JOURNAL OF COMPUTATIONAL PHYSICS 10, 242–251 (1972)

CONTOR: A FORTRAN Subroutine to Plot Smooth Contours of a Single-Valued Arbitrary Three-Dimensional Surface*

EDWARD L. ROBINSON, JR.†

Department of Astronomy, University of Texas, Austin, Texas 78712

AND

HENRY A. SCARTON‡

Department of Applied Mathematics and Theoretical Physics, University of Cambridge, Cambridge, England

A fast, easy-to-use, accurate FORTRAN subroutine for smoothly contouring an arbitrary three-dimensional surface is described. Practical use of the subroutine is discussed, and a sample complicated set of level curves drawn by the subroutine is presented.

INTRODUCTION

The fast accurate automatic contouring of an arbitrary three-dimensional single-valued surface is often desired in mathematical physics. In this article, an accurate FORTRAN subroutine for the fast smooth plotting of desired contour elevations is described.

PROGRAM THEORY

The contour subroutine assumes a single-valued function of the form $w = f(x, y)$ defined on the rectangle ($x_{\min} \leqslant x \leqslant x_{\max}$; $y_{\min} \leqslant y \leqslant y_{\max}$). The function is held in the computer in the form of a two-dimensional array $w_{i,j} = f(x_i , y_j)$,

* Performed by one of us [E.L.R.] at Mellon Institute; and by the other [H.A.S.] at Carnegie-Mellon University, Mechanical Engineering Department, under a National Science Foundation Traineeship, a Ford Foundation grant, and National Science Foundation Grant GK2452.

† Graduate student at the University of Texas.

‡ Senior Visitor at D.A.M.T.P. on a National Science Foundation Postdoctoral Fellowship. Present address, Mechanics Division, School of Engineering, Rensselaer Polytechnic Institute, Troy, New York 12181.

where $x_i = x_{\min} + (n_x - 1)\,\Delta_x$, $y_j = y_{\min} + (n_y - 1)\,\Delta_y$, and $(1 \leqslant n_x \leqslant N_x$; $1 \leqslant n_y \leqslant N_y)$ with N_x and N_y being the total number of grid points of even spacing $\Delta_x = (x_{\max} - x_{\min})/(N_x - 1) = x_i - x_{(i-1)}$ and $\Delta_y = (y_{\max} - y_{\min})/(N_y - 1) = y_j - y_{(j-1)}$ in the x and y directions, respectively. The program draws smooth curves of constant height w through the rectangular array $w_{i,j}$ and consists of three major parts: (i) subroutines which set up the plotter logic and scale factors, and which draw and label a set of axes and a grid for the plot; (ii) subroutines which find successive points through which a contour passes, and which decide whether a contour is open or closed; and (iii) subroutines which interpolate and draw a smooth curve through these points.

(i) *The plotter set-up subroutines.* The plotter set-up subroutines are concerned with global bookkeeping duties, such as the exact description of the plotter rectangle physical dimensions in terms of actual scale values by means of a calculation of appropriate scale factors, and the setting up of the many internal logic options which determine such things as letter size, possible grid plotting, and labeling of contours. Once these controlling parameters are defined, if desired by the user, the program then draws the coordinate axes with the optional labels and tick marks, and can draw one of several types of grids over the plotting surface. Further details of the straightforward working of this portion of the program can be found in comment statements in the subroutine listing.

(ii) *The contour finding subroutines.* The basic structure of the contour finding subroutines is determined by the possibility of more than one relative maximum occurring in $w_{i,j}$, which could give rise to several disjoint and highly convoluted curves at a given contour level w. Therefore, the routine must (1) examine the complete array $w_{i,j}$ at each contour level so that all curves are found, (2) remember the position of all previously found and plotted curves so that the curves are not repeatedly redrawn, and so that individual curves are not confused with each other, (3) find successive points along a contour and keep them in their proper order so that complicated contours are drawn correctly, and (4) have some means of deciding when all points in a contour curve have been found, and whether the curve is open or closed.

Two methods are employed simultaneously to overcome these basically bookkeeping difficulties. First, the array of points, $w_{i,j}$, is conceptually divided up into rectangular blocks with the four corners $w_{i,j}$, $w_{i+1,j}$, $w_{i,j+1}$, and $w_{i+1,j+1}$. Since the blocks are adjacent to each other, each point in the array is simultaneously a member of up to four blocks. In order to find a point from which to begin a contour curve, the blocks are successively scanned to see if a contour level lies between any two of the four points of the block. As soon as such a block is found, the exact position of the contour level on the edge of the block is found by quadratic polynomial interpolation. For blocks interior to the array $w_{i,j}$, the interpolation

uses the two block points on the proper edge and the two adjacent exterior points on the extension of the proper block edge. The polynomial is required to have the value of w at the two block points, and to minimize the sum of the squared residuals at the two exterior points. If the block is at a boundary or corner of the array $w_{i,j}$ and only one adjacent exterior point exists, the quadratic polynomial is required to have the value of w at the two blocks points and at the exterior point. For the moment, the subroutine ignores the inside of the block and only finds a point along the side. Further interpolation will take place later when the curve is plotted. This gives a beginning point of the contour curve in the (x, y) plane. The subroutine must now find a nearby point which will extend the contour. The side on which the beginning point lies is also the side of an adjacent block. The adjacent block is examined to find a side other than the one already found through which the contour passes. The exact position of the contour level crossover point on this side is again found by interpolation, thus adding a second point in the (x, y) plane through which the contour passes. By continuing this procedure, successive points through which the contour passes are found. The points are stored in the order in which they are found and plotted when all the points on the curve have been found.

The technique of interpolating in a rectangular grid of points is commonly used by other programs as well as this one [1–3]. The quadratic interpolation in the grid which is used here allows the use of a coarser grid of points and gives much improved results over linear interpolation, particularly in the neighborhood of saddle points in $w_{i,j}$. Wahl [4] avoided the use of a grid altogether. Wahl plotted electron density contours in molecules and could calculate an analytic function which approximated the electron densities. Thus, he was able to calculate the position of the contours exactly by extending them ahead in the direction of the tangent to the contour, and then performing a differential correction to the new position by direct comparison to the original function. The disadvantage of Wahl's method is that it must be reprogrammed for each new application. The contours plotted by Pfleiderer *et al.* [5] illustrate another difficulty with Wahl's method. The large mass of experimental data which they plot cannot easily be approximated by an analytic function, so they used a regular rectangular grid. Electron density contours have also been plotted with our program [6]. The reader may wish to compare the plots by Conroy [6] and by Wahl [4] and decide for himself if there has been any loss by using a regular rectangular grid of points.

The second method used for eliminating these bookkeeping difficulties involves setting up a separate memory array. Each element of this integer array is identified with a block in the array $w_{i,j}$ and, by means of a simple code, contains all the necessary information concerning previously found contours. In particular, each element tells whether or not any contour points have been previously found along an edge of the corresponding block. If any have been found, the total number of

points and their block side locations are also stored. For each contour level, the whole memory array is initially set to indicate that no contour points have been found. Whenever a contour point is found along the edge of a block, the corresponding element in the memory array is immediately recalculated to indicate that a point has been found and to indicate on which side it was found. Of course, the memory element which corresponds to the adjacent block must also be recalculated.

The information in the memory array is used for two purposes. First, if a point is found, while scanning blocks to find a beginning point of a contour curve, then the memory array is checked to see if the point has previously been found. If it has, the point is ignored and the scanning process proceeds. In this way (with the exception of block corners to be explained later), a contour curve is only found and drawn once. Second, when extending a contour from block to block, the memory array is checked for previously found points in the new block. If such points are found, one of several alternatives will occur. The routine will first check all remaining sides of the new block for contour points. If one can be found which has not been previously used, the contour curve is simply extended in that direction. If no new points can be found, the subroutine checks to see if it is in the same block as the beginning point of the contour curve. If so, the contour curve is assumed to be closed, with the last point joining on to the beginning point. If it is not the beginning block, the contour curve is assumed to be open, and is then extended backwards from the beginning point until no further points can be found. Whenever a contour curve runs into the edge of the array $w_{i,j}$, it is assumed to be an open curve and is then extended backwards from the beginning point until no further new points can be added. A minor complication arises if a contour exactly passes through one of the corner points of a block. In this case the term "adjacent block" refers to the block which is diagonal to the first block, and which also contains the corner point. The memory method presently being used may cause parts of a contour which contain corner points to be redrawn once. However, the curve will be exactly redrawn so that the user will not be able to detect that the process has occurred without actually watching the plot being made.

(iii) *The contour interpolation and plotting subroutines.* As soon as all points on a contour curve have been found, a smooth curve is interpolated between the points and then plotted. The choice of an interpolation algorithm is severely limited by two properties of many contours. First, in attempting to interpolate with a function of the form $y = g(x)$, in many cases $g(x)$ is found to be multiple valued, and will have points where the first derivative with respect to x is infinite. Second, contours can have segments of very small radii of curvature. This occurs particularly in the neighborhood of saddle points in $w_{i,j}$. Interpolation schemes such as polynomial interpolation are obviously inappropriate because of the first property. Furthermore, an adequate fit to sharp (high curvature) curves requires a high degree

polynomial. Unfortunately, an exactly determined high degree polynomial will often oscillate wildly between its determining points. Spline interpolation formulas [7] alleviate part of the problem by specifying the first and second derivatives at the points. However, they are still ruled out by the first property of contour curves. Also, spline interpolation tends to be slow on a computer. Accordingly, a new interpolation algorithm was developed for use in this program.

Assume that an ordered set of points (x_i, y_i), $i = 1, 2, 3, 4$ is given, and define the following quantities

$$\begin{aligned} \Delta x_i &= x_{i+1} - x_i , \\ \Delta y_i &= y_{i+1} - y_i \qquad (i = 1, 2, 3), \end{aligned} \tag{1a, b, c}$$

and

$$D_i = (\Delta x_i^2 + \Delta y_i^2)^{1/2}.$$

The parametric equations of the three straight lines joining successive points are formed:

$$\begin{aligned} \alpha_1 &= (\Delta x_1/D_1)t + x_2 , \\ \beta_1 &= (\Delta y_1/D_1)t + y_2 , \\ \alpha_2 &= (\Delta x_2/D_2)t + x_2 , \\ \beta_2 &= (\Delta y_2/D_2)t + y_2 , \\ \alpha_3 &= (\Delta x_3/D_3)t + x_3 - (D_2/D_3)\,\Delta x_3 , \\ \beta_3 &= (\Delta y_3/D_3)t + y_3 - (D_2/D_3)\,\Delta y_3 . \end{aligned} \tag{2a, b, c, d, e, f}$$

The interpolated curve from (x_2, y_2) to (x_3, y_3) is then given by

$$\begin{aligned} x &= \frac{\alpha_1(1 - t/D_2)^n + \alpha_2 + \alpha_3(t/D_2)^n}{1 + (t/D_2)^n + (1 - t/D_2)^n} , \\ y &= \frac{\beta_1(1 - t/D_2)^n + \beta_2 + \beta_3(t/D_2)^n}{1 + (t/D_2)^n + (1 - t/D_2)^n} , \end{aligned} \tag{3a, b}$$

where t, the independent parameter, runs from 0 to D_2 and must have the same value when calculating the α_i's and β_i's as when calculating x and y. For the usual set containing more than four points, the set is broken up into appropriate groups of four successive points, and the algorithm is applied repeatedly.

This algorithm is sufficient for closed contour curves. If, however, the contour curve is open, the end points must be handled somewhat differently. The simplest method and the method employed in the program, is to exclude one of the end

straight lines from the calculation of x and y. Thus, for the curve joining (x_1, y_1) to (x_2, y_2) the reduced parametric equations are

$$\begin{aligned} \alpha_1 &= (\Delta x_1/D_1)t + x_1, \\ \beta_1 &= (\Delta y_1/D_1)t + y_1, \\ \alpha_2 &= (\Delta x_2/D_2)t + x_2 - (D_1/D_2)\,\Delta x_2, \\ \beta_2 &= (\Delta y_2/D_2)t + y_2 - (D_1/D_2)\,\Delta y_2, \end{aligned} \tag{4a, b, c, d}$$

and

$$\begin{aligned} x &= [\alpha_1 + \alpha_2(t/D_1)^n]/[1 + (t/D_1)^n], \\ y &= [\beta_1 + \beta_2(t/D_1)^n]/[1 + (t/D_1)^n] \qquad (0 \leqslant t \leqslant D_1). \end{aligned} \tag{5a, b}$$

The accuracy of any new interpolation scheme should be investigated. However, it must be understood that the consideratiions given at the beginning of this section are so overriding, that virtually any interpolation scheme which avoids these difficulties would be acceptable. Indeed, most contour programs simply give up here and draw straight lines between the points (x_i, y_i) [1, 3, 4]. We believe that our algorithm represents a considerable improvement over a simple linear interpolation. The algorithm resists any but the simplest analytic investigation of its error properties, but it is exact in the sense that it passes through every point in the set (x_i, y_i). Also, the algorithm gives an exact fit to any function of the form $y = a + bx$, and it gives an exact fit to the equation $x = a$.

Further analysis of its error properties must be done numerically. We have chosen to illustrate the algorithm by interpolating between points lying on the curve $y = x^2$. Figure 1 is a diagram showing the interpolation process. The points (x_i, y_i), $i = 1, 2, 3, 4$ are $(-1, 1)$, $(0, 0)$, $(1, 1)$ and $(2, 4)$. The straight lines joining these points are given by Eqs. (2), and are plotted as solid lines in Fig. 1. The points (α_i, β_i) are also plotted for the specific value of $t = 0.5\,D_2$. The dotted line is the interpolated curve for $n = 2.0$. The maximum vertical deviation of the interpolated curve from the parabola is $\Delta y = 0.027$ and occurs at $x = 0.833$. Therefore, on the scale in which Fig. 1 is drawn, the interpolated curve is indistinguishable from a parabola. It should be noted that for this case, the error in the fit with our interpolation scheme is an order of magnitude (9.3 times) smaller than the error with a linear interpolation. In practice, one usually plots contours on a scale such that the points (x_i, y_i) appear considerably closer together than they appear on the scale of Fig. 1, so that the limiting factor in the accuracy is often the digital plotter.

The value of n has several effects on the interpolated curve. The curve is everywhere continuous for n greater than zero, but the slope of the curve will only be continuous for n greater than 1. As n increases, greater weight is given to (α_2, β_2)

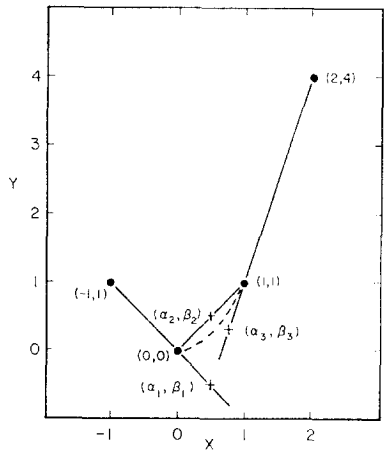

FIG. 1. A plot of the example given in the text of $y = x^2$ and $n = 2$. The filled circles mark the points (x_i, y_i) $i = 1, 2, 3, 4$ and lie on the parabola. Their coordinates are given in parentheses. The crosses mark the points (α_i, β_i), $i = 1, 2, 3$ for $t = 0.5\, D_2$. The solid lines are the straight lines defined by Eqs. (2). The dotted line is the interpolated curve.

in the weighted average of Eqs. (3). Thus, as n increases, the interpolated curve approaches the straight line joining (x_2, y_2) to (x_3, y_3). By examining the errors in many examples such as the one just given, including cases intractable to ordinary interpolation methods, we have chosen $n = 2.5$ as giving generally excellent results. In the example given, $n = 2.5$ results in a maximum error of $\Delta y = 0.069$, which would give a curve just barely distinguishable from the parabola on the scale of Fig. 1.

PROGRAM DESCRIPTION

The contour program is compiled as a FORTRAN subroutine named CONTOR. Once the subroutine argument list is externally prepared, only one call to CONTOR is required per contour plot. The subroutine argument list is too long to include here, but is well documented in the actual FORTRAN source program listing appearing in the dissertation by Scarton [8, pp. 623–654]. Also appearing in this work are additional information concerning the data structure [8, pp. 588–590], a listing of a general program PLTCR which uses CONTOR [8, p. 664 and pp. 668–672] for an arbitrary $w_{i,j}$, and a typical running data deck for PLTCR [8, p. 753]. PLTCR is easily changed to suit individual contour requirements. CONTOR is designed using the Calcomp 663 incremental xy-plotter FORTRAN subroutines. The program can be easily adapted to other equivalent systems.

Possible Difficulties and Their Solutions

Since w is only sampled at a finite number of discrete points, several difficulties with CONTOR may be encountered. To begin with, a completely disarranged and confused contour map, with many contour lines crossing over one another, may be drawn. This behavior is usually the fault of a very coarse grid which is trying to draw curves with a radius of curvature of the order of the grid spacing D_i, Eq. (1c), and is remedied by increasing the number of grid points until well-behaved level curves are constructed. When many changes in curvature occur on the boundary, the boundary needs to be extended for an adequate description of that region. A well-behaved surface, except for a few isolated regions, is again a fault of the ripple radius of curvature of the contour becoming the order of D_i. This distortion of the contour map may occur near a saddle point, for example. If the behavior of the surface in these isolated regions is desired, then a separate localized resolved and magnified plot will have to be constructed. A very flat surface with a sudden spike or hole (or narrow ridge or valley) is difficult to handle and often such regions may be missed entirely if the location of the surface anomaly is situated further away from a grid point than the average width of the anomaly. This sort of problem is cured by experimentation with the various function parameters, and grid size, until this anomaly is fortuitously discovered and resolved and magnified. If no contours are drawn on the plot, the desired contour elevations are either too high or too low.

An Example

A sample of a complicated three-dimensional surface is provided from the field of guided acoustic waves in viscous compressible liquids [8, Chapter III 9, 11]. The magnitude of the dispersion relation from this problem is given by the absolute value of the complex function

$$E(k_r, k_i) = \left| MA \frac{J_1(M)}{J_0(M)} - k^2 \frac{J_1(A)}{J_0(A)} \right|, \tag{6}$$

where

$$M = \left[k^2 + \frac{F^2}{1 + j\frac{4}{3}FD}\right]^{1/2}, \tag{7}$$

and

$$A = \left[k^2 - j\frac{F}{D}\right]^{1/2}, \tag{8}$$

with the dimensionless frequency F and viscosity D being positive real parameters, k being the complex wave number $k = k_r + jk_i$ ($j = \sqrt{-1}$), and J_0 and J_1 being ordinary complex Bessel functions of the first kind and of zeroth and first order.

Equation (6) is ideally suited for contouring and is shown in Fig. 2 with the abscissa k_r extending from -2.0 to 14.0 and the ordinate k_i extending from -1.5 to 6.5. The grid spacing used is $\Delta_{k_r} = \Delta_{k_i} = 0.4$, which allows for 41 grid points in the k_r-direction and 21 points in the k_i-direction. The contour elevations increase logarithmically in height starting with the contour number 2 height of 1.0, with

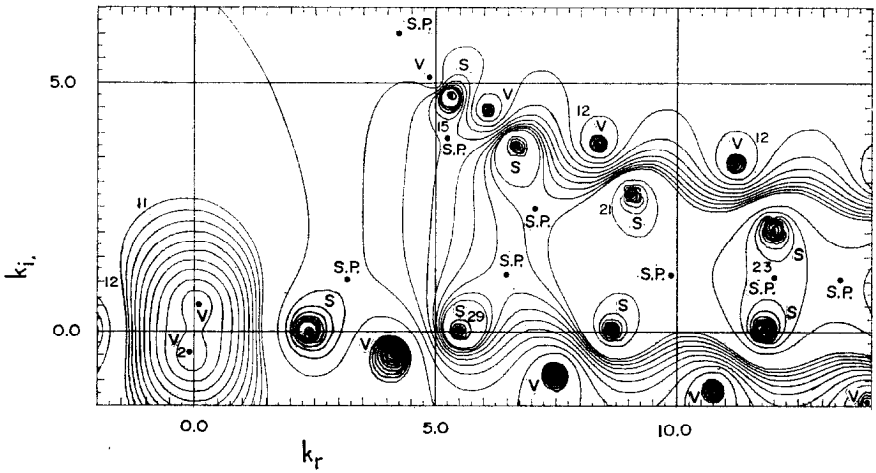

FIG. 2. Contour plot of Eq. (6) with $\Delta_{k_r} = \Delta_{k_i} = 0.4$, $N_{k_r} = 41$, $N_{k_i} = 21$, $F = 0.5$, and $D = 0.01$. (The numbered contour elevations extend logarithmically from contour number 2 with the value of 1.0.)

successively higher elevations of 2, 3,..., 9, 10, 20,..., 90, 100, 200, etc. The values of F and D are 0.5 and 0.01, respectively. In the figure both spikes S [in this case actual poles of $E(k_r, k_i)$] and valleys V (where the function is identically zero) are handled quite well, with the exception of the small circular contours (radii of curvature less than 0.2) near S and V which have become slightly distored because of the coarseness of the grid size (0.4) selected. The contours away from these irregular regions, and also the several saddle points SP, are very nicely drawn. The total on-line computer time on a Univac 1108 digital computer (0.75μ sec add time) required to produce this plot (which includes the calculation of 3, 444 complex Bessel functions [10], but does not include off-line plotter time) was two minutes.

ACKNOWLEDGMENTS

One of us [E.L.R.] gratefully acknowledges the encouragement and support of his supervisor Dr. Harold Conroy; the other [H.A.S.] expresses his gratitude to his thesis advisor Professor Wilfred T. Rouleau for his continued help, advice, and guidance throughout the course of his thesis research.

REFERENCES

1. M. O. DAYHOFF, "A Contour-Map Program for X-Ray Crystallography," *A.C.M. Comm.* **6** (1963), 10, 620.
2. F. FRITSCH, "Contour Plotting Routines," Lawrence Radiation Laboratory Computer Information Center Report N2.3-003, 1966.
3. J. F. VALLEE, "Isomap — Interpolation of Contours from a Set of Grid Points," University of Texas Department of Astronomy Program No. 013S, 1963 unpublished.
4. A. C. WAHL, "Molecular Orbital Densities: Pictorial Studies," *Science* **151** (1966), 961–967.
5. J. PFLEIDERER AND U. MAYER, "Near-Ultraviolet Surface Photometry of the Southern Milky Way," *Astron. J.* **76** (1971), 691.
6. H. CONROY AND G. MALLI, "Molecular Schrödinger Equation. IX. Square and Rectangular States of H_4 and the Molecular Ions H_4^{3+} and H_4^{2+}," *J. Chem. Phys.* **50** (1969), 5049.
7. T. N. E. GREVILLE, "Mathematical Methods for Digital Computers," Vol. 2, (A. Ralston and S. Wilf, Eds.), p. 156, Wiley, New York, 1967.
8. H. A. SCARTON, Waves and Stability in Viscous and Inviscid Compressible Liquids Contained in Rigid and Elastic Tubes by the Method of Eigenvalleys, Ph.D. dissertation, Carnegie-Mellon University, Pittsburgh, 1970. (Available from University Microfilm, Ann Arbor, Michigan 48102, Order No. 70-18, 029E.)
9. H. A. SCARTON, The Method of Eigenvalleys, *J. Comp. Phys.*, to appear.
10. H. A. SCARTON, Double precision FORTRAN subroutines to compute both ordinary and modified Bessel functions of the first kind and of integer order with arbitrary complex argument: $J_n(x + jy)$ and $I_n(x + jy)$, *J. Comp. Phys.* **8** (1971), 295–299.
11. H. A. SCARTON AND W. T. ROULEAU, Axisymmetric waves in compressible Newtonian liquids contained in rigid tubes: Part 1. Steady-periodic mode shapes and dispersion by the Method of Eigenvalleys, *J. Fluid Mech.*, to appear.