever it has at least one zero coordinate. For t slightly larger than 0, $\mathbf{b}_{\mathbf{v}}(t)$ lies in the interior of σ (because $\mathbf{v}$ points into σ), so τ is the smallest positive number t for which one of the coordinates of $\mathbf{b}_{\mathbf{v}}(t)$ is 0. In symbols,

$$(**) \qquad \tau = \min\{-b^i_{\text{in}}/v^i : v^i \neq 0, -b^i_{\text{in}}/v^i > 0\}.$$

The index k' that achieves the minimum gives the exiting facet, that is, $-b^{k'}_{\text{in}}/v^{k'} = \tau$. (If the minimum were achieved for more than one k' the contour would pass out of σ through one of its faces of dimension less that $n-1$, and a degeneracy of the second kind would occur.)

The simplex σ' into which C now passes shares all but one of its vertices with σ, namely, $s_{k'}$. Thus to find σ' we just need to compute the new $s_{k'}$. In the next section we show how our triangulation of $\mathbf{R}^n$ can be chosen in such a way that this is very easy to do.

4. Triangulations

So far we have identified two properties that we would like our triangulation to have: (a) it should be easy to find the simplex that shares a facet with a given simplex; and (b) it should be possible to label the vertices of all the simplices at the same time with indices $0, \ldots, n$, in such a way that each of the $n+1$ vertices of an n-simplex has a different label. Two more conditions are intuitively desirable: (c) all simplices should have more or less the same size, and (d) their dimensions should be roughly the same in all directions. The reasoning here is that, having normalized the largest dimension of a simplex to the pixel size, say, we don't want to cross too many very small or very skinny simplices, because we would be progressing more slowly for the same amount of computation. (Slowing down to gain in resolution would be a good idea if we could do it only when the function requires it—this is the variable step size approach. But since we're choosing the triangulation beforehand, a large amount of variability in step size is wasteful.)

There is a whole class of triangulations, in all dimensions, that fit these requirements beautifully, namely, *monohedral triangulations generated by reflections.*

"Monohedral" means that all n-simplices are congruent, and "generated by reflections" means that they can all be obtained from a fixed one by successive reflections in its facets. (Alternatively, if two simplices share a facet they are reflections of each other in that facet.) Coxeter [1934] has enumerated all such triangulations, and this section presents very briefly their unifying properties. Although this material is not difficult, it is by nature more mathematical than the remainder of the paper, and a reader interested only in the practical aspects of our algorithm can skip to the paragraph containing formulas ($*$) and ($**$), at the end of this section.

A simplex in such a triangulation must have dihedral angles that are each a submultiple of π. In other words, for each pair of facets i and j of the simplex, there is an integer $p_{ij} > 1$ such that the dihedral angle between the two facets is π/p_{ij}. To see why, consider the dihedral angle between two facets of one of these special simplices, and imagine alternately reflecting the simplex in each of the two facets. After a finite number of reflections we must arrive back at the original simplex, in exactly the position in which we started; otherwise this simplex does not triangulate by reflection. Thus an integer number times the dihedral angle between our two faces is equal to 2π. If we further demand that the last simplex in this chain has the same orientation as the first (that is, a set of vertex labels on the first simplex will get reflected back into position on the last simplex), this integer number must be even—an odd number of reflections will reverse the orientation.

Coxeter introduced a shorthand way of identifying, up to similarity, a simplex of one of our triangulations, and, by extension, the triangulation itself. The *Coxeter diagram* of such a simplex is a graph with a node for each facet and with $p_{ij} - 2$ undirected edges connecting the nodes corresponding to facets i and j. Thus, two nodes have no edges between them if the angle between the corresponding facets is $90°$, a single edge if $60°$, and so on.

Figure 2 is Coxeter's enumeration, showing Coxeter diagrams and his names for the simplices they represent (P_m, Q_m, and so on). The subscript in these symbols stands for the number of vertices of the simplex, $n + 1$ in our notation; from now on we will consistently use $m = n + 1$. The vertex and edge labels are explained below.

Figure 2 about here

It is important to remember when interpreting a Coxeter diagram that each *node* of the diagram corresponds to a *facet* of the corresponding simplex. This can be particularly confusing for the 2-dimensional case; look at the diagram for P_3. It looks like an equilateral triangle, and in fact that is precisely what it represents, but only by accident: the correct way to read the diagram is by observing that it stands for a simplex with three edges (nodes), each pair of which have a $60°$ angle between them (one edge in the diagram).

A sample of each of the three two-dimensional Coxeter triangulations, P_3, R_3 and V_3 is given in figure 3.

Figure 3 about here

Notice that there are four families that have representatives in all but a few low dimensions, namely, P_m, Q_m, R_m and S_m. These are the natural candidates for an algorithm that is meant to work in any dimension, although the remaining entries in the list might be useful in their specific dimensions.

To wrap up the description of the iteration step of our algorithm, in section 3, we needed to display a rule to compute the simplex that shares a given facet with the current simplex. Here this boils down to finding the reflection of a vertex s_i of the simplex in the opposite facet. The reflection of a point p in a hyperplane H is reached by going from p to its orthogonal projection $\pi_H(p)$ on H, then once again as far in the same direction: in symbols, $2\pi_H(p) - p$. Finding $\pi_H(p)$ is straightforward given the equation of H and the coordinates of p, but it turns out that we can avoid all of that because of the geometry of the simplices. Our treatment loosely follows Coxeter's [1973, p. 182–184].

Let $\pi_i(s_i)$ be the projection of s_i in the hyperplane containing facet i. We first write $\pi_i(s_i)$ in barycentric coordinates with respect to the vertices of facet i: $\pi_i(s_i) = \sum_{j \neq i} w_i^j s_j$, with $\sum w_i^j = 1$. To figure out the w_i^j, consider the plane containing the edge that joins s_i and s_j and orthogonal to the opposite face $\Phi_{\{i,j\}}$, which it meets at u. The intersection of the simplex with this plane is shown in figure 4. In this figure, w_i^j is, by definition, the ratio between the length of the segment $u\pi_i(s_i)$ and the side us_j. Now the length of $u\pi_i(s_i)$ equals the length of side us_i times the cosine of the angle at u, which is just the dihedral angle $\theta_{ij} = \pi/p_{ij}$; since the sides of a triangle are inversely proportional to the corresponding altitudes, we can write $w_i^j = \cos\theta_{ij} \cdot (h_i/h_j)$, where h_i and h_j are the altitudes of the simplex corresponding to vertices s_i and s_j.

Figure 4 about here

To calculate the altitudes h_i, we first show that $\sum_{i=0}^{n} \dfrac{\mathbf{n}_i}{h_i} = 0$, where $\mathbf{n}_i$ is the unit vector normal to facet i and pointing out. This equality can be verified by considering its inner product with each vector $s_j - s_0$: we have $\mathbf{n}_i \cdot (s_j - s_0) = 0$ unless $i = 0$ or j, in which cases the inner product is h_0 or $-h_j$, respectively. Since the n vectors $s_j - s_0$ form a basis for $\mathbf{R}^n$, this implies the equality.

We have just shown that the expression $\left\|\sum_{i=0}^{n} \mathbf{n}_i x^i\right\|^2$ vanishes when the x^i are inversely proportional to the altitudes h_i. Now this expression is a quadratic form in $(x^0, \ldots, x^n)$, and the entries a_{ij} of its associated matrix $\mathbf{A}$ are just the inner products $\mathbf{n}_i \cdot \mathbf{n}_j$ of the unit face normals of the simplex. More precisely, we have

$$a_{ij} = \begin{cases} 1 & \text{if } i = j, \\ -\cos\pi/p_{ij} & \text{if } i \neq j. \end{cases}$$

Because the $\mathbf{n}_i$, taken n at a time, are linearly independent, $\mathbf{A}$ has rank n, and its kernel is one-dimensional: in other words, all vectors in the kernel are proportional, and if we find any one of them we can get the altitudes (up to scaling) by taking the reciprocals of its coordinates. This, of course, needs to be done only once for each triangulation, and in fact figure 2 shows these numbers for all the Coxeter triangulations: the node labels of the diagrams represent the coordinates of a vector in the kernel of **A**.

For example, here is **A** for P_3, R_3 and V_3:

$$\begin{pmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & -\frac{\sqrt{2}}{2} & 0 \\ -\frac{\sqrt{2}}{2} & 1 & -\frac{\sqrt{2}}{2} \\ 0 & -\frac{\sqrt{2}}{2} & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & -\frac{1}{2} & 0 \\ -\frac{1}{2} & 1 & -\frac{\sqrt{3}}{2} \\ 0 & -\frac{\sqrt{3}}{2} & 1 \end{pmatrix}$$

The vectors in their kernel can be taken as $(1,1,1)$, $(1,\sqrt{2},1)$ and $(1,2,\sqrt{3})$, as shown in figure 2.

To complete the determination of the weights w_i^j, we just need to apply the formula $w_i^j = \cos\theta_{ij}\cdot(h_i/h_j)$. Notice that w_i^j vanishes when facets i and j meet at right angles, so we only have to keep track of pairs (i,j) whose corresponding nodes in the Coxeter diagram have an edge between them. At this point, each edge of the Coxeter diagram has two weights associated with it, corresponding to the pairs (i,j) and (j,i), but, because there are no loops in the diagrams (except for the case of P_m, when all the vertices are equivalent anyway), we can arrange to match the two weights associated with each edge by scaling each set of weights associated with a vertex by the same factor. (Then the weights no longer have sum 1, and we must remember to scale them back before using them.) Thus we get a single set of edge labels for the Coxeter diagram that describe all the projections, and hence all the reflection rules. These edge labels, too, are shown in figure 2.

As an example of how simply these edge labels describe the reflection rules, we consider the valence-three vertex of T_9. The edge labels say to weight the three adjacent vertices with weights 9/36, 12/36, 15/36, to get the projection onto the opposite facet. The reflection through that facet is then easy, as described above: just add the three adjacent vertices with weights twice as heavy, and subtract the valence-three vertex.

At this point it is worthwhile to spell out a prototype simplex and the reflection rules for two important families of triangulations, P_m and R_m. The vertices of an R_m simplex can be taken to be $s_i = (1,\ldots,1,0,\ldots,0)$, where the first i coordinates are 1. (Here we assume the nodes of the Coxeter diagram are numbered in order, with 0 and n at the two ends.) The vertices of a P_m are $s_i = (m-i)\big((m(m-1))^{-1/2},((m-1)(m-2))^{-1/2},\ldots,((m-i+1)(m-i))^{-1/2},0,\ldots,0\big)$, where the first i coordinates are non-zero. (Here the nodes of the Coxeter diagram are numbered in order around the circle, with 0 and n next to one another.) The reflection rules for P_m are extremely simple: to reflect vertex s_i in the opposite facet,

$$(*) \qquad \text{replace } s_i \text{ by } s_{i-1} + s_{i+1} - s_i,$$

where we agree to take the indices modulo m whenever we talk about P_m simplices. The next simplest rules occur for R_m:

$$(**) \qquad \text{replace } s_i \text{ by } \begin{cases} 2s_1 - s_0 & \text{if } i = 0, \\ s_{i-1} + s_{i+1} - s_i & \text{if } 0 < i < n, \\ 2s_{n-1} - s_n & \text{if } i = n. \end{cases}$$

Once we have expressed a particular set of reflection rules in the form $s_i \leftarrow \sum_{j \neq i} w_i^j s_j$, we can apply them to an arbitrary simplex, having nothing to do with the Coxeter triangulation. The rules will no longer describe reflections, but they will generate a triangulation of $\mathbf{R}^n$, because the rules are preserved by affine transformations and we can always find an affine transformation to transform the Coxeter simplex into our newfangled simplex. Thus it is that, if we apply the P_m reflection rules to an R_m simplex, we get the example triangulation in section 3. Like P_m, but unlike R_m, it has the same number of simplices incident at each vertex, as illustrated by figure 5 in the case $m = 3$: in (a), R_3 with R_3 rules, and in (b), R_3 with P_3 rules. (These two triangulations are referred to as K and H in the fixed-point literature; see, for example, [Todd 1976].) Notice that, in general, triangulations obtained in this way from arbitrary simplices are not monohedral, and so may be inferior from the point of view of criterion (c) at beginning of this section; but such is not the case with the ones in figure 5.

Figure 5 about here

5. Some Implementation Details

To complete the non-degenerate algorithm, we must choose a triangulation of the appropriate dimension from figure 2, then find an appropriate instance of the prototype simplex containing the given starting point. We also need a test to determine when we have found the entire contour. We take up each of these points in turn.

5.1. Choice of Triangulation

We have seen that all Coxeter triangulations satisfy criteria (a)–(c) at the beginning of section 4. We now evaluate them according to criterion (d), namely, whether the simplices are nice and chunky or long and skinny. Since this criterion is a bit fuzzy, we use three different, but related, measurements in the evaluation. As it turns out, the P_m family comes out slightly ahead according to all three. Todd [1976] contains a discussion of the efficiency of triangulations, and could profitably be compared with the presentation here.

Perhaps the most direct of the three measures is the expected step size along the contour, that is, the average distance traversed within a simplex. This should be compared with the greatest possible step size—the diameter of the simplex—if we are scaling the triangulation to make sure that the step size is never more than a fixed length. (This may be the length represented by one pixel of the output device, for example. See below for alternative normalizations.)

To estimate the average step size, we assume the contour is a long, randomly placed straight line segment. The number of simplices visited by this segment is equal to the sum, over all the families of parallel hyperplanes defined by facets, of the number of elements of the family that the segment crosses. For a family H whose elements are separated by distance d_H and have normal $\mathbf{n}_H$, this number is roughly $L\left|\mathbf{n}_H \cdot \mathbf{x}\right|/d_H$, where L is the length of the segment and $\mathbf{x}$ a unit vector in its direction. As we average over $\mathbf{x}$, we get a constant, which depends only on the dimension, times L/d_H. Thus the average distance traversed within each simplex is a constant divided by $\sum_H 1/d_H$, where H runs through all the families of hyperplanes containing facets of the triangulation. (Explaining how to determine the number of such families and the separation between hyperplanes in them would take us too far afield.) The results are summarized in table 1, where we show the ratio between maximum and expected step sizes for Coxeter simplices up to dimension 8. The Q_m, R_m and S_m families have asymptotically the same value, and the P_m family has asymptotically half that value.

n	P_m	Q_m	R_m	S_m	T_m	U_m	V_m
2	2.21		3.07				5.68
3	4.24		6.27	4.97			
4	6.57	7.20	10.60	8.90		12.30	
5	9.74	11.86	16.05	13.95			
6	13.20	17.64	22.63	20.14	19.96		
7	17.50	24.55	30.34	27.45	34.10		
8	22.09	32.59	39.18	35.89	49.39		

Table 1. Maximum over expected step sizes for Coxeter simplices

Another simple measure of the nonregularity of a simplex is the ratio between the radii of its circumscribed and inscribed spheres, normalized to one for the regular simplex of the same dimension. Table 2 shows this value up to dimension 8, and again the pattern is already clear: the Q_m, R_m and S_m entries are virtually indistinguishable from one another (asymptotically they are the same) and more than 50% bigger than those for P_m (the asymptotic ratio is $\sqrt{3}$).

The third, and subtlest, measure of nonregularity is the condition number of the covariance matrix of a random point, uniformly distributed inside the simplex. (The *condition number* of a matrix (for the Euclidean norm) is the ratio between its largest and smallest eigenvalues. The condition number for regular simplices is always 1, so no normalization is necessary here.) The logs of these numbers are given in Table 3, again up to dimension 8 only because the relative behavior after that is the same:

n	P_m	Q_m	R_m	S_m	T_m	U_m	V_m
2	1.00		1.21				1.37
3	1.05		1.39	1.61			
4	1.12	1.50	1.56	1.61		1.81	
5	1.18	1.70	1.71	1.76			
6	1.25	1.86	1.85	1.90	2.00		
7	1.31	2.01	1.98	2.03	2.32		
8	1.37	2.14	2.10	2.15	2.72		

Table 2. Normalized ratios of circumscribed to inscribed spheres

n	P_m	Q_m	R_m	S_m	T_m	U_m	V_m
2	0.00		2.20				3.18
3	1.39		3.53	4.29			
4	1.92	3.22	4.50	4.09		5.70	
5	2.77	4.64	5.27	5.03			
6	3.24	5.54	5.91	5.76	5.74		
7	3.84	6.22	6.46	6.36	7.33		
8	4.23	6.78	6.94	6.87	8.72		

Table 3. Logs of the condition numbers of Coxeter simplices

All measures seem to indicate that P_m has a somewhat better geometry for our purposes than all other Coxeter triangulations. It also has the simplest reflection rules. However, it has one slight disadvantage over R_m, in that it requires messier arithmetic to pass from real-world coordinates to simplex coordinates.

5.2. Starting Simplex

Once a triangulation has been chosen, we must find a simplex containing the starting point x_0. We listed at the end of section 4 a prototype simplex for the P_m and R_m families. The other two families are equally easy: the S_m simplex can be obtained by gluing two R_m simplices along their facet 0, and the Q_m simplex by gluing two S_m simplices along their facet n. Instances of simplices for the remaining triangulations are described in table 4; the second row indicates the labeling (cf. figure 2), L is the diameter of the simplex, and the $\mathbf{e}_i$ are elements of the canonical basis of $\mathbf{R}^n$.

(For those who insist on having the whole story, here's how you might find the simplices yourself: First compute an $n \times m$ matrix $\mathbf{V}$ whose columns $\mathbf{n}_i$ are unit face normal vectors, by recalling that the matrix $\mathbf{A}$ defined in section 4 equals $\mathbf{V}^t\mathbf{V}$; completing squares in the form $\mathbf{x}^t\mathbf{A}\mathbf{x}$ leads to $\mathbf{V}$. Next observe that, by construction, the inner product of $\mathbf{n}_i$ with the vector going from the center of a sphere of radius 1 to a vertex s_j of the simplex circumscribed around that sphere, and having the $\mathbf{n}_i$ as face normals, is 1 if $i \neq j$ and $1 - h_i$ if $i = j$. Since you

	V_3	U_5	T_7	T_8	T_9
	$0 \equiv 1-2$	$0-1=2-3-4$	$0-1-\underset{\substack{\vert \\ 5 \\ \vert \\ 4}}{6}-3-2$	$0-1-2-\underset{\substack{\vert \\ 6}}{7}-5-4-3$	$0-1-\underset{\substack{\vert \\ 2}}{3}-4-5-6-7-8$
L	2	$2\sqrt{3}$	$12\sqrt{5}$	$12\sqrt{5}$	$6\sqrt{70}$
s_0	0	0	0	0	0
s_1	$\mathbf{e}_1$	$\sqrt{3}\mathbf{e}_1$	$3\sqrt{30}\mathbf{e}_1$	$4\sqrt{15}\mathbf{e}_1$	$3\sqrt{35}\mathbf{e}_1$
s_2	$s_1+\sqrt{3}\mathbf{e}_2$	$s_1+\mathbf{e}_2$	$s_1+15\sqrt{2}\mathbf{e}_2$	$s_1+4\sqrt{5}\mathbf{e}_2$	$s_1+7\sqrt{5}\mathbf{e}_2$
s_3		$s_2+\sqrt{2}\mathbf{e}_3$	$\frac{1}{5}(3s_1+2s_2)+6\sqrt{3}\mathbf{e}_3$	$s_2+20\mathbf{e}_3$	$\frac{1}{7}(4s_1+3s_2)+2\sqrt{15}\mathbf{e}_3$
s_4		$s_3+\sqrt{6}\mathbf{e}_4$	$\frac{1}{2}(s_1+s_3)+9\sqrt{5}\mathbf{e}_4$	$\frac{1}{5}(3s_2+2s_3)+4\sqrt{6}\mathbf{e}_4$	$s_3+2\sqrt{21}\mathbf{e}_4$
s_5			$\frac{1}{3}(s_1+s_3+s_4)+3\sqrt{10}\mathbf{e}_5$	$\frac{1}{2}(s_2+s_4)+2\sqrt{10}\mathbf{e}_5$	$s_4+3\sqrt{14}\mathbf{e}_5$
s_6			$\frac{1}{3}(s_0+s_3+s_5)+\sqrt{30}\mathbf{e}_6$	$\frac{1}{2}(s_2+s_5)+4\sqrt{5}\mathbf{e}_6$	$s_5+\sqrt{210}\mathbf{e}_6$
s_7				$\frac{1}{8}(3s_2+3s_5+2s_6)+\sqrt{15}\mathbf{e}_7$	$s_6+2\sqrt{105}\mathbf{e}_7$
s_8					$s_7+6\sqrt{35}\mathbf{e}_8$

Table 4. An instance of T_m, U_m and V_m simplices

now know the inner products of this vector with a basis of $\mathbf{R}^n$, consisting of, say, the first n unit normals, you can determine all the vertices by inverting the basis matrix.)

Then we have to scale the prototype simplex to give a reasonable step size. As mentioned above, a reasonable strategy is to normalize the maximum step size to the desired resolution. Another possibility is to make the expected step size equal to the resolution, or perhaps half the resolution; the expected step size can be determined as explained at the beginning of subsection 5.1.

Finally, we have to find an instance of the scaled simplex that contains the starting point. This can be done in two ways, depending on the application. If only one contour of the function is to be traced, we can translate our scaled prototype simplex so that x_0 equals, say, the barycenter of its facet n. Then we initialize the state accordingly ($k=n$, $\mathbf{b}_{\text{in}} = (1/n, \ldots, 1/n, 0)$ and σ as appropriate). If, on the other hand, we are computing several contours, we may desire them all to use the same triangulation, rather than computing a separate translation of the triangulation for each contour. In this case, we must fix one particular instance of our triangulation and find, for each contour to be traced, the particular simplex that contains the starting point. This can be a little tricky; a nice treatment of essentially the same problem is given in [Conway and Sloane 1982]. Unless x_0 happens to lie on a simplex face already, we also have to find our first $\mathbf{b}_{\text{in}}$ and k; as the reader can easily verify, this is accomplished by solving for $\mathbf{v}$ and τ as in section **3** (formulas $(*)$ and $(**)$, with $\mathbf{b}_{\text{in}}$ replaced by the barycentric coordinates of x_0 in the latter), then adding $\tau\mathbf{v}$ to x_0.

5.3. Termination

Termination of the algorithm occurs in two cases: when the contour exits the region U of interest, and when it loops. An out-of-bounds check is straightforward; we stop, say, when one vertex of σ falls outside the region of interest (f may not be defined there anyway). Then we restart the algorithm from x_0 in the opposite direction; having saved the initial state $(\sigma, k, \mathbf{b}_{\text{in}})$ we just have to replace σ by its reflection in facet k.

There is a simple integer test for loops as well. In a basis given by n edges $\bar{s}_i - \bar{s}_0$ of a *fixed* simplex of the triangulation, the coordinates of the barycenter of any simplex (minus $\bar{s}_0$) are integral multiples of $1/m$, so we can multiply these coordinates by m to obtain an integer vector that uniquely identifies a simplex. As a matter of fact, it can be advantageous to work with coordinates in the basis $\{\bar{s}_i - \bar{s}_0\}$ for other computations as well, including applications of the reflection rules; this avoids possible accumulation of roundoff errors.

Checking if we're back to the start, then, is trivial: we just compare the integer coordinates of the current simplex with those of the starting simplex, and stop if they are equal. It may also be useful to keep a hash table of the simplices that have been visited; for example, if we have a method of generating starting points on different components of the same contour, we will probably want a test for whether two starting points lie on the same component.

6. The Degenerate Case

Having completed the description of the non-degenerate algorithm, we now tackle the degenerate case. Recall first that two sorts of degeneracies can occur: the first is when F_σ has rank less than n for the current simplex σ, and the second when C passes through a face of σ of dimension less than $n-1$. The following notation will be useful: r is the dimension of the kernel of F_σ, and q is the codimension of the face of σ through which we are exiting the simplex (this will be made more precise later). The two types of degeneracy correspond to $r > 1$ and $q > 1$, respectively.

An apparently simple idea to resolve degeneracies is to jitter a bit the value of f at some of the vertices of σ, thereby restoring genericity. The problem with this is that the particular direction and amount of jitter will determine the exiting facet of σ, so it is critical to apply the same jitter to any vertices shared by σ and the new simplex, lest we lose the contour. Now if we are holding these values in the matrix F_σ this will happen automatically for the next simplex; but if the contour, at some later time, passes through a simplex that shares one or more jittered vertices with σ, the jittering information will have been lost. We could try keeping a hash table of which vertices have been jittered and which haven't, effectively "freezing" the changed (and unchanged) values of f at all vertices of all simplices already visited, but this still wouldn't work, since a jitter might very well be required while we're in a simplex all of whose vertices have already been frozen on visits to adjacent simplices. For this reason we use an approach that does not require altering the value of f.

Whenever it is non-empty, C^σ is a polytope of dimension at most r, and equal to r if C intersects the interior of σ. The *1-skeleton* of C^σ is the union of all its edges, and the *1-skeleton* of C is the union of the 1-skeletons of C^σ. The algorithm described below draws the 1-skeleton of C. It uses a depth-first search, so it can easily be adapted to stop after drawing, say, a path in the 1-skeleton that starts and ends at the boundary of the region of interest (or loops back). Of course, in the non-degenerate case, the two outputs are equivalent, and also equivalent to the output of the partial algorithm described before, since then C is its own 1-skeleton.

The description of the 1-skeleton uses the fact that the *open faces* of a triangulation, that is, the interiors of the faces (remember that n-simplices are faces, too), partition $\mathbf{R}^n$. For example, in $\mathbf{R}^2$, each point lies either on a vertex, or along an edge of a triangle (but not on a vertex) or inside a triangle (but not on an edge). To describe the 1-skeleton we give information about its vertices, which we call *nodes* to avoid confusion with the vertices of the triangulation, and about its edges, which we call *links*. The nodes are output as points in $\mathbf{R}^n$, but they are remembered by the algorithm as open faces, because an open face cannot contain more than one node of C—otherwise it would contain a whole link, and the nodes would actually be in lower-dimensional open faces. The links don't need to be stored, and they only need to be output when they connect something other than the previously output node and the next.

We will keep a three-part state $(\sigma, A, \mathbf{b}_{\text{in}})$, where σ and $\mathbf{b}_{\text{in}}$ are as in section 3 and $A \subset \{0, \dots, n\}$ indexes the face through which we entered σ. It is an invariant of the algorithm that Φ_A intersects C in a single point, that is, $\Phi_A \cap C$ is a vertex of the polytope C_σ; furthermore A is never empty.

6.1. Crossing a Simplex

Our task after entering a simplex σ through a face Φ_A is to find at least one not previously visited link starting at the current point $\mathbf{b}_{\text{in}}$ and contained in σ. Let's first consider how to characterize segments in C_σ that start at the current point, then how to characterize the condition that such a segment is a link.

A direction $\mathbf{v} \in \ker F_\sigma$, anchored at the current point, points into σ if and only if $v^i \geq 0$ for all $i \in A$. In addition, not all of these v^i can be zero, otherwise $\mathbf{v}$ is parallel to Φ_A, which implies that C intersects Φ_A in more than one point. Since we're only interested in $\mathbf{v}$ up to a factor, we can express this last condition by saying that $\sum_{i \in A} v^i = 1$. Thus finding a segment in C_σ starting at the current point reduces to finding $\mathbf{v} = (v^0, \dots, v^n) \in \mathbf{R}^{n+1}$ such that

(i) $F_\sigma \mathbf{v} = 0$,
(ii) $\sum_{i \in A} v^i = 1$, and
(iii) $v^i \geq 0$ for $i \in A$.

By definition, such a segment $S_\mathbf{v} = \{\mathbf{b}_{\text{in}} + t\mathbf{v} : t \in [0, \tau]\}$ determines a link—an edge of C^σ—if and only if it does not intersect any segment whose endpoints $\mathbf{b}_1, \mathbf{b}_2$

are in C^σ but not in $S_\mathbf{v}$. We claim that, if $B_\mathbf{v} = \{i \in A : v^i = 0\}$ is the set of A coordinates of $\mathbf{v}$ that vanish, this condition is equivalent to saying that there is no solution $\mathbf{v}' \neq \mathbf{v}$ to (i) and (ii) above with $B_\mathbf{v} \subseteq B_{\mathbf{v}'}$. Geometrically, $B_\mathbf{v}$ is the label of the smallest face of σ that contains $S_\mathbf{v}$, and the claim is that $S_\mathbf{v}$ determines a link if and only if $\Phi_{B_\mathbf{v}} \cap C^\sigma$ has dimension one.

For suppose there is such a $\mathbf{v}'$: then there are solutions to (i)–(iii) on both sides of $\mathbf{v}$, say $\mathbf{v}_1 = (1-\varepsilon)\mathbf{v} + \varepsilon\mathbf{v}'$ and $\mathbf{v}_2 = (1+\varepsilon)\mathbf{v} - \varepsilon\mathbf{v}'$, for $\varepsilon > 0$ sufficiently small. For t sufficiently small, $\mathbf{b}_1 = \mathbf{b}_{\text{in}} + t\mathbf{v}_1$ and $\mathbf{b}_2 = \mathbf{b}_{\text{in}} + t\mathbf{v}_2$ belong to C^σ, and the segment $\mathbf{b}_1\mathbf{b}_2$ intersects $S_\mathbf{v}$, so $S_\mathbf{v}$ is not a link. Conversely, assume there is a segment in C^σ whose endpoints $\mathbf{b}_1$ and $\mathbf{b}_2$ do not lie in $S_\mathbf{v}$; then $\mathbf{v}$ is a convex linear combination of the directions $\mathbf{v}_1$ and $\mathbf{v}_2$ corresponding to $\mathbf{b}_1$ and $\mathbf{b}_2$ and since, by assumption, $v_1^i \geq 0$ and $v_2^i \geq 0$ for $i \in A$, this implies $B_\mathbf{v} \subseteq B_{\mathbf{v}_1}$ and $B_\mathbf{v} \subseteq B_{\mathbf{v}_2}$.

In order to identify directions $\mathbf{v}$ such that $\Phi_{B_\mathbf{v}} \cap C^\sigma$ has dimension one, we define the following routine:

```
*solve*$(F, A, B)$
{
  ⟨ try to find $\mathbf{v} = (v^0, \ldots, v^n) \in \mathbf{R}^{n+1}$ such that $F\mathbf{v} = 0$, $\sum_{i \in A} v^i = 1$, and
      $v^i = 0$ for $i \in B$ ⟩;
  **if** ($\mathbf{v}$ exists) {
    **if** ($\mathbf{v}$ is unique) {
      **if** ($v^i \geq 0$ for all $i \in A$) **return** $\mathbf{v}$;
      **else if** ($v^i \leq 0$ for all $i \in A$) **return** $-\mathbf{v}$;
        **else return** *FAIL*;
    }
    **else return** $\infty$;
  }
  **else return** *FAIL*;
}
```

By the preceding paragraphs, any vector returned by *solve* points along a link. If *solve* returns ∞, there might be several links contained in Φ_B, and we have to decrease Φ_B, that is, increase B, in order to locate them. Finally, if *solve* returns *FAIL*, there is no link in Φ_B.

(Since *solve* is part of the algorithm's inner loop, it should be as fast as possible. Fortunately, it does little more than trying to solve a linear system of equations, so one can rely on routines from existing packages such as LINPACK [Dongarra *et al.* 1979], which will also provide the necessary information about the rank. But it may be worthwhile for *solve* to reduce the order of the system, by screening out the variables not in B, before calling the equation-solving routine.)

We are now ready to enumerate all the links that go out of the current position and into the current simplex. This is done by repeatedly calling *find_link*, which returns the direction $\mathbf{v}$ of one such link, or *FAIL* if no more links exist. Basically, *find_link* calls *solve*(F_σ, A, B) for B ranging over all subsets of A, in some order, until it either gets a vector, which is returned, or runs out of subsets, in which case *FAIL* is returned. When *find_link* returns a vector, it remembers what subspace it was working on, so the next invocation of *find_link* starts its investigation where the previous one left off.

As stated, this leads to one invocation of $solve$ for each subset of A, of which there can be up to 2^n, but we can actually skip many of them. Namely, when $solve(F_\sigma, A, B)$ returns $FAIL$ or a vector, we can skip all supersets of B, because we already have the unique solution in Φ_B (or the certainty that there is none), so there is no sense in looking for solutions in smaller faces. Similarly, when $solve(F_\sigma, A, B)$ returns ∞ or a vector, we skip all subsets of B, because a face bigger than Φ_B contains either infinitely many solutions or a unique solution that we already know. In particular, in the non-degenerate case, $solve(F_\sigma, A, \varnothing)$ returns either a vector or $FAIL$, and we can skip all other values of B. In the fully degenerate case ($f = 0$ and A is a vertex), *find_link* calls $solve$ $2n-1$ times, and finds n links.

There is a total order on the set $\mathfrak{P}(A)$ of subsets of A that makes the implementation of these ideas particularly simple. To compare two sets $B, B' \in \mathfrak{P}(A)$, we write the elements of B and B' in ascending order, and declare $B < B'$ if and only if the word thus obtained for B comes before the word for B' in dictionary order. For example, the ordering on $\mathfrak{P}(\{0,1,2\})$ is $\varnothing < \{0\} < \{0,1\} < \{0,1,2\} < \{0,2\} < \{1\} < \{1,2\} < \{2\}$. To find the successor of B in this order, we write $B = \{b_0, \ldots, b_i\}$ in ascending order and, assuming for simplicity that $A = \{0, 1, \ldots, k\}$, we set:

$$successor(B) \leftarrow \begin{cases} \{b_0, \ldots, b_{i-2}, b_{i-1}, b_i, b_i + 1\} & \text{if } b_i < k, \\ \{b_0, \ldots, b_{i-2}, b_{i-1} + 1\} & \text{if } b_i = k \text{ and } i > 0, \\ FAIL & \text{if } b_i = k \text{ and } i = 0. \end{cases}$$

Skipping supersets of B is also easy:

$$next\_non\_superset(B) \leftarrow \begin{cases} \{b_0, \ldots, b_{i-2}, b_{i-1}, b_i + 1\} & \text{if } b_i < k, \\ \{b_0, \ldots, b_{i-2}, b_{i-1} + 1\} & \text{if } b_i = k \text{ and } i > 0, \\ FAIL & \text{if } b_i = k \text{ and } i = 0. \end{cases}$$

We also need to keep a list of values of B for which $solve$ returned ∞ or a vector, called the *hit list*, so we can skip all subsets of sets in this list. Whenever $solve$ returns ∞ or a vector for some B, we add B to the hit list. We also delete any existing subsets of the newly added B from the hit list, as they are now redundant: in future tests, any subset of such a subset will also be a subset of B.

When describing the non-degenerate algorithm, we mentioned that we might want to keep a hash table of simplices already visited. Here such a table becomes a necessity, in a slightly different form: we must store each pair (σ, A), together with a pointer to a structure $\mathcal{B}_{(\sigma,A)}$ that contains the hit list and the value of B at the end of the last call to *find_link*(σ, A). This structure is read every time *find_link*(σ, A) is invoked—these invocations being non-consecutive, since we're conducting a depth-first search—and destroyed when *find_link* knows there are no more vectors to be returned. (Actually, $\mathcal{B}_{(\sigma,A)}$ may not be created at all if we only have to call *solve* once, with $B = \varnothing$.) A pair (σ, A) in the hash table is called *dead* if $\mathcal{B}_{(\sigma,A)}$ no longer exists; otherwise it is called *alive*.

```
find_link(σ, A)
{
    if ($\mathcal{B}_{(\sigma,A)}$ exists)     /* we have seen these arguments before */
        ⟨ get $B$ and the hit list from $\mathcal{B}_{(\sigma,A)}$ ⟩;
    else ⟨ set $B$ to $\varnothing$ and the hit list to the empty list ⟩;
    while ($B \neq$ FAIL) {     /* there are still faces to examine */
        if ⟨ $B \subset B'$ for some $B'$ in the hit list ⟩ {
            $B \leftarrow$ successor($B$);     /* skip this $B$ */
            continue;     /* go back to beginning of while loop */
        }
        $\mathbf{v} \leftarrow$ solve($F, A, B$);     /* look for solutions in $\Phi_B$ */
        if ($\mathbf{v}$ = FAIL) {     /* no solution */
            $B \leftarrow$ next_non_superset($B$);     /* skip this $B$ and all its supersets */
            continue;
        }
            /* at least one solution */
        ⟨ delete subsets of $B$ from the hit list ⟩;
        ⟨ add $B$ to the hit list ⟩;
        if ($\mathbf{v} = \infty$) {     /* too many solutions */
            $B \leftarrow$ successor($B$); continue;
        }
            /* exactly one solution $\mathbf{v}$ */
        $B \leftarrow$ next_non_superset($B$);
        if ($B$ = FAIL) ⟨ destroy $\mathcal{B}_{(\sigma,A)}$ ⟩;     /* this pair is dying */
        else ⟨ update $B$ and the hit list in $\mathcal{B}_{(\sigma,A)}$ ⟩;
        return ($\mathbf{v}$);
    }
    ⟨ destroy $\mathcal{B}_{(\sigma,A)}$ ⟩;     /* this pair is dead */
    return (FAIL);
}
```

Assume, then, that *find_link* has returned a barycentric direction $\mathbf{v}$; our next task is to find the exiting face Z and the barycentric coordinates $\mathbf{b}_{\text{out}}$ of the exiting point. Just as in section 3, $\mathbf{b}_{\text{out}}$ is of the form $\mathbf{b}_{\mathbf{v}}(\tau) = \mathbf{b}_{\text{in}} + \tau\mathbf{v}$, where

$$\tau = \min\{-b^i_{\text{in}}/v^i : v^i \neq 0, -b^i_{\text{in}}/v^i > 0\}.$$

There is, however, a difference: the minimum may be achieved for several values of i. In fact, since $b^i_{\text{out}} = 0$ whenever $\tau = -b^i_{\text{in}}/v^i$, the exiting face is given exactly by $Z = \{i : \tau = -b^i_{\text{in}}/v^i\}$; its codimension is q, by definition. It is a good idea to declare equal all values of $-b^i_{\text{in}}/v^i$ that are very close to the minimum, because of numerical error: this makes the exiting face lower-dimensional, and, as we shall see below, it is safer to make $q > 1$ and search for the next simplex than to assume $q = 1$ and exit through the wrong facet.

We remark that $\mathbf{b}_{\text{out}}$ is the unique point of Φ_Z on the contour, so it is safe to make Z and $\mathbf{b}_{\text{out}}$ the last two components of the new state (cf. the paragraph preceding this subsection). To complete the new state, we must move on to another simplex, as described in the next subsection; but first we look back and check if (σ, Z) is in the table of visited pairs, adding it if necessary. This pair is dead if $q = 1$ and alive if $q > 1$, in which case we initialize or update its hit list with the intersection $A \cap Z$.

6.2. Finding the Next Simplex

When the exiting face has codimension $q = 1$, the contour must perforce enter the unique simplex that shares this face with σ, and this new simplex is easily found with the reflection rules. The situation is more complex when $q > 1$, because then there are many n-simplices sharing the exiting face, and *a priori* we don't know which one to go to next. The *star* of a simplex Φ in a triangulation is the union of all simplices that have Φ as a face; our task is to find what simplex (or simplices) in the star of the exiting face Φ_Z, in addition to σ, intersects C.

One candidate is most likely: the simplex σ_{opp} *opposite to* σ (with respect to Z). This opposition is defined by extending through Φ_Z the q facets of σ that contain Φ_Z; more precisely, σ_{opp} is the only n-simplex (other than σ) having Φ_Z as a face and such that, for all $i \in Z$, facet i of σ_{opp} and facet i of σ lie in the same hyperplane. Notice that σ_{opp} may not be the reflection of σ through Φ_Z in the geometric sense, because this reflection is not always a simplex in the triangulation (think of the 60°-vertices in figure 3(c)); if it is, it coincides with σ_{opp}.

As seen from a point in Φ_Z, the set of directions that point into σ_{opp} is opposite the set of directions that point into σ, which means that when the direction of C changes little at Φ_Z chances are good that C goes into σ_{opp}. But this is by no means always the case, because, in principle, f can take any value at the remaining vertices of σ_{opp}. For a concrete example, consider $f = (x^2 + y^2)/n^2$ on $\mathbf{R}^2$, for n an integer, and the R_3 triangulation whose vertices form the lattice $\mathbf{Z}^2 \subset \mathbf{R}^2$. The lattice point $(0, n)$ (together with $(n, 0)$, $(0, -n)$ and $(-n, 0)$) lies in C, and it is easily seen that the two triangles which intersect C and have $(0, n)$ as a vertex are not opposite with respect to $(0, n)$.

This situation tends to occur when, as in this example, the intersection C^σ is very close to one of the facets of σ. It also occurs more often as n or q increases, simply because the measure of the set of directions that point into σ_{opp} from a point in a codimension-q face decreases as a fraction of the measure of the total set of directions.

Our problem, then, is the following: as we're exiting a simplex σ through a face of codimension ≥ 1, we want to find σ_{opp} quickly, and, if it turns out that $C^{\sigma_{\text{opp}}}$ is empty, we want to examine as few as possible of the other simplices containing the face before we find one where the contour continues.

To do this we will again consider Coxeter triangulations. Readers who skipped the discussion in section 4 are encouraged to go back and read it now, or they can skip this one too, going down to the paragraph containing formula $(*)$.

Imagine applying to σ all the symmetries of the triangulation that leave the face Φ_Z fixed. These symmetries evidently form a group G, which is finite because its elements are in one-to-one correspondence with the simplices in the star of Φ_Z. Furthermore, G is generated by reflections in the q facets of σ that contain Φ_Z. The identity 1 of G represents σ, and some element $\neq 1$ of G of order two, which we call -1, represents σ_{opp}.

One extreme case of this procedure occurs when $q = 1$, and then the group reduces to $\{1, -1\}$. The other extreme is $q = n$, that is, Φ_Z is a vertex: then G can be seen as a group acting by reflections on an $(n-1)$-dimensional sphere around Φ_Z, and determining there a *spherical triangulation*. Spherical reflection groups and spherical triangulations can be assigned Coxeter diagrams, following the same conventions as their Euclidean counterparts. By the very nature of this construction, the Coxeter diagram for the group that leaves fixed a vertex $s_i = \Phi_{\{0,\ldots,i-1,i+1,\ldots,n\}}$ of σ can be obtained by removing from the Coxeter diagram of σ the node that represents facet i, plus any edges that end at that node. Observe that this can yield a disconnected diagram; in this case the face group G is a direct product of smaller groups corresponding to the pieces of the diagram, because the absence of an edge between two nodes means that the angle between the respective facets is $\pi/2$, so reflections in these two facets commute. A list of connected spherical Coxeter diagrams can be found on page 297 of [Coxeter 1973].

Something similar occurs for all other values of q. The Coxeter diagram of G can be obtained by starting with the Coxeter diagram of σ and, for each vertex s_i in the fixed face Φ_Z, or, equivalently, each $i \notin Z$, deleting from the diagram the node that represents facet i, together with any edges that end at that node. There remain one or more pieces of the original diagram, each of which is the Coxeter diagram for some spherical reflection group, and whose direct product is G. (In particular, when $q = 1$, the Coxeter diagram of G is a single node, and $G = \{1, -1\}$ is indeed the reflection group of the zero-dimensional sphere.)

This allows us to state explicit reflection rules for finding the simplex σ_{opp} opposite to σ with respect to an arbitrary face Φ_Z of σ. We will examine in detail the case of P_m, for $m = n + 1$ arbitrary. The Coxeter diagram for this triangulation (figure 2) is an m-gon, all of whose edges have weight 1. By removing some vertices from this diagram, we are left with several simple chains, but right now we assume that there is only one, with k nodes, whose indices may be taken to be $1, 2, \ldots, k$.

By formula $(*)$ in section 4, the reflection r_i in facet i of a P_m simplex takes s_i into $s_{i-1} + s_{i+1} - s_i$ (remember our convention about indices being taken modulo m) and leaves the remaining vertices fixed. Another way to describe this rule is by saying that the differences $s_i - s_{i-1}$ and $s_{i+1} - s_i$ get permuted, while the others remain unchanged. If we now consider the group G generated by the reflections r_i, for $i = 1, 2, \ldots, k$, we see that it must be the group of permutations $\mathfrak{S}_{k+1}$ on $k + 1$

objects, because two generators r_i, r_j commute if $|i-j|>1$ and satisfy $r_i r_j r_i = r_j r_i r_j$ if $|i-j|=1$; these are the same relations satisfied by transpositions. (This reasoning would fail if we were considering all the reflections r_i for $i=0,\ldots,m$, because r_0 and r_m don't commute, while the first and last transposition of a set of $m+1$ elements do.)

Furthermore, it is easy to see that the permutation that corresponds to σ_{opp} is the *inversion*, which replaces the first difference, $s_1 - s_0$, by the last, $s_{k+1} - s_k$, the second by the last-but-one, and so on. In other words, to obtain σ_{opp} we should replace s_i by $s_0 + s_{k+1} - s_i$, for every $i = 1, \ldots, k$.

When the Coxeter diagram of G has more than one piece, G is the direct product of several permutation groups, and we just repeat this procedure for each factor. More precisely, we have the following rule to find the opposite of a simplex σ with respect with a face Φ_Z in a P_m triangulation:

$$(*) \qquad \text{for } i \in Z, \text{ replace } s_i \text{ by } s_a + s_b - s_{a+b-i},$$

where a is the first index in the sequence $i-1, i-2, \ldots$ that does not belong to Z, and b the first such index in the sequence $i+1, i+2, \ldots$.

Having discussed how to obtain σ_{opp}, we turn now to the harder problem of what to do when the continuation of the contour does not lie in σ_{opp}. One approach is to inspect every n-simplex in the star of Φ_Z; but this can be onerous—there are $m!$ simplices to inspect if Φ_Z is a vertex of a P_m, for example—and generally unrewarding, because C is likely to continue through only one of them. For this reason we will propose an alternative method for finding at least one link leaving Φ_Z.

(But first, a word of caution. If we insist on tracking all of the 1-skeleton of C, we do have to examine the star of Φ_Z exhaustively—even if C intersects σ_{opp}. This is because C can actually intersect any number of simplices in the star, and there is no way to tell by looking at some of them what C will do in the others: for example, think of f defined in one simplex σ in such a way that C^σ is a vertex plus an interior line segment, then extend f to the star of this vertex by applying to σ the group of symmetries that fix the vertex.

Thus, in addition to keeping a list of active pairs (σ, A), one could keep a list of active node faces: all exiting faces of codimension $q>1$ whose stars have not been completely examined yet. One could define a procedure *find_simplex*(Φ), similar to *find_link*, that would go through all the n-simplices in the star of such a face Φ, in some order, and return all those that intersect C, one at a time. Obviously, it would be advantageous to have things organized in such a way that, given a current node face Φ, all pairs (σ, A) such that $\Phi = \Phi_A$ in simplex σ are easily accessible: by consulting the hit lists of such pairs one could avoid calling *solve* on faces that had been examined previously. Better yet would be to merge *find_simplex* and *find_link* into one procedure that examines all the simplices, of any dimension, that have Φ as a face.)

Here's the alternative method. The *corona* of a simplex Φ in a triangulation is the union of all simplices that are faces of simplices in the star of Φ, but do not have Φ as a face: in some sense, the simplices "opposite" Φ. (The word "link" is sometimes used for this concept, but here it would be terribly confusing.) Thus, if Φ contains a node of C, the links of C starting at this node end at points on the corona of Φ. In particular, the point $\mathbf{b}_{\mathrm{in}}$ where we got into the current simplex σ and from which we drew a link to the exiting face Φ_Z lies on the corona of that face. Since Φ_Z is where we got into trouble, we will try to skirt it altogether by navigating on its corona.

To do this, we relax momentarily the condition that the last component of f vanishes, and follow the mini-contour defined by the first $n-2$ components of f on the corona of Φ_Z, starting at $\mathbf{b}_{\mathrm{in}}$. If all goes well, somewhere along this mini-contour we will find another zero for the last component of f. This is a point where C exits the star of Φ_Z, and we can resume our previous contour tracing, leaving Φ_Z behind. Conceptually, then, we're reducing the dimension of the problem, by solving for the contour of a function having $n-2$ components and defined on an $(n-1)$-dimensional space, the corona of Φ_Z.

This may sound complicated, but algorithmically it is not, because we can use the whole machinery developed so far with very little change. First observe that a point in the star of Φ_Z is on the corona if and only if its k-th barycentric coordinate is zero, for some $k \notin Z$. Thus tracing the contour of the first $n-2$ components of f on the corona of Φ_Z is more or less like tracing the contour of $f_{(1)} = (f^1, \ldots, f^{n-2}, b^k)$ in $\mathbf{R}^n$, where $f^1, \ldots, f^{n-2}$ are the components of f and b^k is the function that gives the k-th barycentric coordinate of a point with respect to the current simplex. However, k may change during this process.

We start by stashing away the current state, and adapting a copy of it to be the current state for $f_{(1)}$. The adaptation is needed because a current face for $f_{(1)}$ must have codimension at least two, that is, its label A_1 must have at least two elements. If A happens to have more than one element, we can copy the old state $(\sigma, A, \mathbf{b}_{\mathrm{in}})$ without change; otherwise, we make the current simplex σ_1 equal to σ, but we take for A_1 an appropriate superset of A, and choose the current point accordingly on Φ_{A_1}, just as we did at the end of 5.2 to initialize the original state. Then we set the auxiliary index k to an element of A_1 that is not in Z.

Next we call *find_link*(σ_1, A_1), with $f_{(1)}$ replacing f in the definition of F_σ; and we move in the direction returned by *find_link* until we hit an exiting face Φ_{Z_1}, which again must have codimension at least two. Assume for now that the exiting face has codimension exactly two, say $Z_1 = \{k, k'\}$, so that we have described a link in the interior of facet k of σ_1. Where will the next link be? The answer depends on whether or not k' belongs to Z: if it does, going beyond Φ_{Z_1} on the corona of Φ_Z takes us to facet k of a new n-simplex σ_1', the reflection of σ_1 in its facet k'; but if $k' \notin Z$, we'll just be taken to facet k' of the same simplex σ_1. In sum, to update the state we do this:

$$\begin{aligned} &\sigma_1 \leftarrow \text{reflection of } \sigma_1 \text{ in face } k', && \text{if } k' \in Z; \\ &k \leftarrow k', && \text{if } k' \notin Z. \end{aligned}$$

The reflection rules, of course, are the same ones discussed in section 3 (formulas $(*)$ and $(**)$, for example).

Before we move on to the next simplex, however, we check whether the last component f^{n-2} of f has changed sign along the link just described. If it has, we have found a simplex σ_1 containing the continuation of the contour of f. We can conclude our detour by fetching the current state that we had saved and modifying it to reflect the new situation: the current simplex becomes σ_1, the current face becomes Z and the current point becomes $\mathbf{b}_{\text{out}}$. The temporary current state that we used to describe our progress along the contour of $f_{(1)}$ can be discarded. In the event it seems we needn't have saved the current state at the beginning of the detour, but there are weird situations in which a loop in the the contour of $f_{(1)}$ does not contain other points where $f^{n-2} = 0$, so we have to try other loops beginning at the entry point $\mathbf{b}_{\text{in}}$.

A little while ago we assumed that the exiting faces Φ_{Z_1} we meet in our detour have codimension two. What if they don't? It will come as no surprise that we may need to make a detour from our detour, having first saved the auxiliary current state right next to the original one: in other words, we need a stack. (Here it doesn't make sense to try the simplex opposite σ_1.) We have sunk to yet another level of degeneracy. The details of this recursion will be left to the reader.

7. Examples

We give two sets of examples of the algorithm in action, in dimensions 2 and 4. The first illustrates robustness in the presence of high curvature of the contour, and the second in the tracing of contours of non-regular values. In each case the R triangulation was used with the P reflection rules, as outlined in section 4.

Consider the complex function $q_c : \mathbf{C} \to \mathbf{C}$ defined by $q_c(z) = z^2 + c$, where c is a fixed complex number. Fix a positive integer $k > 0$ and define $f : \mathbf{C} \to \mathbf{R}$ by $f(z) = \left|q_c^k(z)\right|$, where q_c^r is the r-th iterate of q_c, that is, $q_c^1(z) = q_c(z)$ and $q_c^{r+1}(z) = q_c(q_c^r(z))$ for $r \geq 1$. If we now consider f as a function from $\mathbf{R}^2$ into $\mathbf{R}$, it becomes a candidate for the contour tracing procedure. For the two values of c used in the examples below, any sufficiently large real number is a regular value of f. In addition, the corresponding contour is a single closed curve.

This example is interesting because if we compute a sequence of contours for a fixed value of f, for increasing values of k, they will converge to the so-called *Julia set* of q_c [Blanchard 1984], which is a fractal. Thus, as k increases, the curvature of the contour increases without bound, and the contour would be very difficult to trace with an infinitesimal method. This behavior is not surprising since f is the absolute value of a polynomial of degree 2^k.

Figure 6 shows $f^{-1}(9)$ for $k = 15$ and for two values of c. Notice that in Figure 6(a) the contour comes very close to itself in many places. If the resolution of the underlying triangulation is not high enough it is possible for the contour tracer to make a "mistake" at one of these points and cross over to the other part of the contour. But when the tracer comes back to this point again it makes the opposite mistake, and carries on to the end of the contour. This can be seen (in two dimensions) by making a few sketches to enumerate the possibilities.

Figure 6 about here

In our second set of examples, we are interested in values of c and z for which z is a fixed point of q_c^k, that is, $q_c^k(z) = z$, and for which the derivative of q_c^k evaluated at z has absolute value 1. So we define $g : \mathbf{C} \times \mathbf{C} \to \mathbf{C} \times \mathbf{R}$ to be $g(c,z) = \big(q_c^k(z) - z, |(q_c^k)'(z)|\big)$; thinking of g as a function from $\mathbf{R}^4$ into $\mathbf{R}^3$ our desired points now form the contour $g^{-1}(0,0,1)$. Unless $k = 1$, the value $(0,0,1)$ is not regular.

The interest in the contours $g^{-1}(0,0,1)$ lies in that their union for all $k \geq 1$ is related to the *Mandelbrot set* M [Blanchard 1984]; more exactly, the projection of this union onto the c-coordinate plane is the boundary of the interior of M. This, admittedly, is not a good method to draw M, because $g^{-1}(0,0,1)$ is not connected for $k > 2$, so in addition to tracing the contours we must worry about starting points. Still, it is instructive to watch the algorithm drawing the largest component of $g^{-1}(0,0,1)$, which contains the point $g(\frac{1}{4}, 0, \frac{1}{2}, 0) = (0,0,1)$ for all k: see figure 7, where the projections of this largest component onto each of the two coordinate planes are shown for several choices of k.

For example, the contour tracer begins at $(\frac{1}{4}, 0)$ in Figure 7(c) and at $(\frac{1}{2}, 0)$ in Figure 7(d). As it sweeps out the top half of the cardioid in (c), it traverses the top half of the circle in (d). It then traverses the small circle in (c) twice as it makes the lemniscate in (d) and ends by sweeping out the bottoms of the cardioid and circle. At the point $(-\frac{1}{4}, 0, -\frac{3}{4}, 0)$ the contour intersects itself and the derivative of g is 0. This degeneracy is reflected in the fuzziness of the computed contour in the (d). In 7(f) there are multiple points of self-intersection, and the computed contours are even more fuzzy at some of those points.

Figure 7 about here

8. Extensions

We have mentioned several ways in which the ideas presented here can be extended, or applied to more general problems with slight changes. First of all, we can immediately generalize the algorithm to deal with functions $f : \mathbf{R}^n \to \mathbf{R}^k$, for $k < n - 1$: this is like dealing with degenerate cases of functions $f : \mathbf{R}^n \to \mathbf{R}^{n-1}$. However, some refinements might be desirable, because we can no longer view the issue of efficiency in searching for links when $r > 1$ or $q > 1$ as peripheral, as we tacitly did in section 6, under the excuse that degeneracies were relatively uncommon. Furthermore, the method as stated will only give a wire-frame description of the contour, and we might want to output also the higher-dimensional cells. (This generalization, however, is not difficult.)

We touched on the possibility of dynamically changing the triangulation, in order to approximate regions of high curvature more closely. One idea is to subdivide our simplices into 2^n subsimplices in regions where greater resolution is desirable. Unfortunately, this destroys the simplicity of our reflection rules, and increases the amount of bookkeeping necessary to recognize when we have been somewhere before (for instance, we must ensure that the same fineness of triangulation is used when we return to a previously visited region).

A more exciting possibility is to use a so-called *quasiregular* triangulation. These arise by orthogonal projection of a regular lattice in Euclidean space onto some linear subspace (see [Duneau and Katz 1986]). The projection of the entire lattice is generally dense in the subspace, but if the subspace is chosen to be invariant under some subgroup of the symmetry group of the lattice, and if only the lattice points in a tubular neighborhood of the subspace are projected, the result is a quasiregular lattice in the subspace: the Delaunay triangulation of this lattice consists of an infinite number of copies of a finite collection of congruent triangles. Furthermore, as the thickness of the tubular neighborhood is increased, only triangles that are similar to ones in the original collection are introduced. We can obtain a variable density triangulation by using a tubular neighborhood with variable, rather than constant, thickness: greater thickness means more lattice points would be projected, yielding a denser triangulation. The thickness would be controlled by the desired resolution. The basic drawback with this approach is that the computations involved in passing from one simplex to the next are a good deal trickier.

Finally, there is the matter of finding starting points. There is much literature on this subject, and here we limit ourselves to briefly describing the method used by *plotline*, a program by one of the authors that implements the ideas described here. (A presentation of this program is in preparation.) This method is simple and blends well with the rest of the algorithm; it does not intend to be optimally efficient. The idea is to restrict the search space first to one dimension, until we find a zero of some component of f; then to two dimensions, where we trace the contour of this component until finding a zero of another one; and so on until we find a point where all the components of f vanish. Notice that this is very similar to what we did in 6.2, where we traced the contour of some components of f on a subset of $\mathbf{R}^n$ of appropriate dimension.

9. Bibliography

E. L. Allgower and K. Georg, "Simplicial and continuation methods for approximating fixed points and solutions to systems of equations," *SIAM Review* **22** (1980), 28–85.

E. L. Allgower and P. H. Schmidt, "An algorithm for piecewise linear approximation of an implicitly defined manifold," *SIAM Journal of Numerical Analysis* **22** (1985), 322–346.

D. S. Arnon, "Topologically reliable display of algebraic curves," *ACM SIGGRAPH Computer Graphics* **17** (1983), 219–227.

P. Blanchard, "Complex analytic dynamics on the Riemann sphere," *Bulletin of the American Mathematical Society* **11** (1984), 85–141.

J. H. Conway and N. J. A. Sloane, "Fast quantizing and decoding algorithms for lattice quantizers and codes," *IEEE Transactions on Information Theory* **28** (1982), 227–232.

H. S. M. Coxeter, "Discrete groups generated by reflections," *Annals of Mathematics* **6** (1934), 13–29.

H. S. M. Coxeter, *Regular Polytopes*, third edition, Dover Publications, New York, 1973.

J. J. Dongarra, J. R. Bunch, C. B. Moler, G. W. Stewart, *LINPACK Users' Guide*, SIAM, Philadelphia, 1979.

M. Duneau and A. Katz, "Quasiperiodic patterns and icosahedral symmetry," *J. Physique* **47** (1986), 181–196.

J. D. Foley and A. van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, Mass., 1982.

A. Geisow, "Surface Interrogations," Ph.D. Thesis, University of East Anglia.

S. Karamardian, ed., *Fixed Points: Algorithms and Applications*, Academic Press, 1977.

A. Requicha and H. Voelcker, "Solid Modeling: Current Status and Research Directions," *IEEE Computer Graphics and Applications* (1983), 25–37.

M. A. Sabin, "Contouring—the State of the Art," in *Fundamental Algorithms for Computer Graphics*, edited by R. A. Earnshaw, Springer-Verlag, New York, 1985.

H. Scarf, "The approximationn of fixed points of a continuous mapping," *SIAM Journal of Applied Mathematics* **15** (1967), 1328–1343.

E. Sperner, "Neuer Beweis fur die Invarianz der Dimensionszahl und des Gebietes," *Abh. a. d. Math. Sem. D. Univ. Hamburg* **6** (1928), 265–272.

M. J. Todd, *The Computation of Fixed Points with Applications*, Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, New York, 1976.

P. A. Tukey and J. W. Tukey, "Data-driven view selection; agglomeration and sharpening," in *Interpreting Multivariate Data*, edited by V. Barnett, Wiley, 1981.

Figure 1

Two approaches to the problem of tracing contours. In (a) derivative information is used to take a step, while in (b) local values of the function are used to guide the tracing.

Figure 2

Coxeter diagrams for all simplices in all dimensions that triangulate by reflection.

Figure 3

Portions of the three possible triangulations by reflection of $\mathbf{R}^2$: (a) P_m; (b) R_m; (c) V_m

Figure 4

Finding the barycentric coordinates of the projection of a vertex onto the opposite face.

Figure 5

Two triangulations using R_3; (a) shows triangulation by reflection and (b) just replicates the squares, which is equivalent to using P_3 reflection rules on an R_3 triangle.

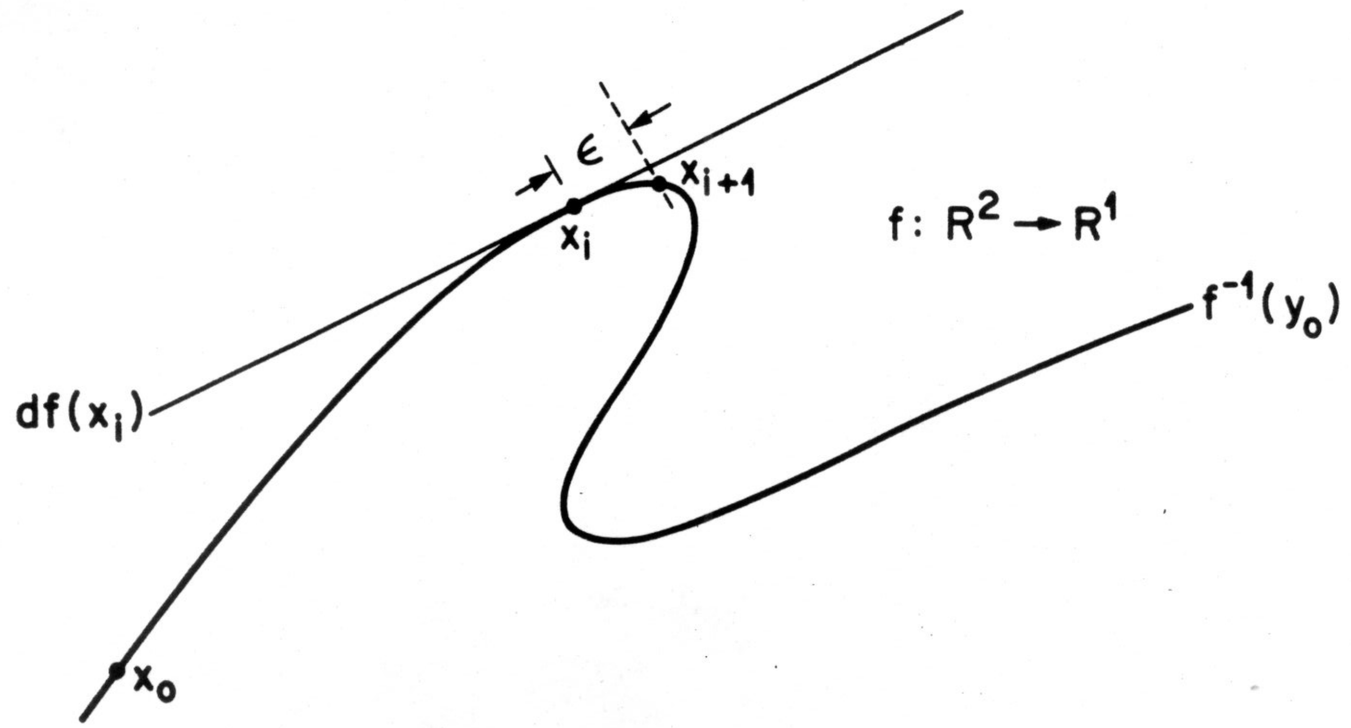
Figure 6

Two contours that approximate Julia sets. The parameter values are approximately (a) $c = -0.156546 + 1.03226i$ and (b) $c = -0.122561 + 0.744862i$. The boxes are 3 units wide in each direction. The R_3 triangulation was used, with step size (short sides of each triangle) equal to .0002.

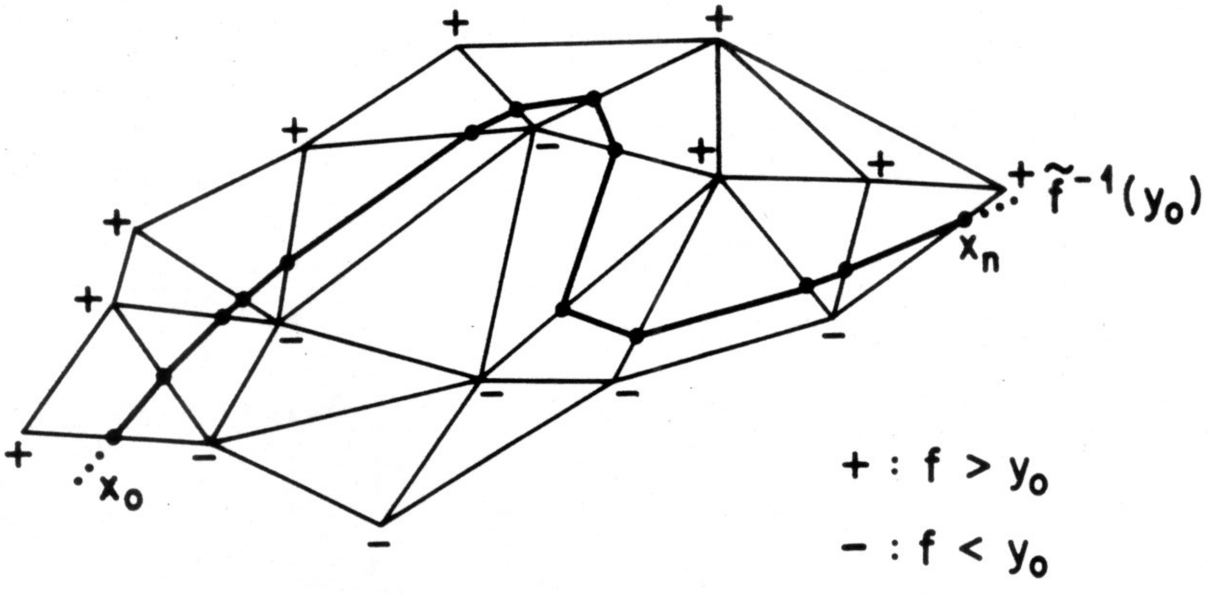
Figure 7

Six contours associated with the Mandelbrot set. The values of k are 1 for (a) and (b), 2 for (c) and (d) and 6 for (e) and (f). The first of each pair is the projection of the contour onto c-space and the second is the projection onto z-space. Each box is 2.5 units wide and 2 units high. The R_5 triangulation was used, with step size (short edges of each simplex) equal to .002 in (a) through (d) and .00025 in (e) and (f).

Figure 1

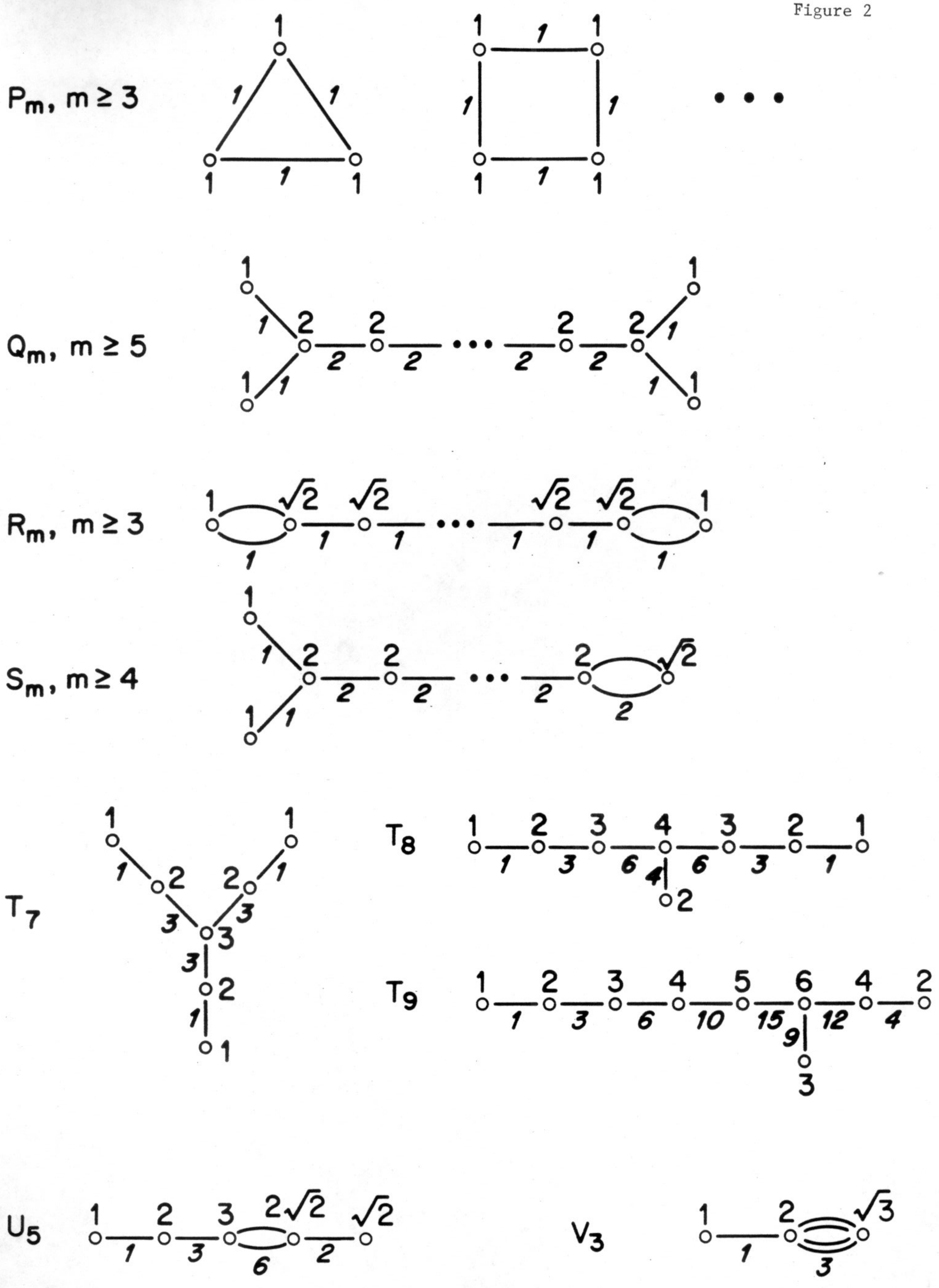

ϵ
x_{i+1}
x_i
$f: R^2 \rightarrow R^1$
$f^{-1}(y_0)$
$df(x_i)$
x_0

(a)


$\tilde{f}^{-1}(y_0)$
x_n
x_0
$+ : f > y_0$
$- : f < y_0$

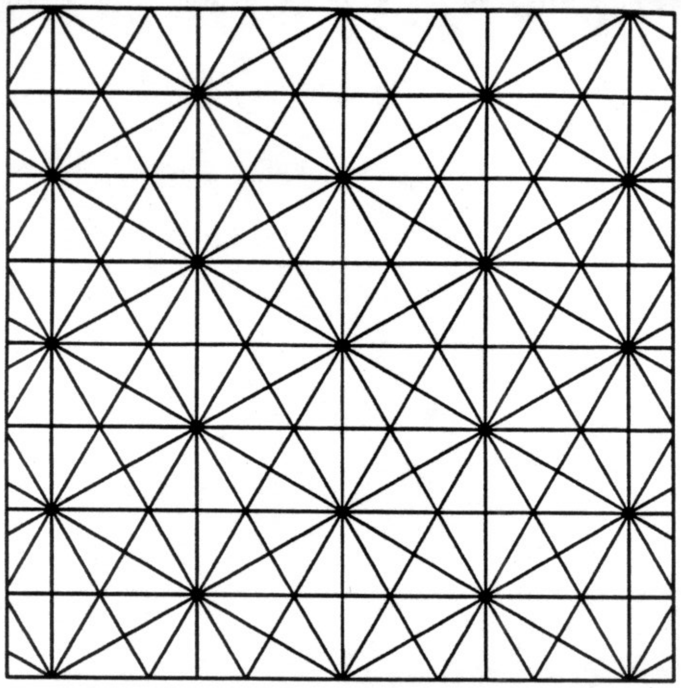
(b)

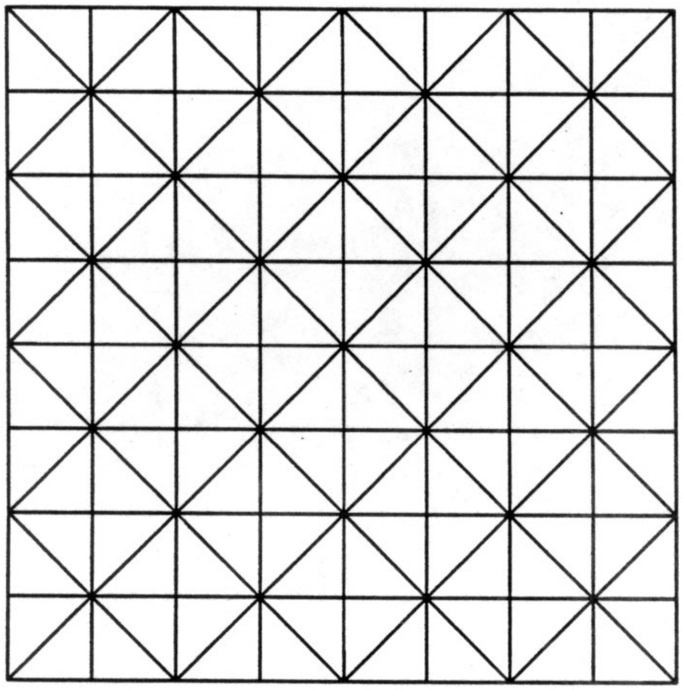
Figure 2



P_m, $m \geq 3$
Q_m, $m \geq 5$
R_m, $m \geq 3$
S_m, $m \geq 4$
T_7
T_8
T_9
U_5
V_3
$\sqrt{2}$
$2\sqrt{2}$
$\sqrt{3}$

Figure 3


(c)


(b)

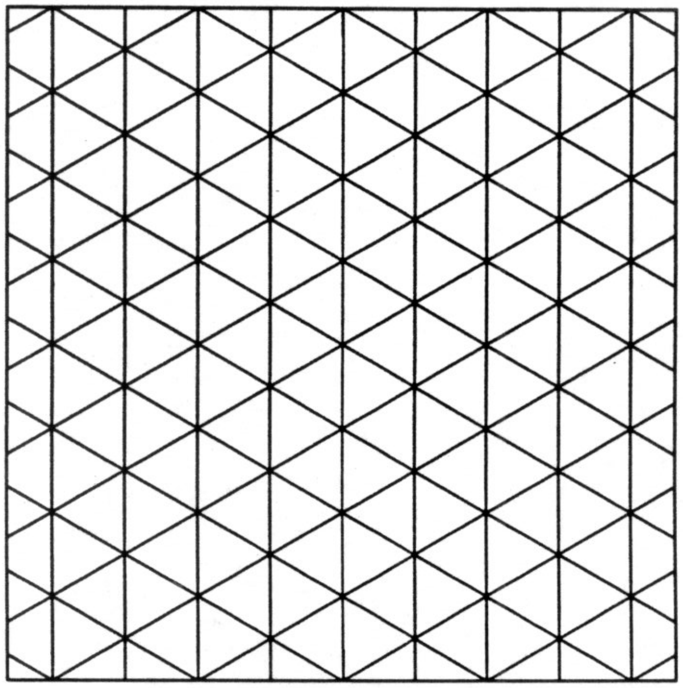

(a)

Figure 4

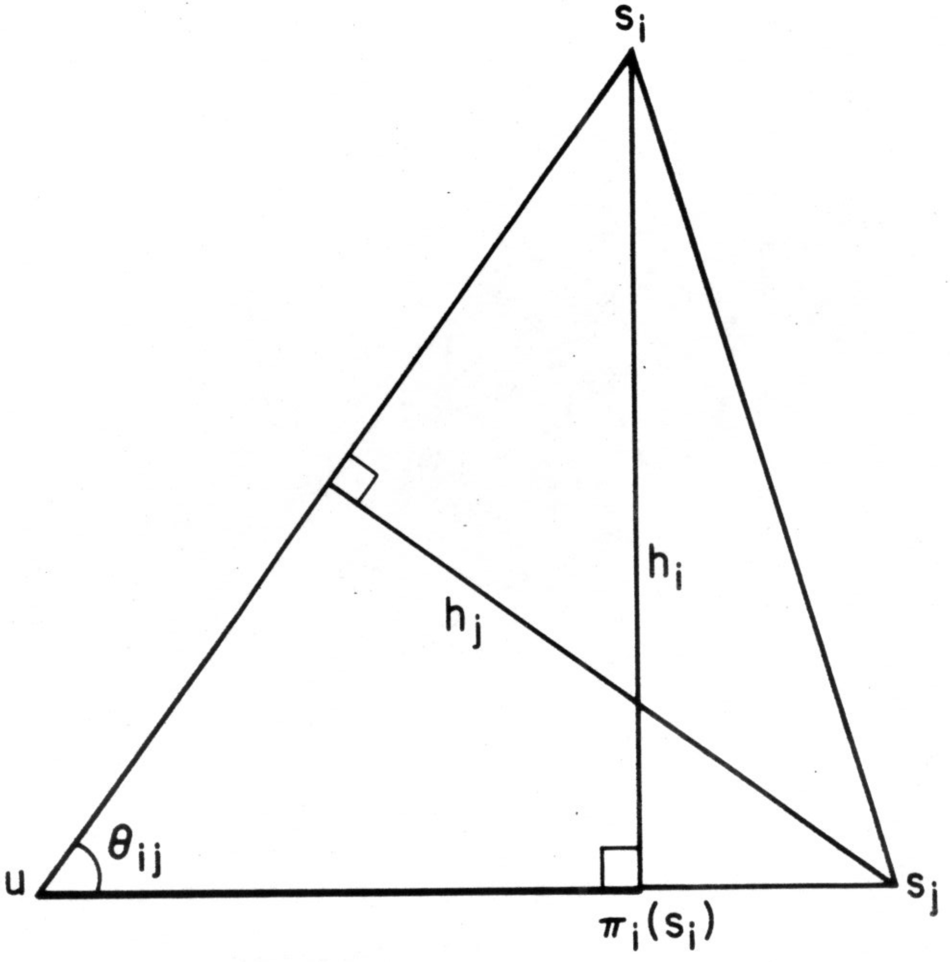
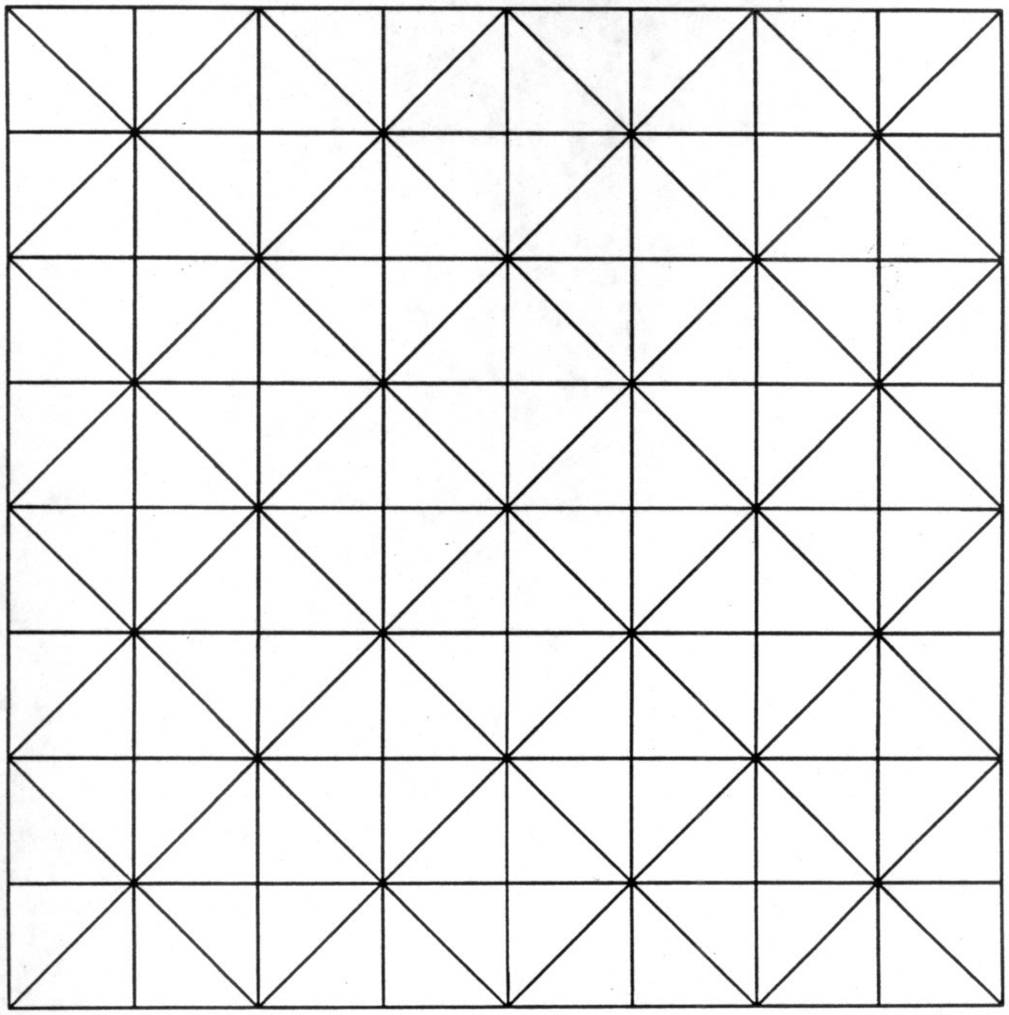


Figure 5


(a)

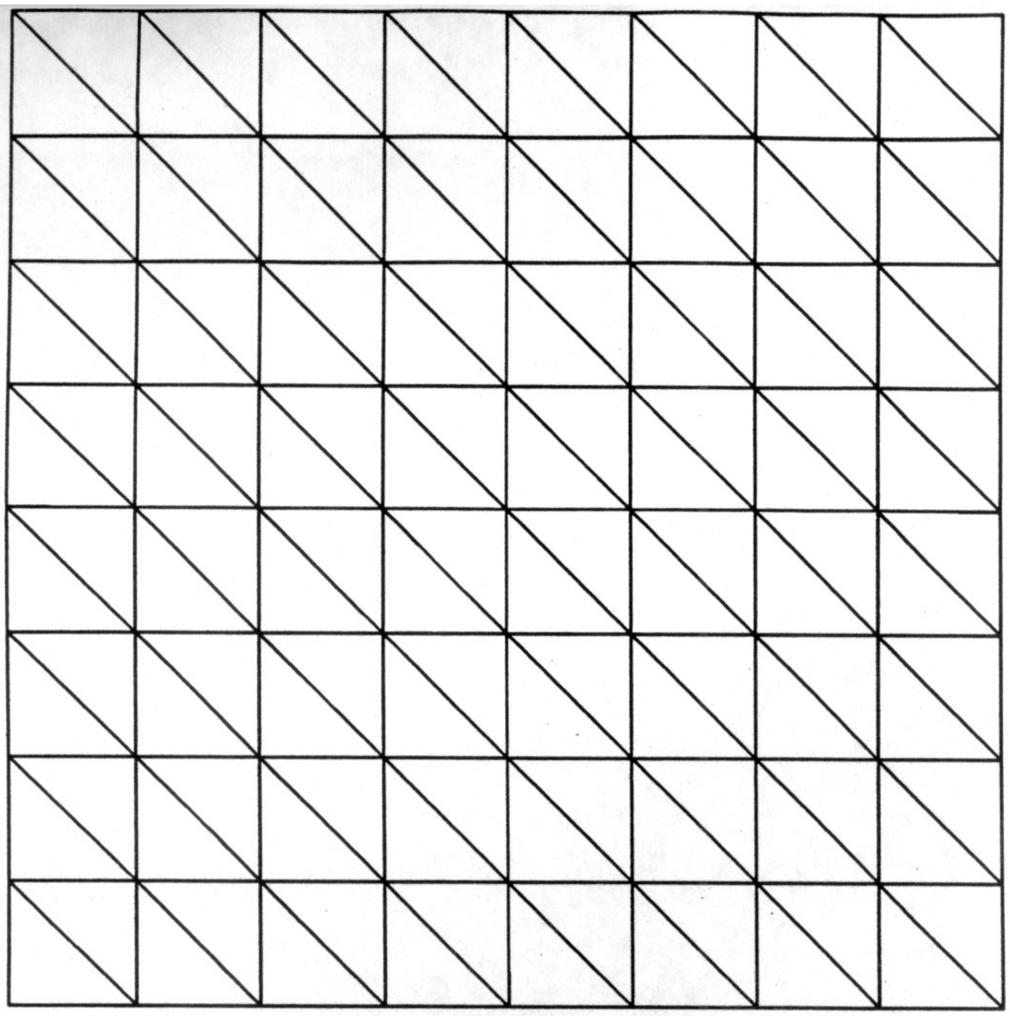
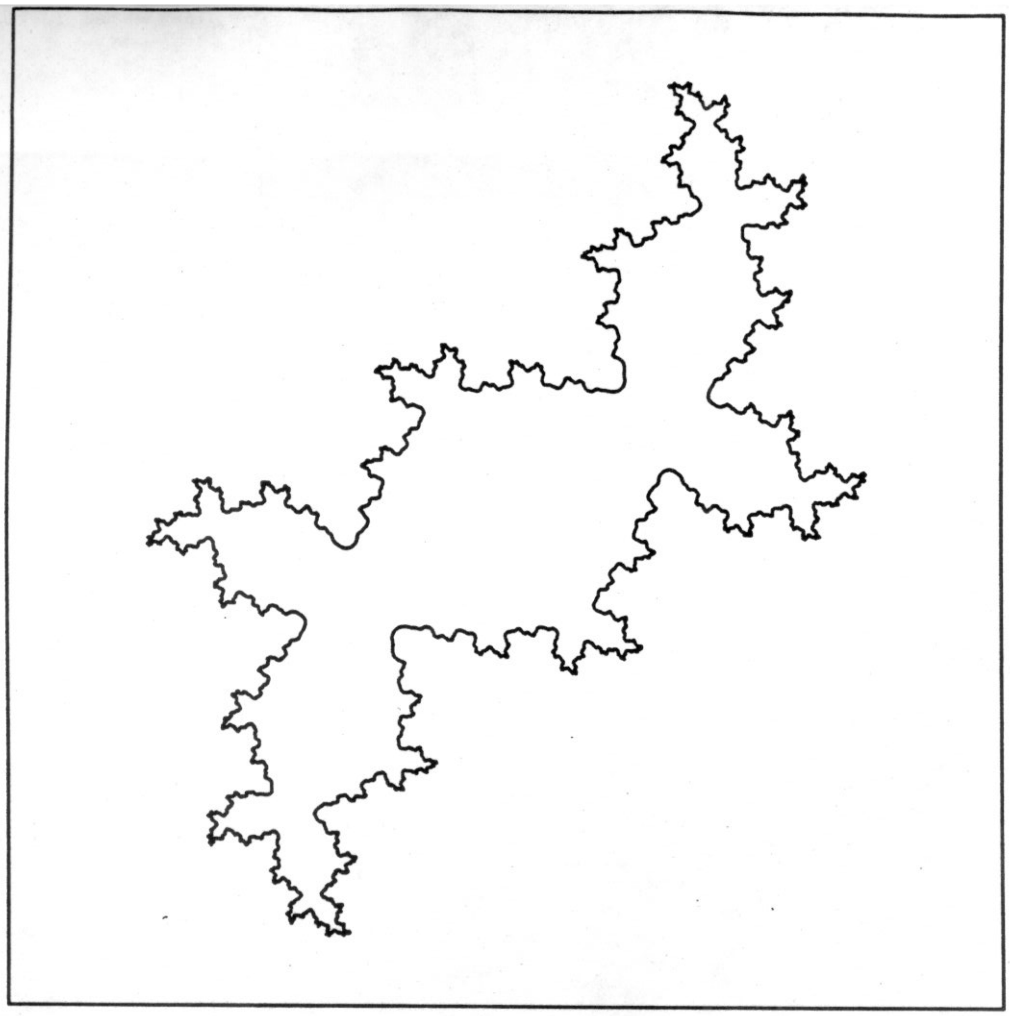

(b)

Figure 6


(b)

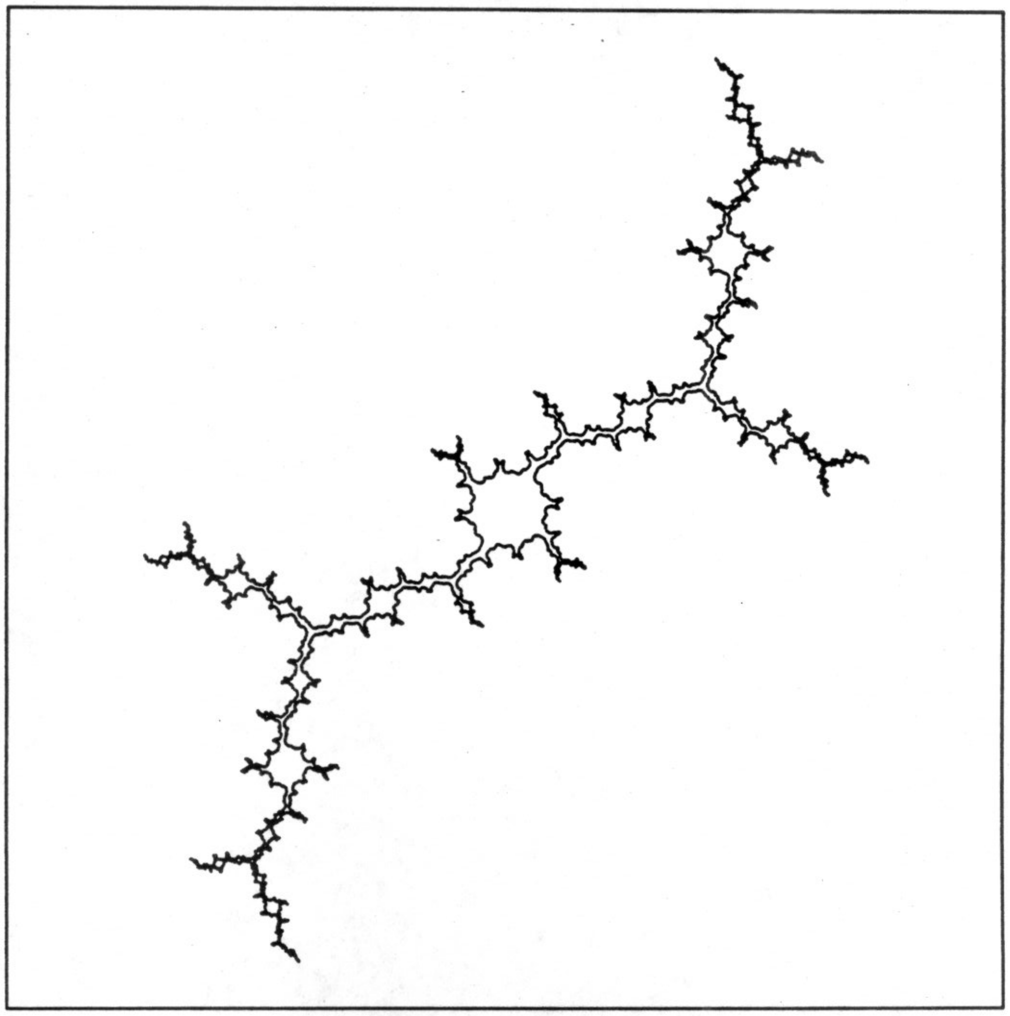

(a)

Figure 7