ALGORITHM 671
FARB-E-2D: Fill Area with Bicubics on Rectangles—A Contour Plot Program

ALBRECHT PREUSSER
Max-Planck-Gesellschaft

An algorithm plotting contour lines for discrete values z_{ij} given at the nodes of a rectangular mesh is described. A bicubic Hermite polynomial $f(x, y)$ is determined for every rectangle of the mesh, interpolating the z_{ij} and the derivatives z_x, z_y, and z_{xy}. The derivatives are optionally computed by the algorithm. The contours found are normally smooth curves. They consist of polygons approximating intersections with the bicubics. It is possible to fill the areas between them with certain colors or patterns. This is done with a piecewise technique rectangle by rectangle. The method for finding the points of the polygons is shortly reviewed, and some numerical problems are pointed out. The algorithm has a flexible, easy-to-use interface and is easily installed with all plotting systems, provided that a fill-area command is available. A GKS interface may be used.

Categories and Subject Descriptors: G.1.1 [**Numerical Analysis**]: Interpolation—*spline and piecewise polynomial interpolation*; G.m [**Mathematics of Computing**]: Miscellaneous—FORTRAN; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modelling—*curve, surface, solid, and object representations*

General Terms: Algorithms, Design

Additional Key Words and Phrases: Bicubic Hermite surfaces, bivariate interpolation, contour plotting, fill area polygons

1. INTRODUCTION

The algorithm we are presenting can be regarded as a comfortable contouring program. More precisely, it handles the graphical display of curves, which are solutions of the equation $f(x, y) - c = 0$, where $f(x, y)$ are bicubic polynomials defined over rectangles, and c the contour levels. The bicubics can be determined in a way that they fit values z, and derivatives z_x, z_y, z_{xy} at the nodes of a rectangular mesh formed by the rectangles. Then they have smooth and continuous interfaces to their neighbors.

Different contouring methods are reviewed in [13]. Our work differs significantly from other programs in this field because

—it is based on nonlinear bivariate interpolation functions (the bicubics), and
—the areas between the contour lines may be filled with different colors or patterns by a device-independent method.

Author's address: Fritz-Haber-Institut, Faradayweg 4-6, D-1000 Berlin 33, West Germany.

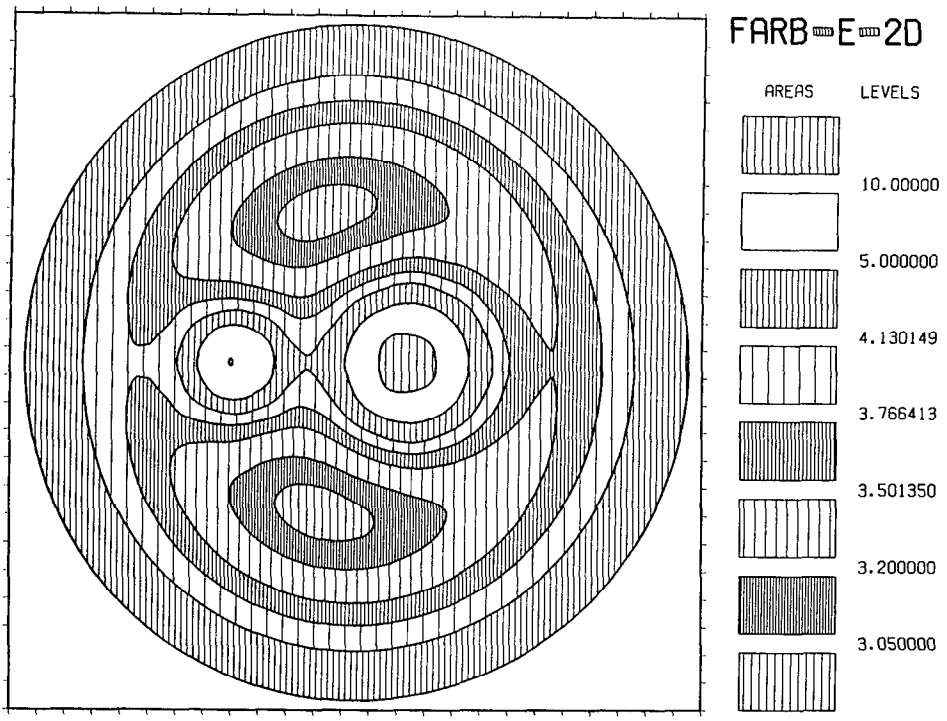

Fig. 1. Sample output computed by **FARB-E-2D**. The function displayed is taken from [14].

We believe that, for the moment, no other program which is capable of both is available.

Producing filled areas is very useful, since it allows one to avoid a pointwise (pixelwise) evaluation of the nonlinear functions. Especially when using high-resolution raster devices, it is often not feasible to evaluate the bicubics at every raster pixel because the computing time needed would be immense.

To give a visual impression of the results that can be achieved with this nonlinear area-filling algorithm, Figure 1 was prepared. It shows the contours of a function taken from [14]. The tick marks at the border indicate the 26×26 mesh. A bicubic patch is used for every rectangle of the mesh. The alternative to a nonlinear method, smoothing the lines resulting from a linear method, yields less accurate results, and may sometimes even lead to overlapping lines if smoothing is done without care (Figure 2). Figures 1–4 demonstrate the superiority of the bicubic interpolation, a very well-known fact. For many applications, it is important that wider meshes be used.

In the next section, we give a review of the methods our algorithm is based on. In Section 3, we discuss some numerical problems that may occur in praxi. When the algorithm was developed, the study of such numerical phenomena was very time consuming. We hope that our efforts resulted in a robust program coping with most situations. Finally, we give a description of the user interface in Section 4 and hints for the installation with local plot systems in Section 5. The plot interface is kept simple, and the user interface is designed to be "easy to use" but also flexible enough to meet the needs of a wide range of users.

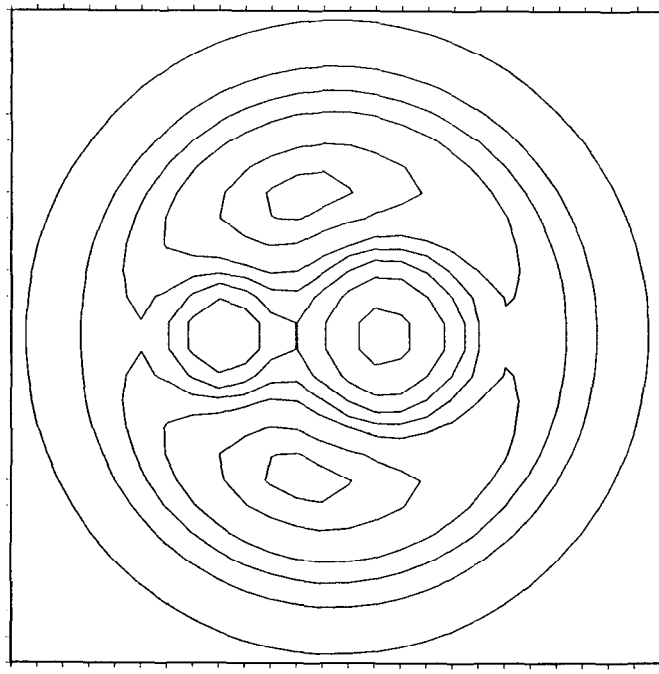

Fig. 2. Same function values as in Figure 1, but linear interpolation as computed by algorithm GCONTR [14]. No smoothing.

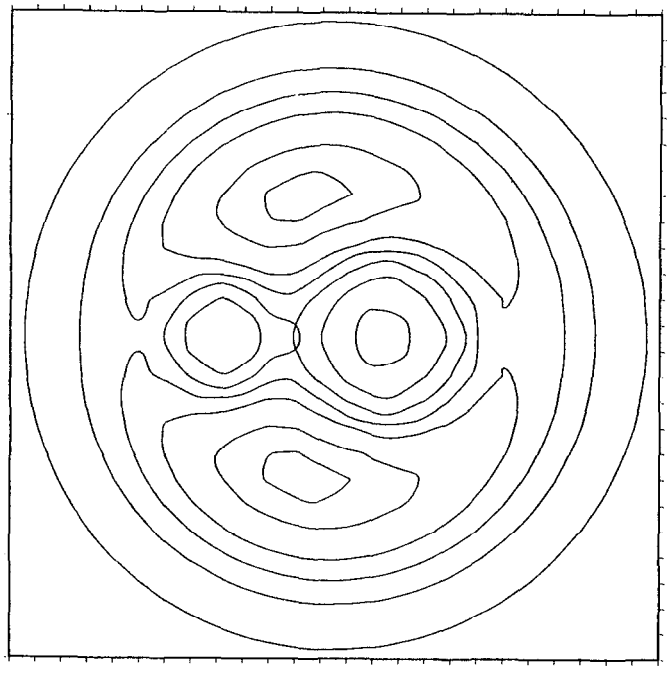

Fig. 3. Linear interpolation of GCONTR (Figure 2) smoothed with the method of [1].

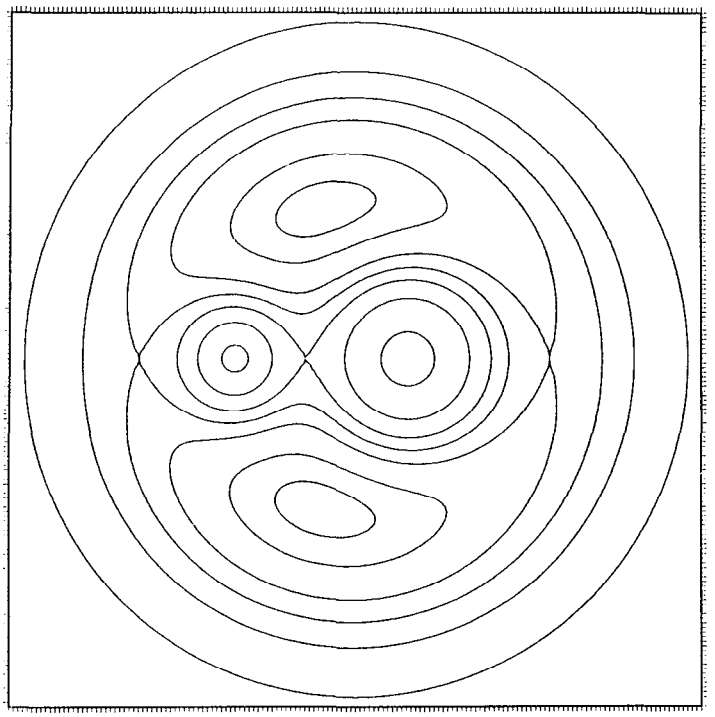

Fig. 4. "Exact" results for Figures 1–3 computed by a 151-×-151 mesh and linear interpolation.

2. METHODS

Bicubics of the Hermite type are well known and widely used for the definition of surfaces over rectangular meshes. Some references are given in [8]. Sixteen values at the vertices of every rectangle uniquely define 16 coefficients of a bivariate Hermite polynomial. These values are the coordinate z, the two partial derivatives z_x, z_y, and the twist z_{xy}. The polynomial is third order on lines parallel to the cartesian coordinates x and y and sixth order in all other directions.

For the estimation of the partials z_x, z_y, and z_{xy} we chose the method of Akima [2] mainly because we think his algorithm is widely in use, and we could rely on existing code [3]. We recognize, however, that there is an ongoing discussion about the best method for the computation of the partials [4], and especially of the twist [7, 9]. Therefore, we designed the third level of the user interface in a way that a user may supply personal derivative values.

The basic ideas for finding the points of the fill-area polygons have been published in [12]. We repeat shortly the main principles of the *Trip Algorithm.* It describes how the points, defining areas of different colors or patterns between two contour levels, can be found for nonlinear interpolation functions within a domain restricted by linear boundaries. In **FARB-E-2D**, the function is bicubic, and the domain a rectangle.

First, the intersection points S of the contours and the sides of a rectangle are determined as zeros of the cubic functions $f - c_i$, where f is the cubic interpolation function on the mesh lines and c_i the contour levels. These zeros are called

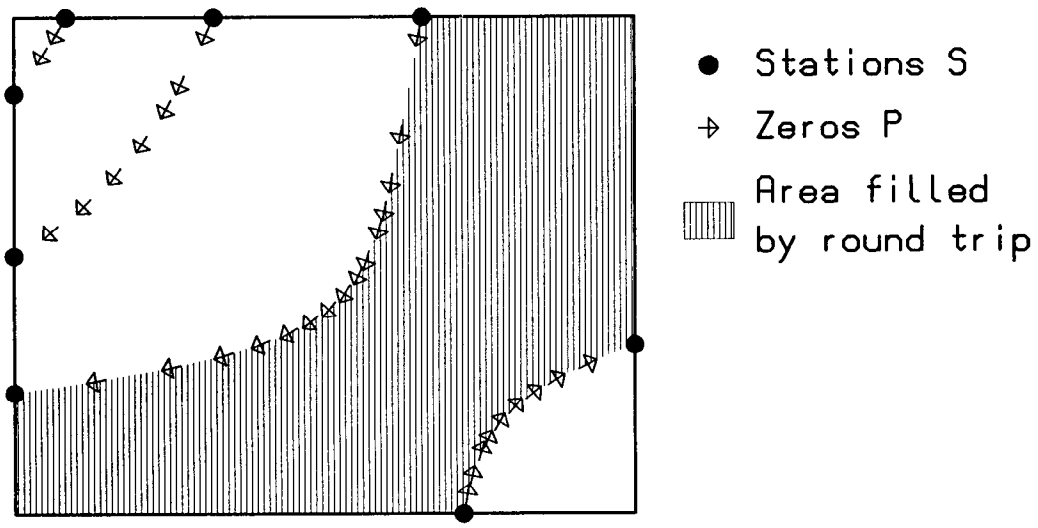

Fig. 5. The *Trip Algorithm* computes *fill-area polygons* for nonlinear interpolation functions. This is a typical situation in a rectangle formed by the mesh lines.

stations S. Instead of computing them level by level for every side, they are generated in a topological sequence. They are ordered counterclockwise on the sides. The contour level c_i and the first derivative of f in the counterclockwise direction of the sides are stored for each station. The sign of this derivative and the contour level define the two different colors to be given to the areas on the two sides of a station. Now nonlinear *rides* (Figure 5) are carried out by computing successive zeros P of the function $f - c_i$ on the contour line, where f is now the bicubic, bivariate interpolation function inside the rectangle. We call this process a "ride" because it connects two stations of the same level. The zeros P are searched with the *regula falsi* on lines normal to the tangent vector of the arising curve. The deviation from the tangent and other local parameters control the step size. The step size is represented by the distance, where the next search line is laid. Unlike in Algorithm 626 [11] we ensure that the normal derivative (perpendicular to the curve) does not change its sign during a ride. This inhibits crossing of curves near saddle points.

After reaching an *end station* of a ride, a *transfer* along the sides is made to the next station, where a new ride begins until a *round trip* is complete and the start station of the trip has been reached. A transfer is carried out in counterclockwise direction on the sides. The vertices of the rectangle are added to the polygon during a transfer, if necessary. The trips will cover the whole area of the rectangle when all stations are used twice, once as the start and once as the end of a ride. We keep the points P of every first ride of a trip in a stack because they often represent a ride in the following, neighboring trip. However, the points will be needed in reverse order. Figure 5 illustrates the *Trip Algorithm*.

3. SPECIAL SITUATIONS

In this section, we would like to draw attention to some numerical problems arising in this algorithm. One class of problems is common to all algorithms,

which try to find the course of an algebraic curve by numerical methods alone, without the help of symbolic computations [5]. When tracing the curve, discrete points are determined with a limited accuracy. This is carried out without determining the *topological structure* [5] of the curve in advance. So it may theoretically happen that the tracing process "switches" to another branch of the curve, which may even be a closed line or a part already encountered in a trip. In this way, *horror trips* may take place, that is, trips which cannot be completed. Such undesirable events cannot be completely inhibited, but their occurrence can be made unlikely by a careful selection of the local parameters in the search process. In **FARB-E-2D**, this is done automatically; the user does not need to be concerned about it.

Another problem on the same level is that we implicitly assume that the curve of a ride is continuous and smooth. For contouring problems, this assumption is allowed in all situations but one: at a saddle point, if the contour line with the function value of the saddle point is to be plotted (Figure 8g). At such a point, where contour lines of the same level cross, a ride has to change its direction discontinuously in order to avoid *self-crossing trips* that would leave some areas unfilled if we assume our strategy for area filling. In this situation our postulation of a constant sign for the normal derivative of a ride is helpful: The ride will not continue in the same direction to the other side of the saddle point. It will stop or take a correct way. If it stops, the direction of the search line is deliberately changed by 90 degrees, and the search for a correct continuation is repeated.

The determination of the stations S may also be ill conditioned. A contour line may be fully or partly identical with a side of a rectangle. In such cases, the position of S is undefined, and different procedures may yield different results. So it may be impossible for a ride to find its correct end, or it may even fail to start.

We tried to improve the reliability of the algorithm in such situations by adding some extensions to the basic procedure described in [12]. We mention some of them, which seem to be unnecessary or redundant at first glance. For instance, we allow a ride to end between two stations near a side. And, more important, if trips cannot be ended successfully, they will be started again from other stations. If stations, which have been used only once, are finally left, trips will be also started in the clockwise direction. In addition, differences are used for determining the sign of derivatives instead of evaluating a polynomial when values near zero are expected. This should eliminate the effect of rounding errors on the sign.

Such instability problems are unlikely to occur in drawings resulting from numerical simulations of problems arising in theoretical physics and engineering, for instance, the solution of partial differential equations. However, in modelled surfaces (Figure 6) or when the mesh values consist of integer values (Figure 7), such situations can easily be produced if certain singular values for the contour levels are chosen. Figure 8 shows a selection of examples with increasing complexity, which we ran successfully on several computer systems with different data formats and arithmetic (see Section 5). "Successfully" in this context means that the whole area of all rectangles is filled.

We feel that pure numerical treatment of our subject must lead to solutions that are safe only up to a certain extent. We believe that the algorithm presented

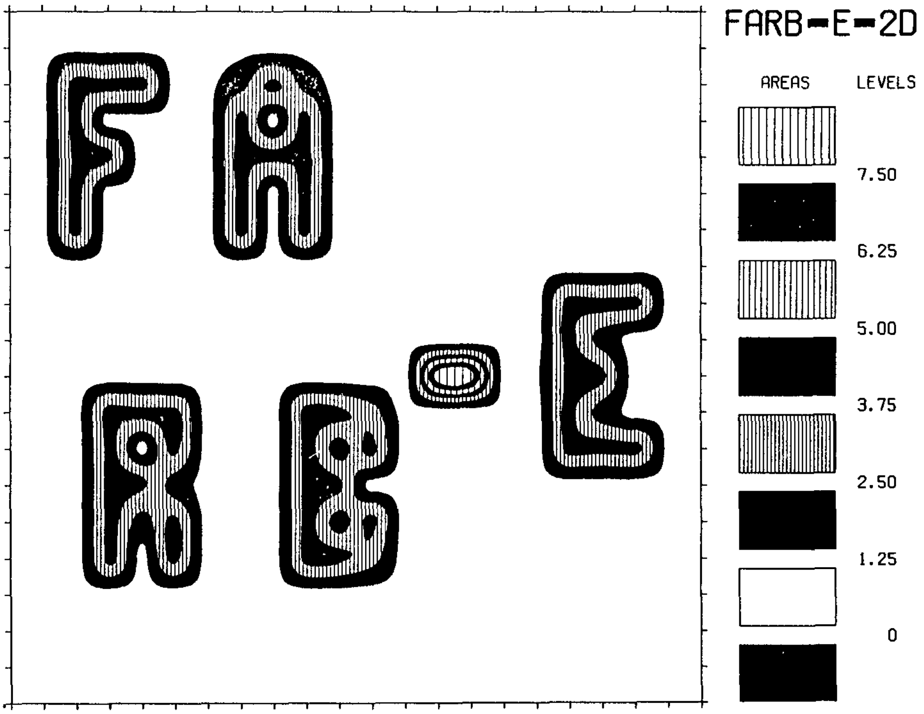

Fig. 6. Image of a modeled surface.

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	4	4	4	1	1	2	4	2	1	1	1	1	1	1	1	1	1	1	1	1
1	1	4	1	1	1	1	4	1	4	1	1	1	1	1	1	1	1	1	1	1	1
1	1	4	4	1	1	1	4	4	4	1	1	1	1	1	1	1	1	1	1	1	1
1	1	4	1	1	1	1	4	1	4	1	1	1	1	1	1	1	1	1	1	1	1
1	1	4	1	1	1	1	4	1	4	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4	4	4	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4	2	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	8	8	1	1	4	4	1	1	1
1	1	1	4	4	4	1	1	1	4	4	3	1	1	1	1	1	4	2	1	1	1
1	1	1	4	1	4	1	1	1	4	2	4	1	1	1	1	1	4	4	4	1	1
1	1	1	4	4	2	1	1	1	4	4	1	1	1	1	1	1	1	1	1	1	1
1	1	1	4	2	4	1	1	1	4	2	4	1	1	1	1	1	1	1	1	1	1
1	1	1	4	1	4	1	1	1	4	4	3	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Fig. 7. Input values for the modeled surface of Figure 6.

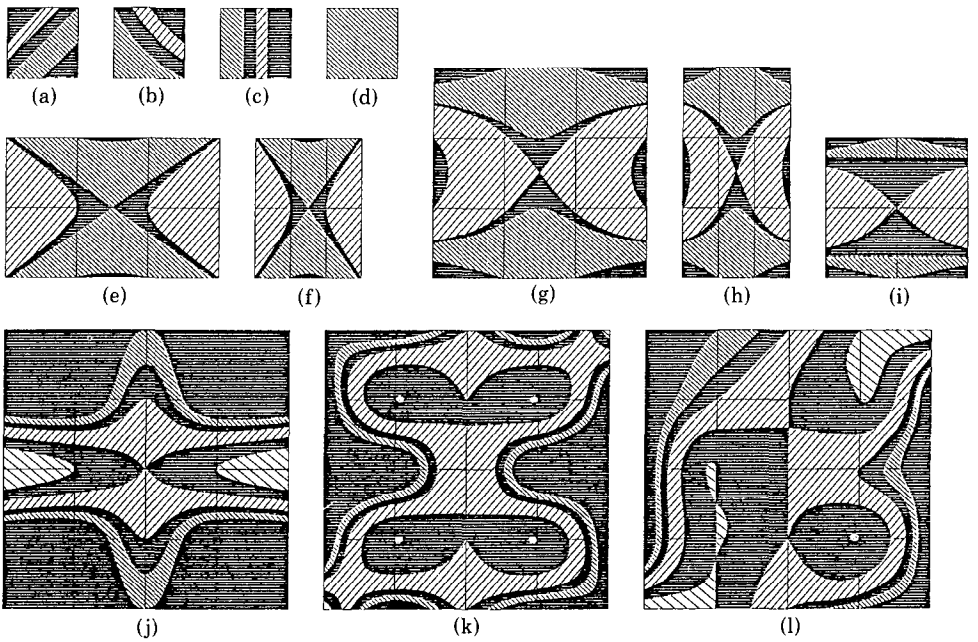

Fig. 8. Examples for special cases. (a)–(d): Contour lines cross vertices or are identical to a side. (e)–(j): Saddle points on sides, inside the rectangle, and at vertices. (k)–(l): Multiple zeros at vertices and on sides.

is a well-balanced mixture between robustness and efficiency, and that it will be helpful in most problem fields.

A user should also be aware of the fact that closed contour lines within a rectangle will not be detected because only the sides of the rectangles are searched for starting points S.

4. USER INTERFACE

As is indicated by the trinomial **FARB-E-2D**, a three-level user interface was designed for the algorithm. For getting access to the first and highest level, a user only needs to pass the two-dimensional array Z, containing the values z_{ij} at the mesh points, the dimension of its first index NXDIM, and the limits NX and NY of the two indices to a subroutine with name FARBE ($i = 1$, NX; $j = 1$, NY). The E in FARBE stands for "Easy to use." Such a call may look like

```
CALL FARBE (Z, NXDIM, NX, NY, MODE)
```

As a default, MODE should be set to zero. The program takes care of all scaling, including the selection of suitable contour levels at round values. The result will be a complete contour plot with *area-filling contours.* The different colors or patterns used for the areas between the contour levels are identified by a legend which will be automatically plotted. When calling FARBE, the result will always be a mesh in the form of a square, even if NX $\neq$ NY. The two indices of the

array determine the position of the z values at the intersections of the mesh lines in the x and y direction, respectively. The mesh lines are displayed by tick marks (Figures 1 and 6).

If the shape of the sketch should be a rectangle or the user wants to specify the mesh lines explicitly, then the entry with name FARB2D has to be called (the second level). Scaling of the x and y coordinates has to be carried out by the user and also the values for the contour levels have to be passed. A call to FARB2D looks like

```
CALL FARB2D (X, NX, Y, NY, Z, NXDIM, CN, ICOL, NC, MODE),
```

where the arrays X and Y with length NX and NY, respectively, give the coordinates for the mesh lines, CN are the values of the contour levels in ascending order, and the integer values of ICOL characterize the selected colors or patterns for the NC-1 different areas between the contour levels (NC = number of contour levels). The first value ICOL(1) of array ICOL with length NC + 1 is used for the areas below and up to the first contour level CN(1), the last, ICOL(NC + 1), for the areas above the highest level CN(NC). Parameter MODE indicates whether pure line drawing, or area filling, or both is requested.

When choosing the third level as entry, a user receives control over a single rectangle that is normally formed by mesh lines. So the user can specify all values at the vertices of the rectangle that are needed to form the bicubic. In principle, all calculations for all rectangles could be executed in parallel because there are no dependencies between them, when all nodal values have been computed in advance. However, we implemented an option which saves computing time in a sequential calculation: If the next rectangle in the queue is the right neighbor of the last, the position of the stations S on the common side may be copied. We also installed a *fill-area buffer*, that will be increased as long as successive rectangles completely receive the same color. This reduces the output of fill-area polygons. For more details, see the documentation part of subroutine FARBRC in the source code.

5. PORTABILITY

The algorithm is written in conformity with the American National Standard FORTRAN, X3.9-1978. However, most of the language constructs used are compatible with earlier versions of FORTRAN, so that only relatively simple modifications have to be made if a compiler of the latest standard is not available.

The plotting interface has been kept very simple. From level two (FARB2D) and below, all plotting is carried out in a single subroutine with name USRPLT, which is responsible for line plotting as well as filling areas. The parameters of USRPLT are all explained within the source code, and an example USRPLT with Graphical Kernal System (GKS) calls [6, 10] is supplied. Mainly, two arrays with the coordinates to be plotted and an index value for the array ICOL have to be passed to USRPLT. At some installations, there may be restrictions concerning the number of points allowed for a fill-area polygon. This presents no problem for our algorithm because the number of points is kept small by the piecewise strategy for area filling (typically not more than 50 points).

When using the highest level FARBE, two additional adaptions to the local plot system may be necessary for the legend and the frame with the tick marks: The calls to GPL for plotting of polygons and to GTEXT for the graphical output of text may have to be modified. GTEXT is called like the CALCOMP routine SYMBOL. GKS calls [6, 10] are supplied for all routines of the graphical interface.

Please note that all x and y coordinates passed to **FARB-E-2D** have to be scaled to centimeters or inches depending on the setting of the installation parameter CMSCAL. The value of CMSCAL may have to be corrected in the modules FARBE and FARBRC when installing the package. When calling FARBE, all scaling is carried out by the program. However, when calling FARB2D or FARBRC, scaling of the x and y coordinates by the user is essential because different representations for the curves will be found depending on the scale of the plot. Larger plots may require that more points with higher precision must be computed. "Problem coordinates" or unscaled "GKS world coordinates" should not be passed to FARB2D or FARBRC.

With the exception of CMSCAL for switching between centimeter and inch, no other machine constant is used in the code. However, the execution of the code, and thus the representation found for the curves, is in special situations sensitive to the arithmetic of different machines. The differences should not be visible, however. We tested the examples of Figure 8 on a CRAY X-MP, CDC Cyber 170, VAX, and IBM AT, all in single precision and with different rounding options. The data for these examples are included in the source code as a test driver for FARB2D. The author would appreciate any feedback about the performance of the algorithm in general, and especially of the results achieved with this test data on other machines.

REFERENCES

1. AKIMA, H. A new method of interpolation and smooth curve fitting based on local procedures. *J. ACM 17*, 4 (Oct. 1970), 589–602.
2. AKIMA, H. A method of bivariate interpolation and smooth surface fitting based on local procedures. *Comm. ACM 17*, 1 (Jan. 1974), 18–20.
3. AKIMA, H. Algorithm 474: Bivariate interpolation and smooth surface fitting based on local procedures. *Coll. Alg. from CACM*, (474-P 1-0 to 474-P 7-0).
4. ALLISTER, D. F., AND ROULIER, J. A. An algorithm for computing a shape preserving osculatory quadratic spline. *ACM Trans. Math. Softw. 7*, 3 (Sept. 1981), 331–347.
5. ARNON, D. S. Topologically reliable display of algebraic curves. *Comput. Graph. 17*, 3 (July 1983), 219–227.
6. BECHLARS, J., AND BUHTZ, R. *GKS in der Praxis*, Springer-Verlag, New York, 1986.
7. BRUNET, P. Increasing the smoothness of bicubic spline surfaces. *Comput. Aided Geom. Des. 2* (Sept. 1985), 157–164.
8. FOLEY, T. A. Scattered data interpolation and approximation with error bounds. *Comput. Aided Geom. Des. 3*, 3 (Nov. 1986), 163–177.
9. FRITSCH, F. N., AND CARLSON, R. E. Monotonicity preserving bicubic interpolation: A progress report. *Comput. Aided Geom. Des. 2*, 2 (Sept. 1985), 117–121.
10. HOPGOOD, F. R. A. *Introduction to the Graphical Kernel System (GKS)*, Academic Press, Orlando, Fla., 1983.
11. PREUSSER, A. Computing contours by successive solution of quintic polynomial equations. *ACM Trans. Math. Softw. 10*, 4 (Dec. 1984), 463–472; also, Algorithm 626, TRICP: A contour plot program for triangular meshes. *ACM Trans. Math. Softw. 10*, 4 (Dec. 1984), 473–475.

12. PREUSSER, A. Computing area filling contours for surfaces defined by piecewise polynomials. *Comput. Aided Geom. Des. 3*, 4 (Dec. 1986), 267–279.
13. SABIN, M. A. Contouring—the state of the art. In *Fundamental Algorithms for Computer Graphics*, R. A. Earnshaw, Ed., NATO ASI Series, Springer-Verlag, New York, 1985.
14. SNYDER, W. V. Algorithm 531, Contour plotting [J6]. *ACM Trans. Math. Softw. 4*, 3 (Sept. 1978), 290–294.

Received June 1987; revised April 1988; accepted July 1988