Pergamon

0098-3004(94)00058-1

Computers & Geosciences Vol. 21, No. 1, pp. 31–49, 1995

0098-3004/95 $9.50 + 0.00

C PROGRAM FOR AUTOMATIC CONTOURING OF SPHERICAL ORIENTATION DATA USING A MODIFIED KAMB METHOD

FREDERICK W. VOLLMER

Department of Geological Sciences, State University of New York at New Paltz, New Paltz, NY 12561, U.S.A.
e-mail: vollmerf@npvm.newpaltz.edu

(Received 13 July 1992; accepted 6 October 1992; revised 13 July 1994)

Abstract—Kamb's method for contouring density diagrams is a simple technique for the preliminary analysis and comparison of orientation data distributions. The method is based on the departure from a uniform distribution, and, unlike the Schmidt method, the dependence of contours on sample size is limited. Several improvements can be made, particularly with regard to the implementation of automatic contouring. To reduce smoothing, the expected count for a random sample drawn from a uniform distribution can be decreased. This gives more localized density estimates that can improve the resolution of features. Density estimates are done directly on the sphere for accuracy. This also permits contouring on stereographic and other nonequal area projections, and accommodates vectorial data. Weighting functions provide better density estimates and increase the smoothness of contour lines. These concepts are implemented in the C program Sphere Contour. Options include selection of data rotation, linear or planar data, equal area or stereographic projection, upper or lower hemisphere, and scatter diagrams. Graphics output is either to the screen (MS-DOS) or to a computer-aided drafting file (AutoCAD DXF). The program is modified easily for other computer systems and graphics devices.

Key Words: Axes, C language, Computer-aided drafting, Density diagrams, Graphics, Spherical projections, Vectors.

INTRODUCTION

Spherical orientation data, including directed lines (unit vectors), nondirected lines (unit axes), and poles to planes (unit axes or vectors) are abundant in structural geology, geophysics, and other fields. They are plotted typically as scatter diagrams on equal area projections of a unit hemisphere and contoured to produce density diagrams. In the Schmidt and related methods, a counting circle comprising 1% of the total area is used to make density estimates at the nodes of a regular grid, and the grid is contoured in units of percent density per 1% area (Turner and Weiss, 1963, p. 58–67; Ragan, 1985, p. 274–279; Marshak and Mitra, 1988, p. 148–157).

Although widely used, the Schmidt and other 1% area methods are not well suited for comparing data sets because the contours are dependent strongly on sample size (Flinn, 1958; Kamb, 1959; Dudley, Perkins, and Gine, 1975; Starkey, 1976; Schaeben, 1982). This is significant particularly for geological field studies, where sample sizes may range widely. Kamb (1959) proposed an alternative in which contours represent standard deviations away from the expected density for a random sample drawn from a uniform distribution. Kamb's method reduces the effect of sample size on contours, allowing comparison of data sets with different sample sizes.

Dudley, Perkins, and Gine (1975) and Schaeben (1982) provide reviews of fabric diagram construction and related statistical methods. Other discussions of Kamb's method include Starkey (1976), Cheeney (1983, p. 108–110), Robin and Jowett (1986), and Jowett and Robin (1988). Various published and unpublished computer implementations of Kamb's method exist (e.g. Tocher, 1978, 1979; Griffis, Gustafson, and Adams, 1985; Robin and Jowett, 1986; Allmendinger and others, 1991; Van Everdingen, Van Gool, and Vissers, 1992). Computer implementations of other types of density diagrams are numerous; early examples include those by Robinson, Robinson, and Garland (1963), Spencer and Clabaugh (1967), Starkey (1969), and Warner (1969). Ramsden and Cruden (1979), Schaeben (1982, 1986), Diggle and Fisher (1985), and Fisher, Lewis, and Embleton (1987, p. 41–46) describe alternate methods involving the estimation of the probability density function of a data distribution.

This paper first reviews Kamb's method and a number of modifications which have been adopted for automatic contouring. These include changing the expected count for a uniform distribution, estimating point density on the sphere, modification for directed data, and using weighting methods for smoothing. Example data sets are used to illustrate these modifications, and to compare qualitatively variations of

the Kamb method, the Schmidt method, and the method of Diggle and Fisher (1985). These ideas are used in the C program Sphere Contour which is implemented on an IBM PC microcomputer, and can export diagrams into AutoCAD. It is written in ANSI C for easy porting to other systems.

KAMB'S METHOD

Kamb (1959) proposed a method for producing density diagrams based on binomial statistics. A binomial random variable has mean, μ, and standard deviation, σ

$$\mu = np \tag{1}$$

$$\sigma = [np(1-p)]^{1/2} \tag{2}$$

where n is the number of trials and p is the probability of success for a single trial (Hoel, 1971, p. 58–63). If n points are selected randomly from a uniform population distributed over an area A, then the probability that any given point will lie within an arbitrary subarea, a, of A will be

$$p = a/A. \tag{3}$$

The number of points occurring within area a then can be treated as a binomial random variable with an expected count, E, equal to the mean, μ. For a circular area A with radius R, and a counting circle of area a and radius r, Equation (3) can be rewritten as

$$p = r^2/R^2. \tag{4}$$

Kamb (1959) selected a binomial probability model with

$$E = \mu = 3\sigma \tag{5}$$

so that, given a random sample from a uniform population, the counting circle would be large enough so the observed counts would not be likely to fluctuate wildly from the expected count. Combining Equations (1), (2), and (5) gives

$$p = 9/(n+9) \tag{6}$$

and substituting Equation (4) in (6) gives the desired radius of the counting circle

$$r = 3R/(n+9)^{1/2}. \tag{7}$$

For manual contouring, the centers of two counting circles are joined by a line of length $2R$, and overlain on an equal area projection so the line remains on the center of the projection. Counting is done on a regular grid. A grid spacing of r usually is used (e.g. Marshak and Mitra, 1988), but this can give poor results for small sample sizes. A regular grid of about 21 × 21 nodes probably is more appropriate for hand contouring.

Contour levels greater than 3σ (E) indicate a density higher than expected for a uniform distribution, and levels less than 3σ indicate a density lower than expected. The 0σ contour, for example, represents densities 3SDs less than expected ($E - 3\sigma$). Contour levels may be set at 2σ, but can be set at multiples of E. It should be cautioned that the assignment of confidence levels to the contours is not straightforward. For example, the $E + 2\sigma$ contour does not imply a 95% confidence level as might be expected from comparison with a normal distribution, a larger departure generally is required for such confidence (Dudley, Perkins, and Gine, 1975). Jowett and Robin (1988) give empirical methods for statistical evaluation of peak and trough densities.

COUNTING ELEMENT SIZE AND SAMPLE SIZE

A primary benefit of Kamb's method is that it reduces the influence of sample size by differing the counting element size [Eq. (7)]. Kamb selected a binomial model with $E = 3\sigma$ to define this relationship. However, other functions are possible. The use of a counting circle with an area $1/n$ times the projection area has been suggested (Flinn, 1958; Starkey, 1976). Procedures for determining an optimal counting element size are available (Ramsden and Cruden, 1979; Schaeben, 1982, 1986; Diggle and Fisher, 1985). Kamb's approach does not provide necessarily such an optimal density estimate. In particular, as shown in the examples, it may over-smooth distributions with strong preferred orientations. Changing the binomial model will alter the amount of smoothing (Dudley, Perkins, and Gine, 1975; Robin and Jowett, 1986). The following modifications therefore are suggested for subjective control of the contouring process.

If the number of standard deviations defining the expected count for a uniform distribution is allowed to differ, so that $E = k\sigma$ where k is the expected count in standard deviations, then Equations (6) and (7) become

$$p = k^2/(n+k^2) \tag{8}$$

$$r = kR/(n+k^2)^{1/2}. \tag{9}$$

Note that, for models where $E = 3\sigma$, $E = 2\sigma$, and $E = 1\sigma$, a count of 0 will differ from the expected count by 3, 2, or 1 SDs, an increasingly smaller departure. Selecting a lower k gives a smaller counting element, more localized density estimates, and less smoothing.

Figure 1 illustrates the effect of sample size on the expected distribution of counts in a random sample from a uniform distribution. These statistics apply only to arbitrary, prospectively selected counting areas (Dudley, Perkins, and Gine, 1975). The Schmidt and related 1% methods set the probability to 0.01 regardless of sample size. This generally is too small, giving, for example, an expected count of 0.50 ± 0.70 for a sample of 50 points [$E \pm \sigma$, Eqs. (1) and (2)]. A sample size of 99 is required before $E = 1\sigma$, 396 for $E = 2\sigma$, and 891 for $E = 3\sigma$. In contrast, the standard Kamb method gives an expected count of 7.63 ± 2.54 (Fig. 1).

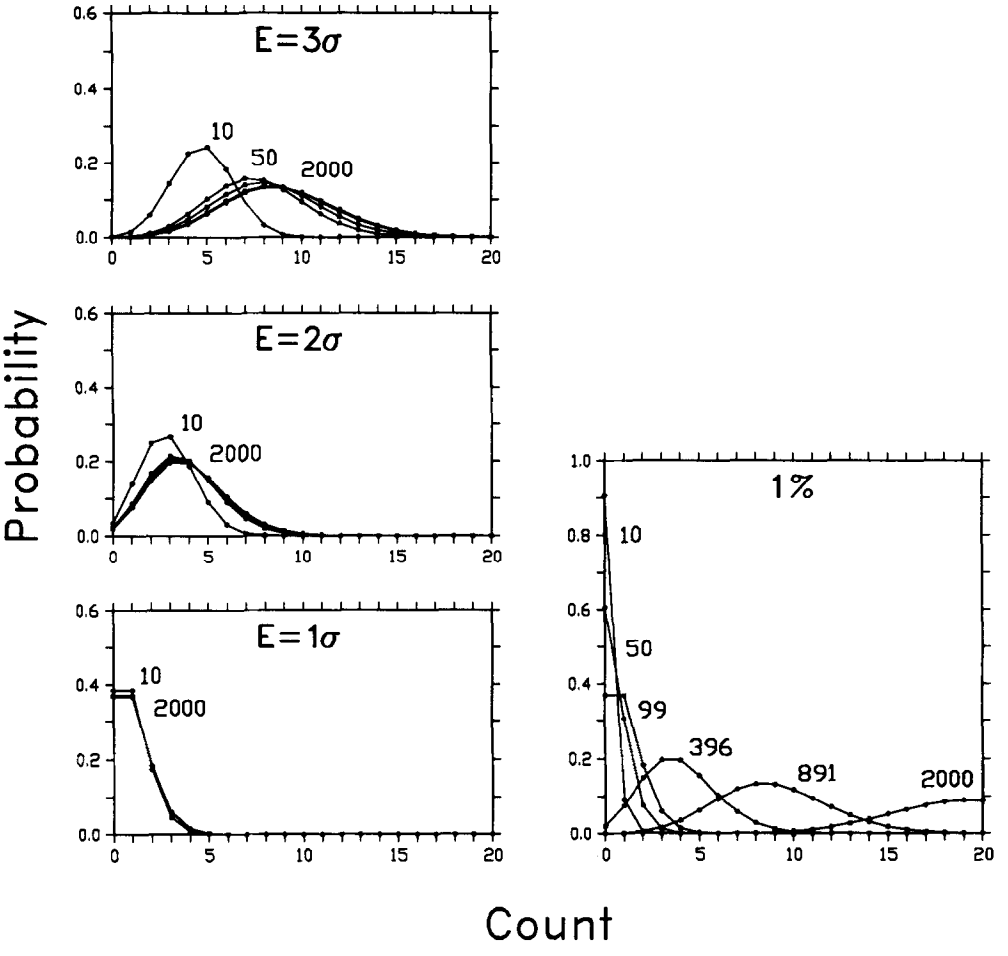

Figure 1. Probability distributions illustrating effect of sample size on probability of obtaining different counts for random sample of uniform distribution. E is expected count for Kamb method. 1% model represents Schmidt method. Six curves in each diagram are for sample sizes: 10, 50, 99, 396, 891, and 2000.

As illustrated in the examples, for data sets that clearly deviate from a uniform distribution, models with $E = 2\sigma$ or $E = 1\sigma$ may improve the resolution of features, while limiting the effect of sample size. For weak fabrics a higher degree of smoothing may be obtained by increasing the expected density.

COUNTING ON THE SPHERE

When estimating density it is necessary to determine the number of points that lie within a counting element on the surface of a sphere. This can be approximated by using counting circles on a projection, but counting directly on the sphere is more accurate and computationally simpler (Warner, 1969). The area of a spherical cap on a unit radius sphere is

$$a = 2\pi(1 - \cos\theta) \tag{10}$$

where θ is the semiapical angle of the cone defining the cap. For directed data distributed on a unit sphere of area 4π this gives

$$p = a/A = (1 - \cos\theta)/2 \tag{11}$$

$$\cos\theta = (n - k^2)/(n + k^2) \tag{12}$$

and for axial data distributed on a unit hemisphere of area 2π

$$p = a/A = 1 - \cos\theta \tag{13}$$

$$\cos\theta = n/(n + k^2). \tag{14}$$

Similar equations are given by Robin and Jowett (1986).

For any desired location on the sphere, the number of data points that fall within the angular distance θ can be counted by computing dot products, taking the absolute values for axial data. The complexity and inaccuracy of the double circle algorithm used in hand contouring are unnecessary.

WEIGHTING FUNCTIONS

A simple tally of points within counting elements results in marginally satisfactory contours. Decreasing the grid spacing helps to bring out additional maxima by increasing the resolution of the contours, but tends to create jagged contours. The problem is

that the density estimate at a point must be made over a finite area. In standard counting each data point within an element is assigned a weight, w, of 1, whereas a weight of zero is given to all points outside. This is the step function

$$w = 1 \quad (x \leqslant a) \tag{15}$$

taken over the area of the spherical cap, a, where x is the area of a cap centered at the same location with the counted point on its perimeter. A simple inverse area weighting function is

$$w = 2(1 - x/a) \quad (x \leqslant a). \tag{16}$$

A higher degree of smoothing is given by the function

$$w = 3(1 - x/a)^2 \quad (x \leqslant a) \tag{17}$$

which gives an inverse area squared weighting (Fig. 2). These are analogous to inverse distance weighting used in map contouring, where the influence of a data point falls off with distance from a node (Davis, 1986, p. 366–368), and act as filter operators for convolution of the data (Kalkani and Von Frese, 1982). Each of these functions has a volume equal to a when integrated over the cap area, so the expected density is not altered. This was confirmed empirically in 30 tests on five geologic data sets with sample sizes from 38 to 957; the average point counts on 25 × 25 grids were 0.99 ± 0.01 of the nonsmoothed grid counts.

An exponential weighting function, based on a spherical Gaussian or Fisher distribution, has been applied to Kamb's method (Robin and Jowett, 1986). It is a model of an ideal unimodal distribution and uses all points over the sphere. In general it will have a greater smoothing effect than Equations (16) and (17) (Fig. 2). This and other smoothing techniques are discussed by Ramsden and Cruden (1979), Kalkani and Von Frese (1980, 1982), Robin and Jowett (1986), and Charlesworth and others (1989).

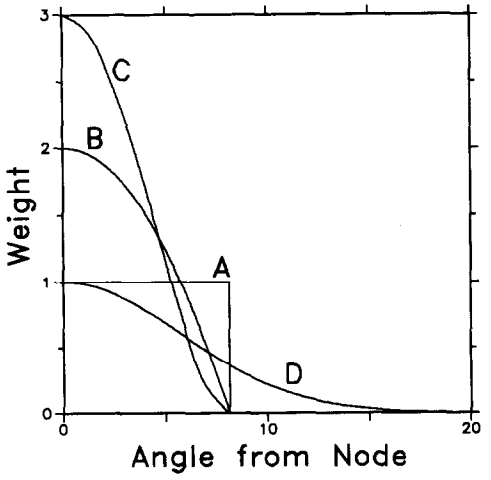

Figure 2. Weighting functions for smoothing density estimates calculated for $p = 0.01$. A—standard step function; B—inverse area; C—inverse area squared; D—exponential. Examples of effect of these functions are shown in Figure 3.

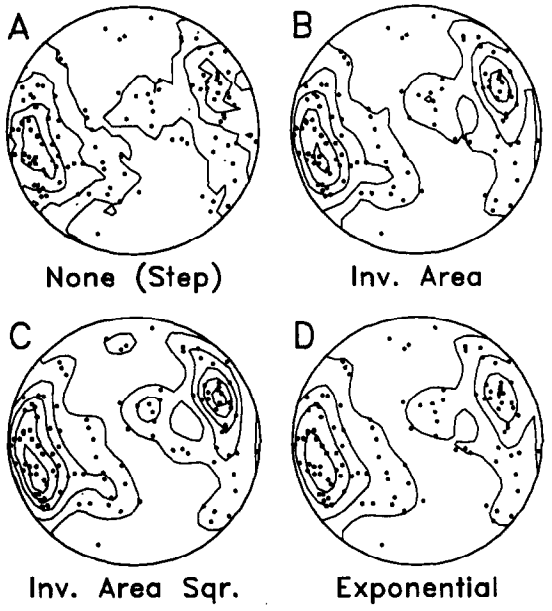

Figure 3. Contoured density diagrams illustrating effect of weighting functions of Figure 2. A—no smoothing (step function); B—inverse area; C—inverse area squared; D—exponential. 112 c-axis orientations in ice (from Kamb, 1959) contoured using Kamb method with $E = 3\sigma$. Contour interval is 2σ beginning at 2σ.

The effect of these functions is illustrated in Figure 3 with a set of 112 c-axes in ice (digitized from fig. 7 of Kamb, 1959; 13 of the 125 points could not be resolved). All diagrams are lower hemisphere equal area projections (except as noted in fig. 8); this data set has up at the top, all others have north at the top.

GRIDDING AND CONTOURING

In the gridding algorithm used here, the nodes of a regular square grid are back-projected onto the sphere. This gives a coverage greater than one full hemisphere allowing contours to be extended to the edge of the projection, where they are clipped. After weighted counts are made at each node the grid is preprocessed for contouring. Because the point count is a noncontinuous variable, and the best estimate of a contour lies halfway between consecutive values, 0.5 is subtracted from each total. For example, a contour line for a density of 0 will pass halfway through two nodes of count values 0 and 1, rather than through the first node. The grid values then are normalized to 1 SD.

The contour lines are drawn by linear interpolation through the grid. Any errors in contour location resulting from interpolation are confined between the nodes; a reasonably fine grid spacing will minimize these errors. Numerous other gridding and contour interpolation methods, including the use of polynomial fitting and alternate grid geometries, have been used (e.g. Kalkani and Von Frese, 1979; Tocher, 1979; Chaio, 1985; Diggle and Fisher, 1985; Yates, 1987; Charlesworth and others, 1989).

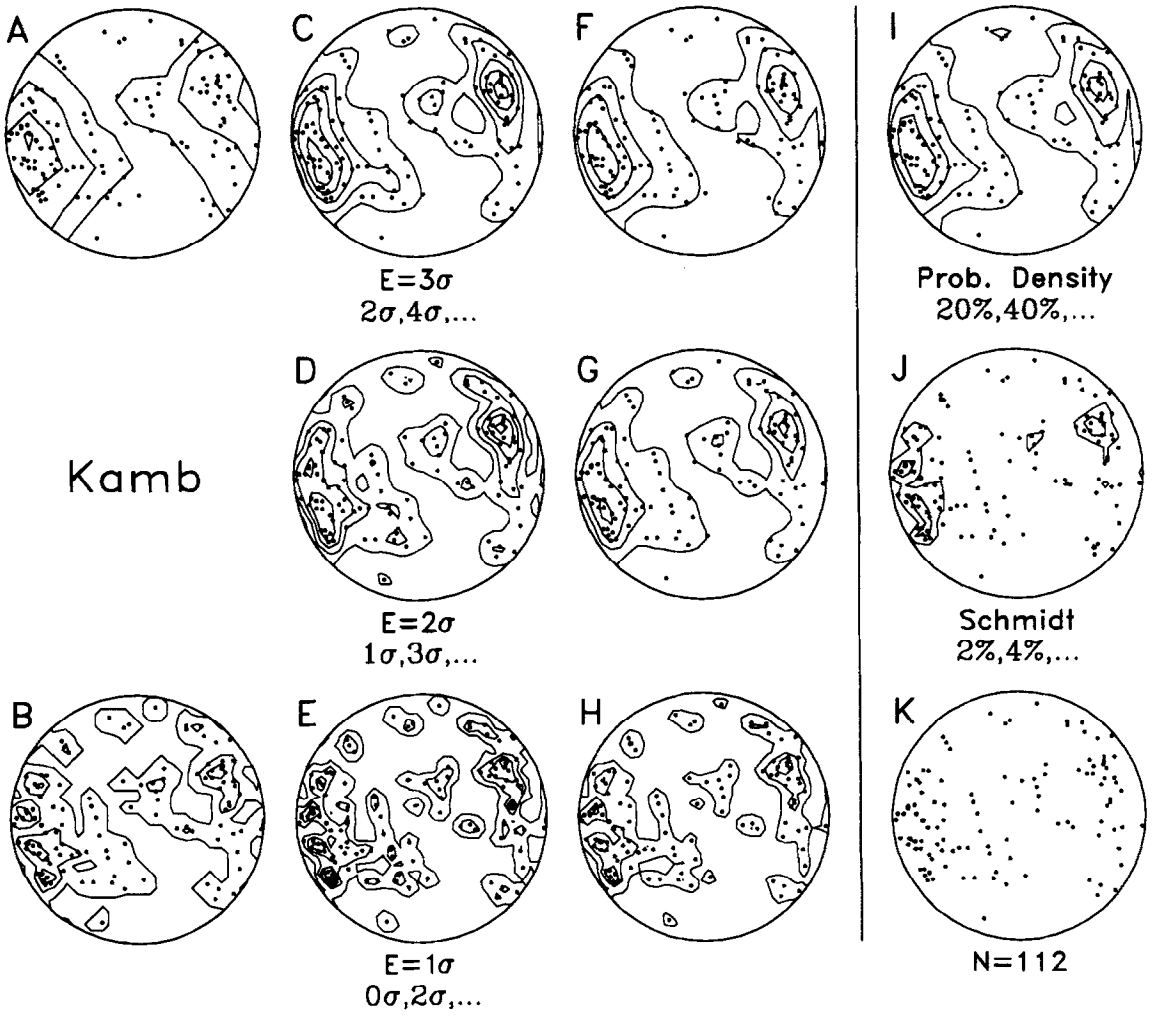

Figure 4. Data (as in Fig. 3) with weak maxima contoured using methods described in text. Diagrams A–H use Kamb contouring with contour intervals of 2σ. A, B—methods analogous to hand contouring; C, D, E—inverse area squared smoothing; F, G, H—exponential smoothing; I—method of Diggle and Fisher (1985), contours in percent of estimated probability density function; J—Schmidt method, contours in percent per 1% area; K—scatter plot of raw data.

EXAMPLES

Three data sets displayed in Figures 4, 5, and 6 illustrate a range of geologic distributions. A triangular fabric plot (Vollmer, 1989, 1990) of the three data sets illustrates the variation of fabric types among the three data sets (Fig. 7). All diagrams, except A, B, and J, were created with a 30×30 grid.

Diagrams A–H were generated using variations of the Kamb method. A and B were generated using double counting circle algorithms on the projection plane with a grid spacing equal to the counting circle radius. These are equivalent to hand-contoured diagrams. C, D, and E use modified Kamb methods with $E = 3\sigma$, $E = 2\sigma$, and $E = 1\sigma$ respectively and inverse area squared weighting. F, G, and H are similar, but with exponential weighting. Note that inverse area squared weighting gives greater resolution and less smoothing, whereas exponential weighting gives smoother, more averaged contours.

Diagram I was generated using the probability density function estimation algorithm of Diggle and Fisher (1985). Estimation of the concentration parameter was done using cross-validation log-likelihood maximization, as the data sets are multimodal and asymmetric. The gridded values are estimates of a probability density function, with contours equally spaced over the range of grid values. J was generated using the Schmidt method and an algorithm similar to that used for diagrams A and B.

Diagrams C–H, and K (also A and B in Fig. 8) were generated using the Sphere Contour program. Diagrams A, B, and J were generated using an early version of the program Orient (by Vollmer, see Allmendinger and others, 1991). Diagram I (also C in Fig. 8) was generated with an unpublished program (by Vollmer) using Diggle and Fisher's (1985) algorithm. Final drafting and layout was done in AutoCAD.

Example 1—weak maxima

The data in Figure 4 (from Kamb, 1959; same data as in Fig. 3) have a large amount of scatter, and broad weak maxima. The characteristics are well-defined using the Kamb method with $E = 3\sigma$. Kamb (1956) categorized this fabric as a single broad maximum as it appears in A (compare with fig. 7 of Kamb, 1959). However, it is clear that two significant maxima are present.

Example 2—bimodal

Figure 5 shows a set of 38 normal fault striations (from Angelier, 1979), contoured as axial data. The unmodified Kamb method, shown in A, fails to bring out the two maxima partially because the small sample size resulted in a large counting circle and a coarse grid, giving poor resolution. For this relatively strong double maxima fabric, a reduction of the expected count more strongly emphasizes finer details of the data set.

Example 3—girdle

Density diagrams of 56 normals to bedding planes from a series of asymmetric folds in Ordovician graywacke (from Vollmer, 1981) are shown contoured as axial data in Figure 6. This data set has a significant girdle pattern with a point maximum. As in the previous example, a reduction of the expected count emphasizes finer details, giving a narrower girdle pattern.

Example 4—directed data

Figure 8 shows lower and upper equal area projections of magnetic remanence measurements from Precambrian volcanics (from Schmidt and Embleton, 1985, as tabulated in Fisher, Lewis, and Embleton, 1987, table B6). These data are unit vectors and therefore are contoured on the sphere rather than the hemisphere. For diagram C the method of Diggle and Fisher (1985) was used, with the concentration parameter (13.62) selected by cross-validated log-likelihood maximization. The contours derived using the

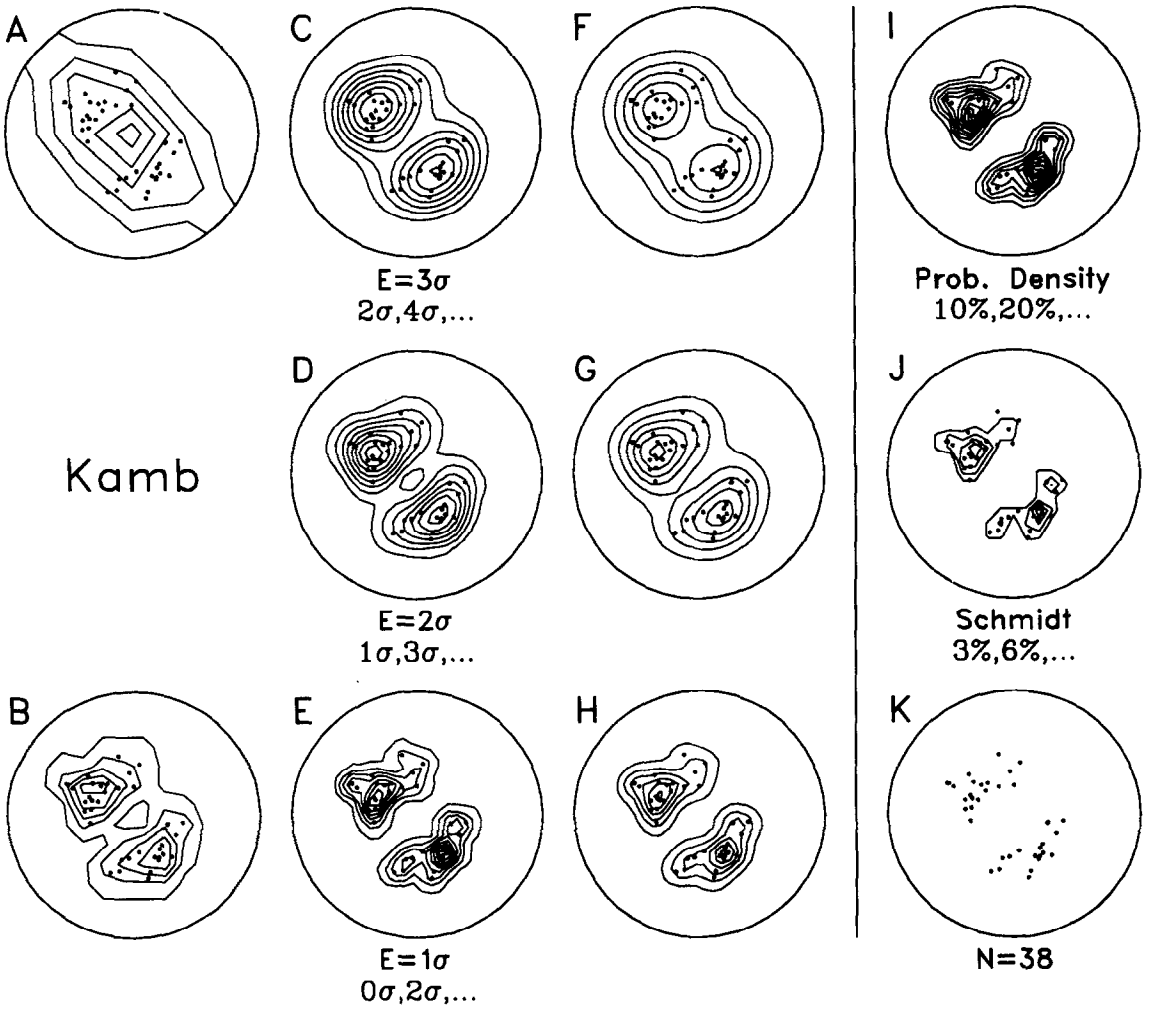

Figure 5. Bimodal data set of 38 normal fault striations (from Angelier, 1979) contoured as axial data using methods as in Figure 4.

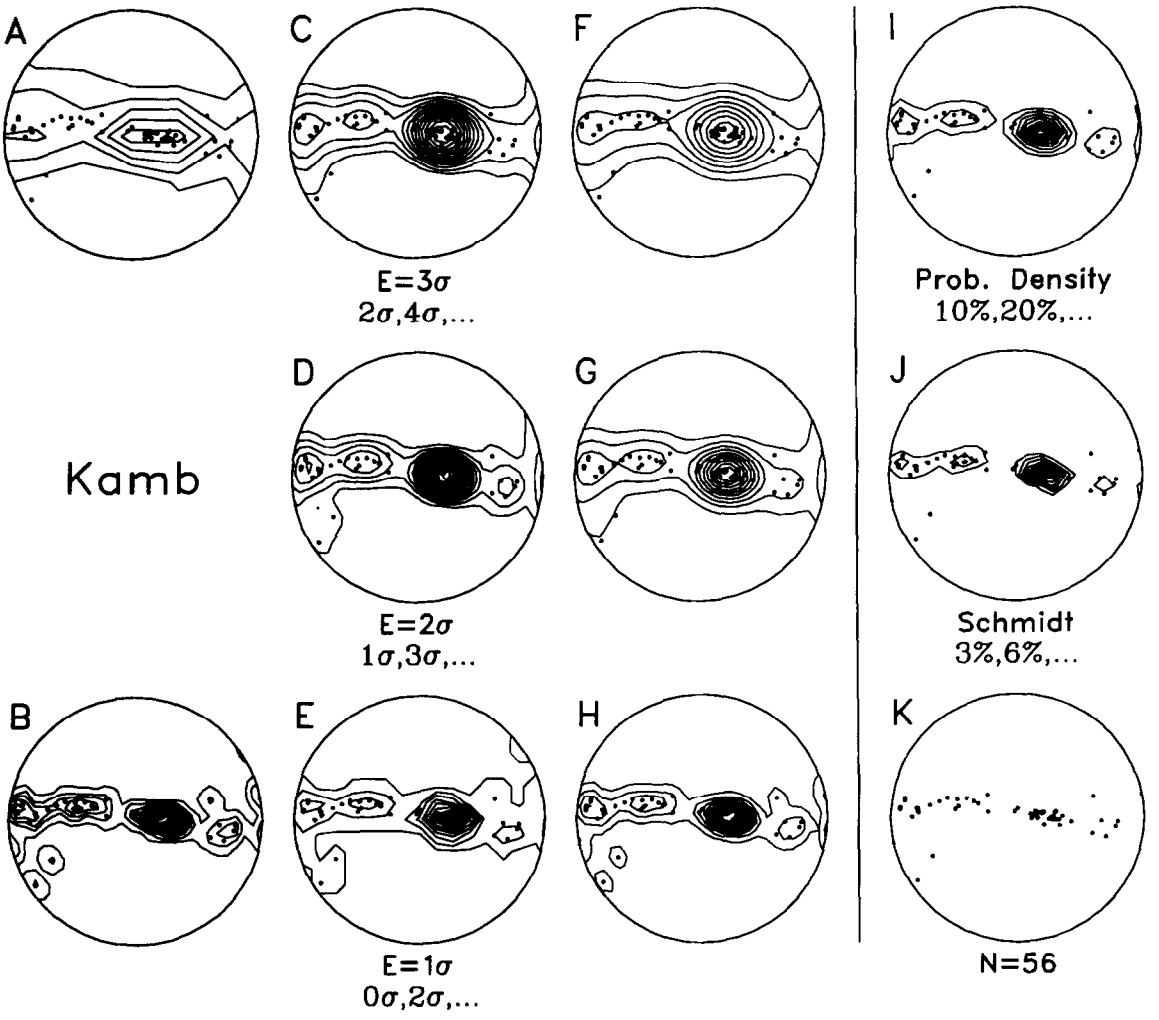

Figure 6. Girdle distribution of 56 poles to bedding planes from series of asymmetric folds (from Vollmer, 1981) contoured as axial data using methods as in Figure 4.

three methods differ in resolution and smoothness, albeit the overall forms are similar.

THE SPHERE CONTOUR PROGRAM

With the Sphere Contour program the user can select options interactively, preview a plot on the screen, and then output a plot to a computer-aided drafting (CAD) file. Selected options can be saved to a configuration file, and additional plots can be generated automatically by entering data file names on the command line. Data are read from a text file in one of four formats: strike (strike, dip, dip octant), dip azimuth (dip, dip azimuth), line (plunge, trend or inclination, declination), or spherical polar (colatitude, longitude). The program will rotate the data up to sixteen times about the coordinate axes.

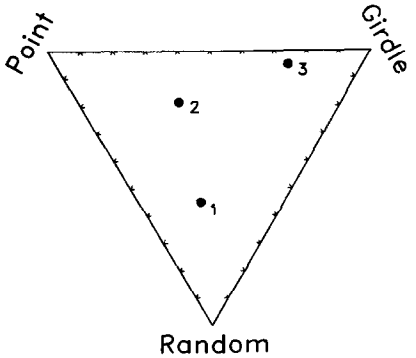

Figure 7. Triangular fabric plot (Vollmer, 1989, 1990) showing range in fabric types used in Examples 1, 2, and 3. Corresponding data sets are plotted in Figures 4–6.

As written, the program uses DOS screen graphics and AutoCAD DXF files. For convenience in drafting, various parts of the drawing are placed on different layers in the DXF file. The file can be imported directly into AutoCAD using the AutoCAD DXFIN command. The device dependent graphics routines are confined to three procedures: InitGraphic, DoneGraphic, and LineOut, making adaptation to other graphics devices and file formats straightforward. Global constants beginning with "dev" control the transformation from millimeters to

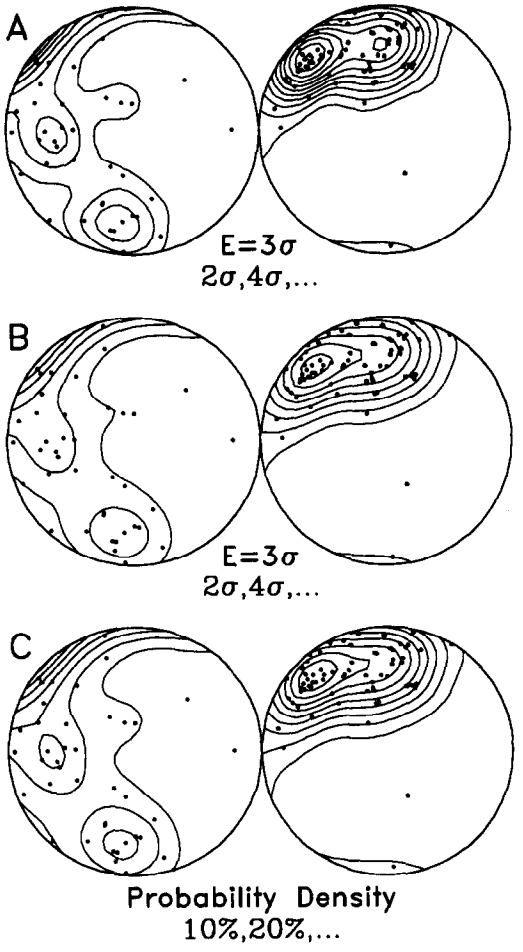

Figure 8. Lower (left) and upper (right) hemisphere equal area projections of 107 measurements of magnetic remanence from Precambrian volcanics (from Schmidt and Embleton, 1985, as tabulated in Fisher, Lewis, and Embleton, 1987, table B6) contoured as vectorial data. A—Kamb method with area squared smoothing, contour interval of 2σ beginning at 2σ; B—as A, with exponential smoothing; C—method of Diggle and Fisher (1985), with contour interval of 10%.

device coordinates. Nonstandard (non-ANSI) C procedures are kept to a minimum, and are commented.

DISCUSSION

Although Kamb's method does not necessarily provide optimal density estimates, it is a simple and useful graphical technique for the preliminary examination of orientation data. With the ability to subjectively control the magnitude of smoothing using the methods described here, it should suffice for many studies. Methods for evaluating the statistical significance of contours are available (Jowett and Robin, 1988). Other more complex techniques for estimating the probability density function of the population (Schaeben, 1982, 1986; Diggle and Fisher, 1985) also should be considered.

From the examples given here, it is clear that a wide range of density diagrams can be generated from any given data set. As noted, many other techniques and contouring programs are available. Even a cursory examination of current geological literature reveals that the methods used to produce density diagrams may be described inadequately. It thus is recommended strongly that the contouring procedures, weighting methods, contour levels, and computer programs be stated clearly for all published diagrams. Whenever feasible, the contour lines should be superimposed upon a scatter plot of the original data, allowing the reader to evaluate visually their significance. To allow comparison of data sets with the results of other studies, contouring methods that are influenced strongly by sample size should be avoided.

Acknowledgments—The author thanks Donal M. Ragan and two anonymous reviews for their comments. Portions of this study were funded by National Science Foundation grant EAR-9003935 to Vollmer.

REFERENCES

Allmendinger, R. W., Charlesworth, H. A. K., Erslev, E. A., Guth, P., Langenberg, C. W., Pecher, A., and Whalley, J. S., 1991, Microcomputer software for structural geologists: Jour. Struct. Geology, v. 13, no. 9, p. 1079–1083.

Angelier, J., 1979, Determination of the mean principal stress directions of stresses for a given fault population: Tectonophysics, v. 56, no. 3–4, p. T17–T26.

Charlesworth, H., Cruden, D., Ramsden, J., and Huang, Q., 1989, ORIENT: an interactive FORTRAN 77 program for processing orientations on a microcomputer: Computers & Geosciences, v. 15, no. 3, p. 275–293.

Chiao, L., 1985, FORTRAN-V program for contouring point density on pi-diagrams using a microcomputer: Computers & Geosciences, v. 11, no. 5, p. 647–657.

Cheeney, R. F., 1983, Statistical methods in geology: George Allen & Unwin, London, 169 p.

Davis, J. C., 1986, Statistics and data analysis in geology (2nd ed.): John Wiley & Sons, New York, 646 p.

Diggle, P. J., and Fisher, N. I., 1985, Sphere: a contouring program for spherical data: Computers & Geosciences, v. 11, no. 6, p. 725–766.

Dudley, R. M., Perkins, P. C., and Gine, M. E., 1975, Statistical tests for preferred orientation: Jour. Geology, v. 83, no. 6. p. 685–705.

Fisher, N. I., Lewis, T., and Embleton, B. J. J., 1987, Statistical analysis of spherical data: Cambridge Univ. Press, Cambridge, 329 p.

Flinn, D., 1958, On tests of significance of preferred orientation in three-dimensional fabric diagrams: Jour. Geology, v. 66, no. 5, p. 526–539.

Griffis, R. A., Gustafson, S. J., and Adams, H. G., 1985, PETFAB: user-considerate FORTRAN 77 program for the generation and statistical evaluation of fabric diagrams: Computers & Geosciences, v. 11, no. 4, p. 369–408.

Hoel, P. G., 1971, Introduction to mathematical statistics (4th ed.): John Wiley & Sons, New York, 409 p.

Jowett, E. C., and Robin, P. F., 1988, Statistical significance of clustered orientation data on the sphere: an empirical derivation: Jour. Geology, v. 96, no. 5, p. 591–599.

Kalkani, E. C., and Von Fresc, R. R. B., 1979, An efficient construction of equal-area fabric diagrams: Computers & Geosciences, v. 5, no. 3, p. 301–311.

Kalkani, E. C., and Von Frese, R. R. B., 1980, Computer construction of equal-angle fabric diagrams and program comparisons: Computers & Geosciences, v. 6, no. 3, p. 279–288.

Kalkani, E. C., and Von Frese, R. R. B., 1982, Convolution of fabric data to determine probability distribution: Jour. Struc. Geology, v. 4, no. 1, p. 93–103.

Kamb, W. B., 1959, Ice petrofabric observations from Blue Glacier, Washington, in relation to theory and experiment: Jour. Geophys. Res., v. 64, no. 11, p. 1891–1909.

Marshak, S., and Mitra, G., 1988, Basic methods of structural geology: Prentice Hall, Englewood Cliffs, New Jersey, 446 p.

Ragan, D. M., 19185, Structural geology, an introduction to geometrical techniques (3rd ed.): John Wiley & Sons, New York, 393 p.

Ramsden, J., and Cruden, D. M., 1979, Estimating densities in contoured orientation diagrams: Geol. Soc. America Bull., v. 90, no. 3, pt. I, p. 229–231; v. 90, no. 3, pt. II, p. 580–607.

Rankin, J. R., 1989, Computer graphics software construction: Prentice Hall, New York, 544 p.

Robin, P. F., and Jowett, E. C., 1986, Computerized density contouring and statistical evaluation of orientation data using counting circles and continuous weighting functions: Tectonophysics, v. 121, no. 1, p. 207–223.

Robinson, P., Robinson, R., and Garland, S. J., 1963, Preparation of beta diagrams in structural geology by a digital computer: Am. Jour. Science, v. 261, no. 10, p. 913–928.

Rodgers, D. F., and Adams, J. A., 1976, Mathematical elements for computer graphics: McGraw–Hill Book Co., New York, 239 p.

Schaeben, H., 1982, Fabric-diagram contour precision and size of counting element related to sample size by approximation theory methods: Jour. Math. Geology, v. 14, no. 3, p. 205–216.

Schaeben, H., 1986, Comment on sphere: a contouring program for spherical data: Computers & Geosciences, v. 12, no. 5, p. 729.

Spencer, A. B., and Clabaugh, P. S., 1967, Computer program for fabric diagrams: Am. Jour. Science, v. 265, no. 2, p. 166–172.

Starkey, J., 1969, A computer programme to prepare orientation diagrams, *in* Paulitsch, P., ed., Experimental and natural rock deformation: Springer-Verlag, New York, p. 51–74.

Starkey, J., 1976, The contouring of orientation data represented in spherical projection: Can. Jour. Earth Science, v. 14, no. 1, p. 268–277.

Tocher, F. E., 1978, Petrofabric point-counting program fabric (FORTRAN IV): Computers & Geosciences, v. 4, no. 1, p. 5–21.

Tocher, F. E., 1979, The computer contouring of fabric diagrams: Computers & Geosciences, v. 5, no. 1, p. 73–126.

Turner, F. J., and Weiss, L. E., 1963, Structural analysis of metamorphic tectonites: McGrawHill Book Co., New York, 545 p.

Van Everdingen, D. A., Van Gool, J. A. M., and Vissers, R. L. M., 1992, Quickplot: a microcomputer-based program for processing of orientation data: Computers & Geosciences, v. 8, no. 2/3, p. 183–287.

Vollmer, F. W., 1981, Structural studies of the Ordovician flysch and melange in Albany County, New York: unpubl. masters thesis, State University of New York at Albany, 151 p.

Vollmer, F. W., 1989, A triangular fabric plot with applications for structural analysis (abstr.): Eos (Am. Geophys. Union Trans.), v. 70, no. 15, p. 463.

Vollmer, F. W., 1990, An application of eigenvalue methods to structural domain analysis: Geol. Soc. America Bull., v. 102, no. 6, p. 786–791.

Warner, J., 1969, FORTRAN IV program for the construction of pi diagrams with the Univac 1108 computer: Kansas Geol. Survey Computer Contr. 33, 38 p.

Yates, S. R., 1987, CONTUR: a FORTRAN algorithm for two-dimensional high-quality contouring: Computers & Geosciences, v. 13, no. 1, p. 61–67.

APPENDIX

Sphere Contour Program Listing

```
/*
*  Program  : Sphere Contour
*  File     : scon.c
*  Purpose  : Contoured density diagrams of spherical orientation data
*  Language : C
*  Compiler : Borland Turbo C++ 1.00
*  Author   : F.W. Vollmer
*  Update   : 6/92, 7/93, 9/93, 5/94
*
*  Portability Notes - Non-ANSI Code
*  ---------------------------------
*  clrscr()        - clears screen in text mode
*  MAXPATH         - maximum characters in DOS filename
*  ChangeFileExt() - changes extension of a DOS filename
*  DoneGraphics()  - shuts down graphics system
*  InitGraphics()  - initializes graphics system
*  LineOut()       - draws line in mm units
*/

#include <conio.h>      /* Turbo C++ Library: Console IO      */
#include <dir.h>        /* Turbo C++ Library: DOS directories */
#include <graphics.h>   /* Turbo C++ Library: BGI graphics    */
#include <ctype.h>      /* ANSI C Libraries...                */
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
char szInfo[] =
  "SPHERE CONTOUR\n"
  "--------------\n"
  "Creates contoured density diagrams of spherical orientation data.\n"
  "Output is to graphics screen or AutoCAD DXF file. Data must be entered\n"
  "into a text file in one of the following formats:\n"
  "\n"
  "  FORMAT    COMPONENTS                                    EXAMPLE\n"
  "  Strike    strike, dip, dip octant                       190 60 E\n"
  "  Dip       dip, dip azimuth                              60 100\n"
  "  Line      plunge, trend (inclination, declination)      30 280\n"
  "  Polar     colatitude, longitude                         120 170\n"
  "\n"
  "Angles are in degrees. Strike, dip azimuth and trend are measured\n"
  "clockwise from Y (north). Longitude is measured anticlockwise from X\n"
  "(east). Dip and plunge are measured downward from the XY plane. Colatitude\n"
  "is measured from Z (up). Each data point must occupy one line, with\n"
  "components separated by spaces or commas. For automatic mode, save desired\n"
  "options, then enter the data file name on the command line.\n"
  "\n"
  "F.W. Vollmer, 1993-1994\n"
  "Department of Geological Sciences, SUNY New Paltz, New York 12561\n"
  "Internet: vollmerf@npvm.newpaltz.edu\n"
  "\n"
  "Press ENTER to continue...";

#define ENTER              '\n' /* enter key                          */
#define DTOR     0.01745329252  /* degrees to radians                 */
#define RTOD     57.2957795131  /* radians to degrees                 */
#define MAXGRID             85  /* maximum number of grid nodes       */
#define MAXSTR              80  /* maximum size of user input string  */
#define WIDTHSTR            48  /* format width for prompt strings    */

#define round(x) (int)floor((x)+0.5)
double sqrarg;
#define sqr(x) ((sqrarg=(x)) == 0.0 ? 0.0 : sqrarg*sqrarg)

enum boolean     {FALSE, TRUE};
enum datatypes   {AXES, VECTORS};
enum devices     {BGI, DXF};
enum formats     {STRIKE, DIP, LINE, POLAR};
enum hemispheres {LOWER, UPPER};
enum methods     {KAMB, SCHMIDT};
enum projections {EQUALAREA, STEREOGRAPHIC};
enum plots       {SCATTER, CONTOUR, BOTH};
enum symbols     {NOSYM, CROSS, TRIANGLE, SQUARE, HEXAGON};
enum smoothing   {NOSMOOTH, INVAREA, INVAREASQR, EXPONENTIAL};

typedef struct {              /* holds user options            */
  double  ci;                 /* contour interval              */
  int     device;             /* BGI or DXF                    */
  int     dataType;           /* AXES or VECTORS               */
  int     format;             /* STRIKE or DIP or PLUNGE       */
  int     hemi;               /* LOWER or UPPER                */
  int     maxdata;            /* maximum number of data points */
  int     method;             /* KAMB or SCHMIDT               */
  double  minimum;            /* minimum contour               */
  double  netX;               /* x coordinate of center        */
  double  netY;               /* y coordinate of center        */
  int     nGrid;              /* number of grid nodes          */
  int     nRot;               /* number of rotations           */
  int     plot;               /* SCATTER, CONTOUR or BOTH      */
  int     proj;               /* EQUALAREA or STEREOGRAPHIC    */
  double  radius;             /* radius                        */
  double  rot[16];            /* rotation angles               */
  int     rotAxis[16];        /* rotation axes, X=0, Y=1, Z=2  */
  double  sigma;              /* binomial sigma value          */
  int     smooth;             /* NOSMOOTH, INVAREA, INVAREASQR */
                              /* or EXPONENTIAL                */
  int     symbol;             /* NOSYM, CROSS, TRIANGLE,       */
                              /* SQUARE or HEXAGON             */
  double  symSize;            /* symbol size in mm             */
  } optiontype;

typedef double point[3];
```

```
/*** Global Variables ***/

optiontype  opt;                    /* holds user options     */
point      *data;                   /* data direction cosines */
int         nData;                  /* number of data points  */
double      grid[MAXGRID][MAXGRID]; /* the grid               */
FILE       *dxf;                    /* DXF text file          */

double dev_xRatio;          /* X device/mm ratio, negative for right origin */
double dev_yRatio;          /* Y device/mm ratio, negative for top origin   */
double dev_xOrigin;         /* left X device coordinate                     */
double dev_yOrigin;         /* bottom Y device coordinate                   */

char data_file[MAXPATH];    /* data file name                               */
char out_file[MAXPATH];     /* DFX file name                                */

/*** User Input ***/

/* GetKey - gets a character from keyboard. */
int GetKey(void) {
  char c;
  c = getchar();
  if (c != '\n') getchar(d);  /* get linefeed */
  return c;
  }

/* GetInt - gets a prompted integer from user. */
void GetInt(char *pmt, int *i) {
  char buf[MAXSTR];
  int  j;
  printf("%-*s %12d: ",WIDTHSTR,pmt,*i);
  fgets(buf,MAXSTR,stdin);
  if (sscanf(buf,"%d",&j) == 1) *i = j;
  }

/* GetDbl - gets a prompted double from user. */
void GetDbl(char *pmt, double *x) {
  char   buf[MAXSTR];
  double y;
  printf("%-*s %12g: ",WIDTHSTR,pmt,*x);
  fgets(buf,MAXSTR,stdin);
  if (sscanf(buf,"%lf",&y) == 1) *x = y;
  }

/* GetStr - gets a prompted string from user. */
int GetStr(char *pmt, char *s) {
  char buf[MAXSTR];
  char t[MAXSTR];
  printf("%-*s %12s: ",WIDTHSTR,pmt,s);
  fgets(buf,MAXSTR,stdin);
  if (sscanf(buf,"%s",t) == 1) { strcpy(s,t); return TRUE; }
  return FALSE;
  }

/* GetChoice - gets a prompted selection from user. Returns number of
/* response in string, 0 is first choice. */
int GetChoice(char *pmt, char *choices, int *i) {
  char *r,c;
  printf("%-*s %12c: ",WIDTHSTR,pmt,choices[*i]);
  c = toupper(GetKey());
  r = strchr(choices,c);          /* points to first occurance */
  if (r) *i = (int)(r-choices);  /* offset into string */
  return *i;
  }

/* ErrorMsg - prints error message, with integer if i > 0. */
void ErrorMsg(char *s, int i) {
  if (i > 0) fprintf(stderr,"%s %d. Press ENTER...",s,i);
  else fprintf(stderr,"%s. Press ENTER...",s);
  GetKey(); /* wait */
  }

/* ChangeFileExt - changes a DOS file name extension. */
void ChangeFileExt(char *fnew, char *fold, char *fext) {
  char drive[MAXDRIVE],dir[MAXDIR],file[MAXFILE],ext[MAXEXT];
  fnsplit(fold,drive,dir,file,ext);
```

```
  fnmerge(fnew,drive,dir,file,fext);
  }

/*** Rotations ***/

/* maxisrot3 - calculates 3D rotation matrix of theta radians about */
/* coordinate axis (0=X, 1=Y, 2=Z).                                 */
void maxisrot3(int axis, double theta, double t[3][3]) {
  int a1,a2,i,j;
  double c,s;
  for (i=0; i<3; i++) for (j=0; j<3; j++) t[i][j] = 0.0;
  t[axis][axis] = 1.0;
  a1 = (axis+1) % 3;              /* x=012 y=120 z=201 */
  a2 = (a1+1) % 3;
  c = cos(theta); s = sin(theta);
  t[a1][a1] = c; t[a2][a2] = c;
  t[a1][a2] = -s; t[a2][a1] = s;
  }

/* mmult - multiplies 3x3 matrix x by y. */
void mmult3(double x[3][3], double y[3][3], double z[3][3]) {
  int i,j,k;
  for (i=0; i<3; i++) for (j=0; j<3; j++) {
    z[i][j] = 0.0;
    for (k=0; k<3; k++) z[i][j] += x[i][k] * y[k][j];
    }
  }

/* GetRotMat - builds the rotation matrix from user data. */
void GetRotMat(double r[3][3]) {
  int i,j,n;
  double s[3][3],t[3][3];
  for (i=0; i<3; i++) { for (j=0; j<3; j++) r[i][j] = 0.0; r[i][i] = 1.0; }
  for (n=0; n<opt.nRot; n++) {
    for (i=0; i<3; i++) for (j=0; j<3; j++) t[i][j] = r[i][j];  /* copy */
    maxisrot3(opt.rotAxis[n],opt.rot[n]*DTOR,s);
    mmult3(s,t,r);
    }
  }

/*** Conversions ***/

/* OctantVal - converts an octant string to degrees. */
int OctantVal(char *s, double *r) {
  int i;
  for (i=0; s[i] != '\0'; i++) toupper(s[i]);
  if (strcmp(s,"N") == 0)       *r =   0.0;
  else if (strcmp(s,"NE") == 0) *r =  45.0;
  else if (strcmp(s,"E" ) == 0) *r =  90.0;
  else if (strcmp(s,"SE") == 0) *r = 135.0;
  else if (strcmp(s,"S" ) == 0) *r = 180.0;
  else if (strcmp(s,"SW") == 0) *r = 225.0;
  else if (strcmp(s,"W" ) == 0) *r = 270.0;
  else if (strcmp(s,"NW") == 0) *r = 315.0;
  else return FALSE;
  return TRUE;
  }

/* PTToDC - converts plunge, trend in degrees to XYZ direction cosines. */
void PTToDC(double p, double t, double dc[3]) {
  double cp;
  p = p*DTOR; t = t*DTOR; cp = cos(p);
  dc[0] = cp*sin(t); dc[1] = cp*cos(t); dc[2] = -sin(p);
  }

/* SphereProject - projects direction cosines to cartesian coordinates of */
/* unit spherical projection.                                             */
int SphereProject(double dc[3], double *x, double *y,
                  int proj, int hemi, int datatype) {
  int    i;
  double f,t[3];
  for (i=0; i<3; i++) t[i] = dc[i];
  if (hemi == LOWER) t[2] = -t[2];
  if (datatype == AXES && t[2] < 0.0) for (i=0; i<3; i++) t[i] = -t[i];
  if (t[2] < 0.0) return FALSE;
  if (proj == STEREOGRAPHIC) f = 1.0/(1.0+t[2]);
```

```
  else f = 1.0/sqrt(1.0+t[2]);
  *x = f*t[0]; *y = f*t[1];
  return TRUE;
  }

/* SphereBProject - back projects cartesian coordinates of unit spherical */
/* projection to direction cosines.                                       */
void SphereBProject(double x, double y, double dc[3], int proj, int hemi) {
  double r2,f;
  r2 = (x*x)+(y*y);
  if (proj == STEREOGRAPHIC) { dc[2] = (1.0-r2)/(1.0+r2); f = 1.0+dc[2]; }
  else { f = sqrt(fabs(2.0-r2)); dc[2] = 1.0-r2; }
  dc[0] = f*x; dc[1] = f*y;
  if (hemi == LOWER) dc[2] = -dc[2];
  }

/*** System Dependent Graphics ***/

/* InitGraphics - initializes graphics system. */
int InitGraphics(char *exepath) {
  char path[MAXPATH],drive[MAXDRIVE],dir[MAXDIR],file[MAXFILE],ext[MAXEXT];
  int  grmode=0,grdriver=DETECT;
  if (opt.device == DXF) {
    if ((dxf = fopen(out_file,"wt")) == NULL) {
      ErrorMsg("Error opening DXF file",-1);
      return FALSE;
      }
    printf("Plotting to %s...",out_file);
    fprintf(dxf,"  0\nSECTION\n  2\nENTITIES\n");
    }
  else {
    fnsplit(exepath,drive,dir,file,ext);  /* get exe path               */
    fnmerge(path,drive,dir,"","");        /* driver is in exe directory */
    initgraph(&grdriver,&grmode,path);    /* load device driver         */
    if (graphresult() != grOk) {
      ErrorMsg("Graphics error, required BGI driver file not found",-1);
      return FALSE;
      }
    setgraphmode(getmaxmode());
    setfillstyle(SOLID_FILL,getmaxcolor());
    bar(0,0,getmaxx(),getmaxy());
    setcolor(0);
    dev_xRatio = 2.5 * (getmaxx()+1.0)/640.0;   /* use 14" VGA as model, */
    dev_yRatio = -2.5 * (getmaxy()+1.0)/480.0;  /* it has 2.5 pixels/mm  */
    dev_xOrigin = 0.0;                          /* left                  */
    dev_yOrigin = getmaxy();                    /* bottom                */
    }
  return TRUE;
  }

/* DoneGraphics - close down graphics system. */
void DoneGraphics(void) {
  if (opt.device == DXF) {
    fprintf(dxf,"  0\nENDSEC\n  0\nEOF\n");
    fclose(dxf);
    printf("done\n");
    }
  else {
    GetKey();      /* wait */
    closegraph();
    }
  }

/* LineOut - output line to graphics system. The string "layer" specifies */
/* the layer output to in a DXF file.                                      */
void LineOut(double x1, double y1, double x2, double y2, char *layer) {
  if (opt.device == DXF)
    fprintf(dxf,"  0\nLINE\n  8\n%s\n 10\n%g\n 20\n%g\n 11\n%g\n 21\n%g\n",
            layer,x1,y1,x2,y2);
  else { /* SCREEN */
    x1 = x1*dev_xRatio+dev_xOrigin; y1 = y1*dev_yRatio+dev_yOrigin;
    x2 = x2*dev_xRatio+dev_xOrigin; y2 = y2*dev_yRatio+dev_yOrigin;
    line(round(x1),round(y1),round(x2),round(y2));
    }
  }
```

```
/*** Non-System Dependent Graphics ***/

/* LineCircleInt - determine intersection parameters for line segment and */
/* circle. Adopted from Rankin 1989, p.220.                               */
int LineCircleInt(double x1, double y1, double x2, double y2,
                  double xc, double yc, double r, double *t1, double *t2) {
  double t,a,b,c,d,disc,dxc,dyc,dx,dy;
  *t1 = *t2 = -1.0;
  dx = x2-x1; dy = y2-y1; dxc = x1-xc; dyc = y1-yc;
  a = dx*dxc + dy*dyc; b = dx*dx + dy*dy; c = dxc*dxc + dyc*dyc - r*r;
  disc = a*a - b*c;
  if (disc > 0.0 && fabs(b) > 1e-6) {
    d = sqrt(disc);
    *t1 = (-a + d)/b; *t2 = (-a - d)/b;
    if (*t1 > *t2) { t = *t1; *t1 = *t2; *t2 = t; }
    return TRUE;
    }
  return FALSE;
  }

/* ClipLineCircle - clip line segment to circle. */
int ClipLineCircle(double xc, double yc, double r,
                   double *x1, double *y1, double *x2, double *y2) {
  double x0,y0,t1,t2;
  if ((*x1 < xc-r && *x2 < xc-r) || (*x1 > xc+r && *x2 > xc+r) ||
      (*y1 < yc-r && *y2 < yc-r) || (*y1 > yc+r && *y2 > yc+r)) return FALSE;
  if (!LineCircleInt(*x1,*y1,*x2,*y2,xc,yc,r,&t1,&t2)) return FALSE;
  if (t2 < 0.0 || t1 > 1.0) return FALSE;
  x0 = *x1; y0 = *y1;
  if (t1 > 0.0) { *x1 = x0 + (*x2-x0) * t1; *y1 = y0 + (*y2-y0) * t1; }
  if (t2 < 1.0) { *x2 = x0 + (*x2-x0) * t2; *y2 = y0 + (*y2-y0) * t2; }
  return TRUE;
  }

/* DrawCircle - output a circle. Adopted from Rodgers and Adams, 1976, p. 216. */
void DrawCircle(double x, double y, double radius, int n,  char* layer) {
  double ainc,c1,s1,x1,x2,y1,y2;
  int    i;
  ainc = 2.0*M_PI/n;
  c1 = cos(ainc); s1 = sin(ainc);
  x1 = x + radius; y1 = y;
  for (i=0; i<n; i++) {
    x2 = x + (x1-x)*c1 - (y1-y)*s1; y2 = y + (x1-x)*s1 + (y1-y)*c1;
    LineOut(x1,y1,x2,y2,layer);
    x1 = x2; y1 = y2;
    }
  }

/* CLineOut - output a line segment clipped to current projection. */
void CLineOut(double x1, double y1, double x2, double y2, char *layer) {
  if (ClipLineCircle(opt.netX,opt.netY,opt.radius,&x1,&y1,&x2,&y2))
    LineOut(x1,y1,x2,y2,layer);
  }

/* DrawSymbol - output a symbol clipped to current projection. */
void DrawSymbol(double x, double y, int symbol, double size, char *layer) {
  double w,h,l;
  switch (symbol) {
    case CROSS:
      w = 0.5*size;
      CLineOut(x,y-w,x,y+w,layer); CLineOut(x-w,y,x+w,y,layer);
      break;
    case TRIANGLE:
      w = 0.5*size; h = w*0.5*sqrt(3.0);
      CLineOut(x-w,y-h,x+w,y-h,layer); CLineOut(x+w,y-h,x,y+h,layer);
      CLineOut(x,y+h,x-w,y-h,layer);
      break;
    case SQUARE:
      h = 0.5*size;
      CLineOut(x-h,y-h,x+h,y-h,layer); CLineOut(x+h,y-h,x+h,y+h,layer);
      CLineOut(x+h,y+h,x-h,y+h,layer); CLineOut(x-h,y+h,x-h,y-h,layer);
      break;
    case HEXAGON:
      w = 0.5*size; h = w*2.0/sqrt(3.0); l = 0.5*h;
      CLineOut(x,y+h,x-w,y+l,layer); CLineOut(x-w,y+l,x-w,y-l,layer);
```

```
      CLineOut(x-w,y-l,x,y-h,layer); CLineOut(x,y-h,x+w,y-l,layer);
      CLineOut(x+w,y-l,x+w,y+l,layer); CLineOut(x+w,y+l,x,y+h,layer);
      break;
    }
  }

/* DrawNetFrame - output projection frame. */
void DrawNetFrame(char *layer) {
  double tickSize = 3.0,x,y;
  DrawCircle(opt.netX,opt.netY,opt.radius,100,layer);
  x = opt.netX+opt.radius; y = opt.netY;
  LineOut(x,y,x+tickSize,y,layer);
  x = opt.netX-opt.radius;
  LineOut(x,y,x-tickSize,y,layer);
  x = opt.netX; y = opt.netY+opt.radius;
  LineOut(x,y,x,y+tickSize,layer);
  y = opt.netY-opt.radius;
  LineOut(x,y,x,y-tickSize,layer);
  }

/*** Gridding ***/

/* GridKamb - calculates grid of density estimates from direction cosine */
/* data. The grid is preprocessed for contouring by subtracting 0.5      */
/* from each count, and normalizing to the contour units.                */
void GridKamb(
  double x[][3], int nData,                    /* dir cos data matrix  */
  double sigma,                                /* sigma value          */
  int datatype, int projection, int hemisphere, /* projection type     */
  double grid[MAXGRID][MAXGRID], int nGrid,    /* the grid             */
  double *zMin, double *zMax)                  /* min and max of grid  */
  {
  double y[3],a,alpha,d,dx,f,xg,yg,zUnit;
  int    i,j,k;
  switch (opt.method) {
    case SCHMIDT :
      a = 0.01;                                /* fractional area      */
      zUnit = nData*0.01;                      /* unit = 1%            */
      break;
    default /* KAMB */ :
      a = (sigma*sigma)/(nData+sigma*sigma);   /* fractional area      */
      zUnit = sqrt(nData*a*(1.0-a));           /* unit = 1 sigma       */
      break;
    }
  if (datatype == VECTORS) alpha = 1.0-2.0*a;  /* half apical angle    */
  else /* AXES */         alpha = 1.0-a;
  switch (opt.smooth) {
    case INVAREA :
      f = 2.0/(1.0-alpha);                     /* weighting factor     */
      break;
    case INVAREASQR :
      f = 3.0/sqr(1.0-alpha);
      break;
    case EXPONENTIAL :
      if (datatype == VECTORS) {
        f = 1.0 + nData/(sigma*sigma);
        zUnit = sqrt(nData*(f-1.0)/(4.0*f*f)); /* unit = 1 sigma       */
        }
      else /* AXES */ {
        f = 2.0*(1.0 + nData/(sigma*sigma));
        zUnit = sqrt(nData*(f*0.5-1.0)/(f*f));
        }
      break;
    }
  dx = 2.0/(nGrid-1);                          /* node spacing         */
  for (i=0; i<nGrid; i++) for (j=0; j<nGrid; j++) grid[i][j] = 0.0;
  xg = -1.0;
  for (i=0; i<nGrid; i++) {
    yg = -1.0;
    for (j=0; j<nGrid; j++) {
      SphereBProject(xg,yg,y,projection,hemisphere);
      for (k=0; k<nData; k++) {
        d = y[0]*x[k][0]+y[1]*x[k][1]+y[2]*x[k][2];  /* dot product      */
        if (datatype == AXES) d = fabs(d);
        switch (opt.smooth) {
          case EXPONENTIAL:
```

```
            grid[i][j] += exp(f*(d-1.0));
            break;
          case INVAREA:
            if (d >= alpha) grid[i][j] += f*(d-alpha);
            break;
          case INVAREASQR:
            if (d >= alpha) grid[i][j] += f*sqr(d-alpha);
            break;
          default:
            if (d >= alpha) grid[i][j] += 1.0;
          }
        }
      yg += dx;
      }
    xg += dx;
    }
  *zMin = 1e30; *zMax = 1e-30;
  f = 1.0/zUnit;
  for (i=0; i<nGrid; i++) for (j=0; j<nGrid; j++) {
    grid[i][j] = (grid[i][j]-0.5)*f;                    /* normalize    */
    if (grid[i][j] > *zMax) *zMax = grid[i][j];         /* find min/max */
    if (grid[i][j] < *zMin) *zMin = grid[i][j];
    }
  }

/*** Contouring ***/

/* Interpolate - determine linear interpolation point between two nodes. */
int Interpolate(double x1, double y1, double z1, double x2, double y2, double z2,
                double *x, double *y, double *z) {
  double dz,dz1,dz2,t;
  dz1 = *z-z1; dz2 = *z-z2;
  if (dz1 == 0.0) {*x = x1; *y = y1; return TRUE;}
  if (dz2 == 0.0) {*x = x2; *y = y2; return FALSE;}
  if ((dz1 > 0.0 && dz2 > 0.0) || (dz1 < 0.0 && dz2 < 0.0)) return FALSE;
  dz = z2-z1;
  t = dz1/dz;
  *x = x1 + (x2-x1) * t; *y = y1 + (y2-y1) * t;
  return TRUE;
  }

/* ContourGrid - output one contour level by linear interpolation among */
/* grid nodes.                                                          */
void ContourGrid(
  double x1, double y1, double x2, double y2,     /* bounding rectangle */
  double grid[MAXGRID][MAXGRID], int ng, int mg,  /* the grid           */
  double level, char *layer)                      /* contour level      */
  {
  double d1,d2,d3,d4,dnx,dny,nx,ny,nxp,nyp;
  double gy1,x3,x4,y3,y4,z,z1,z2,z3,z4;
  int    i,j,found;
  dnx = (x2-x1)/(ng-1.0); dny = (y2-y1)/(mg-1.0);
  z = level;
  gy1 = y1;
  nx = x1;
  for (i=0; i<ng-1; i++) {
    ny = gy1;
    nxp = nx + dnx;
    for (j=0; j<mg-1; j++) {
      nyp = ny + dny;
      z1 = grid[i][j]; z2 = grid[i+1][j];
      z3 = grid[i+1][j+1]; z4 = grid[i][j+1];
      found = 0;
      if (Interpolate( nx,ny,z1,nxp,ny, z2,&x1,&y1,&z)) found += 1;
      if (Interpolate(nxp,ny,z2,nxp,nyp,z3,&x2,&y2,&z)) found += 2;
      if (Interpolate(nxp,nyp,z3,nx,nyp,z4,&x3,&y3,&z)) found += 4;
      if (Interpolate( nx,nyp,z4,nx,ny, z1,&x4,&y4,&z)) found += 8;
      switch (found) {
        case  3: CLineOut(x1,y1,x2,y2,layer); break;
        case  5: CLineOut(x1,y1,x3,y3,layer); break;
        case  9: CLineOut(x1,y1,x4,y4,layer); break;
        case  6: CLineOut(x2,y2,x3,y3,layer); break;
        case 10: CLineOut(x2,y2,x4,y4,layer); break;
        case 12: CLineOut(x3,y3,x4,y4,layer); break;
        case 15:
          d1 = sqrt(sqr(x1-x2) + sqr(y1-y2)); d2 = sqrt(sqr(x2-x3) + sqr(y2-y3));
```

```
          d3 = sqrt(sqr(x3-x4) + sqr(y3-y4)); d4 = sqrt(sqr(x4-x1) + sqr(y4-y1));
          if ((d1+d3) < (d2+d4)) {
            CLineOut(x1,y1,x2,y2,layer); CLineOut(x3,y3,x4,y4,layer);
            }
          else {
            CLineOut(x2,y2,x3,y3,layer); CLineOut(x1,y1,x4,y4,layer);
            }
        }
      ny = nyp;
      } /* for j */
    nx = nxp;
    } /* for i */
  }

/*** Main Procedures ***/

/* LoadDataFile - loads data from file converting to direction cosine */
/* matrix. Returns FALSE on format error.                             */
int LoadDataFile(double x[][3], int *n, FILE *f) {
  char    buf[MAXSTR];
  double  ov,p,t,d[3],r[3][3];
  char    os[MAXSTR];
  *n = 0;
  GetRotMat(r);
  while (fgets(buf,MAXSTR,f)) {
    if (buf[0] == '\0') continue;
    if (opt.format == STRIKE) {
      if (sscanf(buf,"%lf %lf %s",&t,&p,os) != 3)
        if (sscanf(buf,"%lf, %lf, %s",&t,&p,os) != 3)
          if (sscanf(buf,"%lf, %lf %s",&t,&p,os) != 3) return FALSE;
      p = 90.0-p; t = t-90.0;
      if (t < 0.0) t += 360.0;
      if (!OctantVal(os,&ov)) return FALSE;
      if (fabs(ov-t) < 90.0) t += 180.0;
      }
    else {
      if (sscanf(buf,"%lf %lf",&p,&t) != 2)
        if (sscanf(buf,"%lf, %lf",&p,&t) != 2) return FALSE;
      if (opt.format == DIP) {p = 90.0-p; t += 180.0;}
      if (opt.format == POLAR) {p -= 90.0; t = 90.0-t;}
      }
    PTToDC(p,t,d);
    x[*n][0] = d[0] * r[0][0] + d[1] * r[0][1] + d[2] * r[0][2];
    x[*n][1] = d[0] * r[1][0] + d[1] * r[1][1] + d[2] * r[1][2];
    x[*n][2] = d[0] * r[2][0] + d[1] * r[2][1] + d[2] * r[2][2];
    (*n)++;
    if (*n == opt.maxdata-1) {
      ErrorMsg("Data array full at",opt.maxdata);
      return TRUE;
      }
    }
  return TRUE;
  }

/* LoadData - loads data array. */
int LoadData(void) {
  FILE   *f;
  nData = 0;
  if (!(f = fopen(data_file, "rt"))) {
    ErrorMsg("File not found",-1);
    return FALSE;
    }
  if (!LoadDataFile(data,&nData,f)) {
    ErrorMsg("Format error in line",nData+1);
    nData = 0;
    }
  fclose(f);
  return (nData > 0);
  }

/* SaveInit */
void SaveInit(void)
  {
  FILE *f;
  if (!!(f = fopen("scon.ini", "wb"))) {
    fwrite(&opt,sizeof(opt),1,f);
```

```
    fclose(f);
    }
  else ErrorMsg("Error writing file",-1);
  }

/* GetOptions - gets data from user filling the options record "opt", */
/* "data_file" and "out_file".                                        */
int GetOptions(void) {
  int b,n;
  do {
    clrscr();
    printf("\nSPHERE CONTOUR\n");
    printf("--------------\n");
    if (GetStr("Enter data file name, or X to exit program",data_file)) {
      if (data_file[0] == 'x' || data_file[0] == 'X') return FALSE;
      ChangeFileExt(out_file,data_file,".dxf");
      }
    GetInt("Maximum number of data points (up to 32000)",&opt.maxdata);
    GetChoice("Data format (Strike/Dip/Line/Polar)","SDLP",&opt.format);
    GetChoice("Data type (Axes/Vectors)","AV",&opt.dataType);
    b = FALSE;
    if (GetChoice("Change projection details? (Y/N)","NY",&b)) {
      GetChoice("  Projection (Equal area/Stereographic)","ES",&opt.proj);
      GetChoice("  Hemisphere (Lower/Upper)","LU",&opt.hemi);
      GetDbl("  X coordinate of center in millimeters",&opt.netX);
      GetDbl("  Y coordinate of center in millimeters",&opt.netY);
      GetDbl("  Radius in millimeters",&opt.radius);
      }
    b = FALSE;
    if (GetChoice("Change data rotation? (Y/N)","NY",&b)) {
      GetInt("  Number of rotations (0 to 16)",&opt.nRot);
      for (n=0; n<opt.nRot; n++) {
        GetChoice("  Rotation axis (X/Y/Z)","XYZ",&opt.rotAxis[n]);
        GetDbl("  Rotation angle in degrees",&opt.rot[n]);
        }
      }
    GetChoice("Plot type (Scatter/Contour/Both)","SCB",&opt.plot);
    if (opt.plot != SCATTER) {
      GetChoice("Contouring method (Kamb/Schmidt)","KS",&opt.method);
      if (opt.method == KAMB)
        GetDbl("Expected level for uniform dist (1, 2 or 3)",&opt.sigma);
      GetChoice("Smoothing (None/inv Area/inv area Sq/Exp)","NASE",&opt.smooth);
      GetDbl("Minimum contour (0, 1, 2 or 3 recommended)",&opt.minimum);
      GetDbl("Contour interval (1, 2, or 3 recommended)",&opt.ci);
      GetInt("Number of grid nodes (up to 85)",&opt.nGrid);
      }
    if (opt.plot != CONTOUR) {
      GetChoice("Symbols (None/Cross/Triangle/Square/Hexagon)","NCTSH",&opt.symbol);
      GetDbl("Symbol size in millimeters",&opt.symSize);
      }
    GetChoice("Output (Screen/Dxf file)","SD",&opt.device);
    if (opt.device == DXF)
      GetStr("DXF file name",out_file);
    printf("\n");
    b = FALSE;
    if (GetChoice("Save options? (Y/N)","NY",&b)) SaveInit();
    b = FALSE;
    GetChoice("Begin plot? (Y/N)","NY",&b);
    } while (b == FALSE);
  return TRUE;
  }

/* PlotData - outputs scatter plot. */
void PlotData(char *layer) {
  int    i;
  double xn,yn;
  if (opt.symbol == NOSYM) return;
  for (i=0; i<nData; i++) if (SphereProject(data[i], &xn, &yn,
  opt.proj, opt.hemi, opt.dataType)) {
    xn = opt.netX+xn*opt.radius; yn = opt.netY+yn*opt.radius;
    DrawSymbol(xn,yn,opt.symbol,opt.symSize,layer);
    }
  }

/* Contour - grids data and outputs contours. */
void Contour(char *layer) {
```

```
  double level,z1,z2,x1,y1,x2,y2;
  if (nData == 0) return;

GridKamb(data,nData,opt.sigma,opt.dataType,opt.proj,opt.hemi,grid,opt.nGrid,&z1,&z2);
  x1 = opt.netX-opt.radius; y1 = opt.netY-opt.radius;
  x2 = opt.netX+opt.radius; y2 = opt.netY+opt.radius;
  level = opt.minimum;
  while (level < z2+1e-6) {
    ContourGrid(x1,y1,x2,y2,grid,opt.nGrid,opt.nGrid,level,layer);
    level += opt.ci;
    }
  }

/* Initialize - initializes file names and options. */
void Initialize(int argc, char *argv[]) {
  FILE *f;
  opt.ci      = 2.0;
  opt.maxdata = 1000;
  opt.minimum = 2.0;
  opt.netX    = 120.0;
  opt.netY    = 100.0;
  opt.nGrid   = 30;
  opt.plot    = BOTH;
  opt.radius  = 75.0;
  opt.sigma   = 3.0;
  opt.smooth  = INVAREASQR;
  opt.symbol  = HEXAGON;
  opt.symSize = 2.0;
  if (!!(f = fopen("scon.ini","rb"))) {
    fread(&opt,sizeof(opt),1,f);
    fclose(f);
    }
  if (argc > 1) {                              /* file name on command line */
    strncpy(data_file,argv[1],MAXPATH-1);
    ChangeFileExt(out_file,data_file,".dxf");
    }
  }

/* Run - runs main program. */
void Run(char *exepath) {
  if (nData == 0) return;
  if (InitGraphics(exepath)) {
    DrawNetFrame("FRAME");
    if (opt.plot != CONTOUR) PlotData("DATA");
    if (opt.plot != SCATTER) Contour("CONTOUR");
    DoneGraphics();
    }
  }

/* main - initializes options, gets user input, and outputs plots. */
int main(int argc, char *argv[]) {
  int bBatch = (argc == 2);  /* batch mode if one command line parameter */
  Initialize(argc,argv);
  if (argc < 2) {            /* display startup if no parameters         */
    clrscr();
    printf("%s", szInfo);
    if (GetKey() != ENTER) return 1;
    }
  while (bBatch || GetOptions()) {
    if (!(data = malloc((size_t)opt.maxdata*sizeof(point))))
      ErrorMsg("Not enough memory for data array",-1);
    else {
      if (LoadData()) Run(argv[0]);
      free(data);
      }
    if (bBatch) break;
    }
  if (!bBatch) clrscr();
  printf("Sphere Contour terminated.\n");
  return 0;
  }
```