Drawing contours from arbitrary data points

D. H. McLain

Computing Laboratory, The University, Sheffield, S10 2TN

This paper describes a computer method for drawing, on an incremental plotter, a set of contours when the height is available only for some arbitrary collection of points. The method is based on a distance-weighted, least-squares approximation technique, with the weights varying with the distance of the data points. It is suitable not only for mathematically derived data, but also for data of geographical and other non-mathematical origins, for which numerical approximations are not usually appropriate. The paper includes a comparison with other approximation techniques.

(Received December 1971)

1. Introduction

Suppose that we are given the height of a surface at a number of points in some geographical region. We describe in this paper a method by which a computer can draw the contour lines on an incremental graph plotter or similar device. Typically we may have between a hundred and a thousand such points. The points are assumed to be completely arbitrary, and not arranged on a rectangular or other mesh, for which such methods as those of Coons (1967) can be applied.

There appears to be no completely accepted method for this problem and in particular for interpolating between random points on a surface. One method, Bengtsson and Nordbeck (1964), is to dissect the region into triangles using the data points as vertices; linear interpolation within each triangle gives an approximation to the contour by straight lines. Schmidt's approach (1966) is to use quadrilaterals instead of triangles; then if the approximating function within the quadrilateral is taken to be a polynomial $a + bx + cy + dxy$, with coefficients chosen to fit the data at the vertices, the contours become hyperbolic arcs and Pitteway's (1967) algorithm can be used. Probably more widely used are variants of Shepard's interpolation (1968), developed for later versions of the SYMAP program. Shepard calculates the height as a weighted average of the data points, with the weights depending on their distances and relative directions, but 'corrections' must also be imposed near the data points to alter the zero gradients which would otherwise be an undesirable feature of the method. The proprietary Calcomp program GPCP (General Purpose Contouring Program) uses a similar weighted averaging technique, but only taking neighbouring data points into account.

The interpolation method described in this paper, like Shepard's, uses a weighting technique with weights depending on the distances of the data points; however in our case the weights do not determine the height directly, but are used with a least squares fit to find the coefficients of a quadratic polynomial to act as an approximation to the surface. The method is described in Section 2.

The second component of the method described here is the use of the interpolation method to draw the contour lines. The programming difficulties of recording which parts of which lines have already been drawn, discussed in Cottafava and Le Moli (1969), may be avoided by splitting the region into small rectangles or windows within which the contours can be assumed to be uncomplicated. The algorithm is described in Section 3 and may, of course, be applied to any interpolation formula.

In practice, however, if applied directly these techniques are expensive in central processor utilisation. This is because the contribution from every data point is assessed individually for every step of the plotter. The method actually recommended is to use the weighted least-squares interpolation method to estimate the height at the corners of a rectangular mesh, and then to combine the much faster bicubic spline interpolation, e.g. Boor (1962), within each mesh window with the contour method of Section 3. Simpler interpolation methods within the windows can reduce the central processor usage still further, if kinks in the contour lines can be tolerated along the mesh lines, or if great accuracy is not required: these are described in Section 4.

Figs. 1, 2, and **3** illustrate the use of the program on what is a particularly difficult problem; reconstructing the contours of an area of the Peak District between Sheffield and Manchester from various numbers of points. The area extends from the Snake Pass in the North over Kinder Scout and the beautiful Vale of Edale to the Mam Tor ridge on the South. The region is part of the Pennine Ridge, and is basically a plateau, cut by winding glaciated valleys. Fig. 1 shows the reconstruction from 504 data points on a regular 24 × 21 grid using cubic splines alone, and presents a fairly accurate contour map of the region. Figs. 2 and 3 show the reconstruction from a random selection of 400 and 200 of these points respectively.

2. Distance-weighted least-squares approximation

When a large number of data points, (x_i, y_i, z_i) $i = 1, \ldots, n$ are involved it is not normally advisable to use the 'classical' method of fitting a polynomial $z = f(x, y)$ of high enough degree to provide an exact fit at all n points. Apart from the difficulty of specifying such a polynomial (which terms in $x^r y^s$ should be included?), almost always the resulting surface will be extremely folded and mountainous.

The distance-weighted, least-squares approximation is found as follows: Suppose we wish to estimate the surface height $f(a, b)$ at a point (a, b). Our aim will be to find a polynomial $P(x, y)$, which we illustrate by the general polynomial of degree two:

$$P(x, y) = c_{00} + c_{10}x + c_{01}y + c_{20}x^2 + c_{11}xy + c_{02}y^2$$

which will be as accurate a fit as possible, in the usual least-squares sense, to the data points (x_i, y_i). The difference between this and the usual least-squares approximation, as described in most text books in numerical techniques, for example Rivlin (1969), is that here we require those data points (x_i, y_i) close to (a, b) to carry more weight than the more distant points. More precisely we choose the coefficients c_{rs} to minimise the quadratic form

$$Q = \sum_{i=1}^{n} (P(x_i, y_i) - z_i)^2 . w((x_i - a)^2 + (y_i - b)^2)$$

where w is a weight function, such as $w(d^2) = 1/d^2$, which is large when (a, b) is close to (x_i, y_i) and small when it is remote. The minimisation is achieved in the usual way by solving the (in our example, six) linear equations

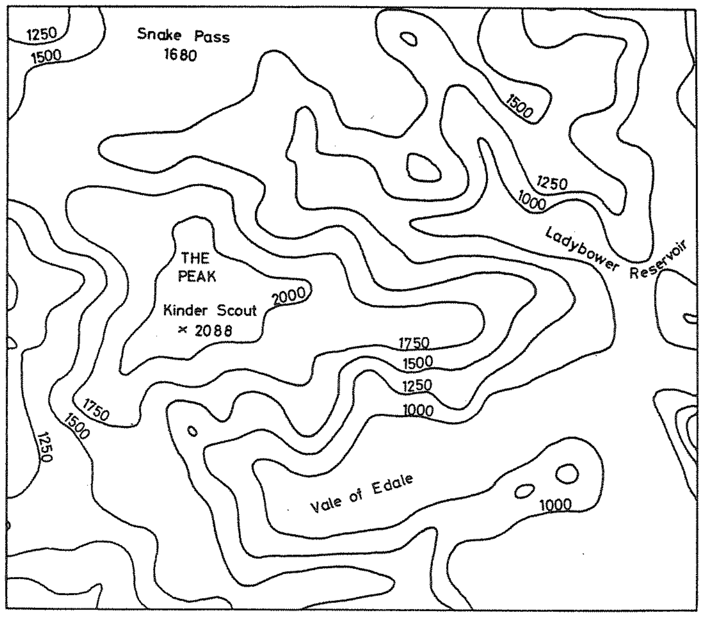

Fig. 1 The Peak District reconstructed from 504 data points

$$\frac{\partial Q}{\partial c_{rs}} = 0$$

to find the (six) unknown coefficients c_{rs}. Having found the coefficients we can evaluate the height of the surface at the point (a, b) as $P(a, b)$.

The method of performing this on the computer is straightforward. We build up a symmetric (in the example, six by six) matrix E and a (six long) vector V as follows. To the element of E in the row corresponding to c_{rs} and the column corresponding to c_{tu} we add $x_i^{r+t} y_i^{s+u}$ times the weight of (x_i, y_i), and to the element of V corresponding to c_{rs} we add $x_i^r y_i^s z_i$ times the weight. When this is done for all n points, the vector of coefficients C may be found by solving the linear equations $EC = V$.

The method may be easily refined to allow some of the points (x_i, y_i) to be more important than others. For example, if a collection of points lie close together each of these should carry less weight than an isolated point. To allow for this we would merely multiply each weight $w((x_i - a)^2 + (y_i - b)^2)$ by a precalculated further term s_i, the significance of the ith point. Clearly s_i would be larger for isolated points than those in a cluster.

The above description has been illustrated by a second degree polynomial $P(x, y)$. Clearly any polynomial could be used, provided that the total number of coefficients c_{rs} was not larger than the number n of data points. Indeed one can construct similar problems, for example when interpolating on a sphere, where one might prefer a trigonometric expansion:

$$P(\theta, \phi) = \ldots + c_{rs} \sin r\theta . \sin s\phi + d_{rs} \sin r\theta . \cos s\phi + \ldots .$$

Nor are we restricted to linear expressions. The referee has suggested the possibility of applying a least-squares fit to determine the coefficients c_{ijk} of a surface expressed in the form of an equation

$$f(x, y, z) = c_{000} + c_{100}x + \ldots + c_{002}z^2 = 0$$

This has the merit of producing an exact fit when the original surface is a sphere or similar quadric. However, as the referee has pointed out, the evaluation of these coefficients to give a general least-squares fit involves a minimisation problem which does not lead to linear equations, and he has suggested that the idea is more applicable to the case where the data is on a regular mesh, so that the 9 independent coefficients are

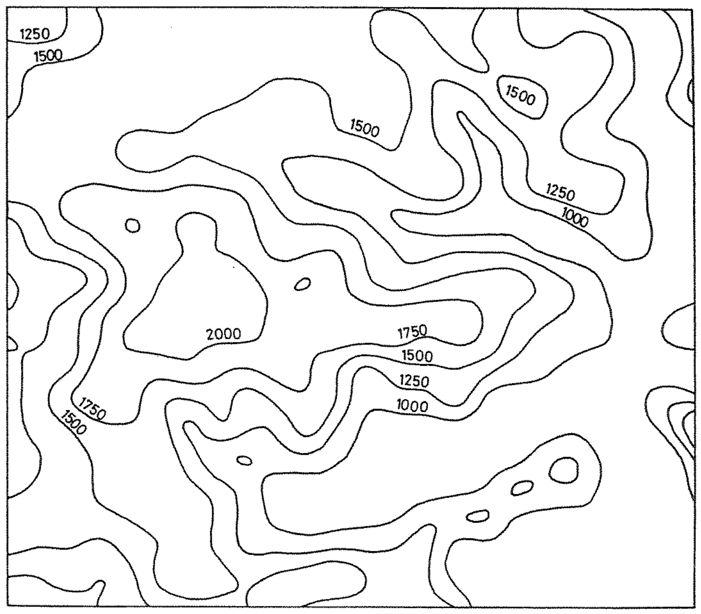

Fig. 2 The Peak District reconstructed from 400 data points

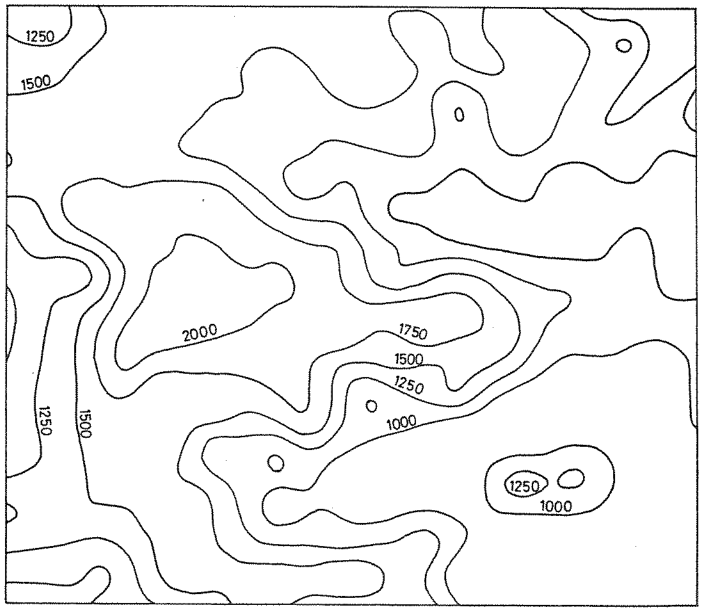

Fig. 3 The Peak District reconstructed from 200 data points

determined from the 9 nearest data points. This method has the disadvantage of not always producing a single-valued function, and in general (see Appendix 4) does not appear to lead to as good a fit as some of the other methods.

The tables in Appendix 4 show that, of the weighted least-squares approximation methods, the second degree polynomial gives satisfactory results over a variety of applications, and offers a good compromise between accuracy and run time. To illustrate the deficiencies of lower degree polynomial approximation, consider the one dimensional analogue of the method applied to the four data points:

$$x = 0, 1, 2, 3$$
$$z = 1, 0, 0, 1$$

Both the linear and first degree polynomials in this case give rise to a hill in the middle of the valley between $x = 1$ and 2. Thus, assuming a distance weighting function of $1/d$, where d is the distance between the points, these polynomials both lead to a value of $\frac{1}{3}$ when $x = 1\frac{1}{2}$, instead of the more reasonable value of $-\frac{1}{8}$ given by the use of a second degree polynomial. The use of third and higher degree polynomials for P sometimes leads to more accurate approximations, particularly if the data is of mathematical origin. But sometimes these can be much less accurate than the second degree case: not surprisingly this occurs for those types of data for which the high degree polynomial method is particularly inaccurate, e.g. for those surfaces without a convergent Taylor series.

There is considerable flexibility over the choice of the function $w(d^2)$ giving the weight of the data point as a function of the square of its distance d^2. The use of the simple $1/d^2$, or to avoid arithmetic overflow, $1/(d^2 + \varepsilon)$ for small ε, does not prove particularly satisfactory, see Appendix 4. This is probably because the data points remote from (a, b) have too much influence. The more rapidly decreasing functions, such as $w(d^2) = 1/(d^2 + \varepsilon)^4$, have been found to give much more accurate results. So do the exponentially decreasing functions $w(d^2) = \exp(-\alpha d^2)$ for some constant α of the order of the inverse of the square of the average distances between neighbouring data points. If the data points are subject to experimental error the use of such an exponential function, with perhaps a smaller value of α, leads to smoothing of the surface. However, if the data are exact, then when (a, b) is very close to some (x_i, y_i) we should expect that to dominate absolutely; hence a function such as

$$w(d^2) = \exp(-\alpha d^2)/(d^2 + \varepsilon)$$

is usually marginally better, and, provided the central processor evaluation time can be tolerated, this is recommended.

Note that the case for negative exponential weighting receives some slight theoretical support from the fact that, in approximations using cubic splines, the effect of remote data points decreases approximately exponentially with distance (though not with the square of the distance). This is described in Ahlberg (1970), who shows that the effect of one data point on a regular mesh decreases approximately with a factor of $\sqrt{3} - 2 = -0{\cdot}2679$ from one interval to the next. The use of an exponential function has the practical advantage that, if n is large, say over 500, then we can divide the region into subregions, and at a slight cost in programming effort, ignore (i.e. assign zero weight to) the data points in remote subregions. If one ensures that the points ignored would otherwise have sufficiently small weights then the resulting loss of continuity and smoothness of the contours is usually not noticeable.

3. A method for drawing contours

We now describe a method for drawing the contour line $f(x, y) = c$, where we have a method of calculating the surface height $f(x, y)$. The present approach is a more direct one than is usually adopted, e.g. Cottafava and LeMoli (1969) which involves finding the points of intersection between the curve and various mesh lines, applying heuristics to determine which of these are linked, and in which order, and fitting curves through these. We also assume that there are no degeneracies in the function $f(x, y)$, such as the cliffs or caves discussed in Morse (1969).

Consider the situation where the plotter has started the line and reached the point (a, b). We present the following simple algorithm to decide in which direction (horizontal, vertical or diagonal) the plotter is to make its next step. Assume that there are no extremely sharp corners on the (true) contour. Then the next step should not be very different from the previous one. Our algorithm chooses between three possible such steps: one in the same direction as the previous, and the two steps at an angle of 45° to either side. For example if the first step was North we would choose the second from North, North-West and North-East, and if the first was North-West the second would be chosen from North-West, North and West. The choice is made by evaluating f for each of the three possible directions, and selecting that one with value closest to c. This is, of course, different from the 'correct' method of finding the distance of the three points from the contour and selecting the step which minimises this. However, unless the second derivative of f is important, i.e. unless the slope is changing rapidly, the effect is normally the same.

There remains the problem of deciding where to start and finish each contour, and recording which contours have been drawn. This can be simply achieved by dividing the region into small rectangles or windows. These are to be so small that we expect no closed contours to lie completely within a window as an isolated peak or hole. In practise this is not a severe limitation—the windows do not seem to have to be very small for this to be satisfied. We can then handle the windows in sequence, and as we now start and finish each contour at an edge we can identify them by their intersections with the edges. These intersections can be found by solving an equation in one variable—thus along the edge $y = y_1$ we need only find the roots of the equation $f(x, y_1) = c$ in the appropriate range of x. Having found such a root it is sensible to test whether it represents the exit point for a contour we have previously drawn, to avoid drawing it twice. If not, then the program can enter the drawing loop, described above, having pretended that its previous step was in the direction at right angles to the edge into the window.

The algorithm continues by choosing between three steps as described above, and after making each step testing whether the pen has moved outside the rectangle. It seems prudent to include a further complication. If the 'true' contour turns back to within a plotter step length or so of crossing itself, a situation can arise where the routine would enter a loop. Therefore the inclusion of a test is recommended to ensure that the number of steps made in any rectangle does not exceed some limit.

An ALGOL procedure for this algorithm is given as Appendix 2.

4. The regular mesh case

If the data are given not at arbitrary points in the region but on a rectangular mesh, more conventional interpolation methods are practicable. Indeed, if the mesh is very fine, then linear interpolation using the four values at the corners of a window may be satisfactory. Because this is equivalent to choosing the four coefficients in the fitting polynomial

$$f(x, y) = c_{00} + c_{10}x + c_{01}y + c_{11}xy$$

to fit the four corners, the contours within a window are hyperbolic arcs. These may, of course, be drawn more rapidly than by using the general technique of Section 3, see for example Pitteway (1967), or Partridge (1968). Although this hyperbolic interpolation ensures the continuity of contours across the window edges, the continuity of the derivative is not ensured. Kinks in the curves are the result; if the mesh is fine enough these can sometimes be accepted, but the method is not generally recommended. There are, of course, ways of smoothing such a collection of connected arcs before drawing. However, the use of bicubic spline approximations leads to much more accurate results.

Bicubic spline approximations (Boor, 1962), are the two dimensional analogue of the better known cubic spline approximations of functions of one variable. Their use leads to the continuity of the contours and their first and second derivatives across window edges, i.e. to smooth curves. Much of the theory of splines of one variable—the uniqueness and best-fit theorems, described for example in Rivlin (1969) or Greville (1967)—can be rederived for splines of two (or more) variables. However, it is simpler to regard a bicubic spline approximation $f(x, y)$

from a rectangular mesh $z(x_i, y_j)$ as the result of first using cubic splines to interpolate in x to obtain $f(x, y_j)$ for each row y_j, and then using these to interpolate the y variable to obtain $f(x, y)$. In contouring, rather than evaluating the function in this way, for each window we would evaluate the sixteen coefficients of

$$f(x, y) = \sum_{r,s=0}^{3} c_{rs}x^r y^s \ .$$

Again, the coefficients may be calculated by treating the variables x and y sequentially. A procedure for this is given in Appendix 3. The one-dimensioned cubic spline procedure which is called for each variable x, y, is an adaptation of the very efficient algorithm Splinefit, Späth (1969), the adaptation being desirable to make it return the polynomial coefficients rather than their values.

5. Recommended method for arbitrary data

Because the methods described in Sections 2 and 3, require the evaluation, for every graph plotter step, of the contribution from every data point to every element of the matrix, the computing time can be large. To avoid this, the recommended method is to reduce the problem to the regular mesh case by imposing a mesh on the region and using the weighted least-squares interpolation to estimate the surface height of the mesh points. Bicubic splines can be used to estimate the surface height within each mesh window, and the method of Section 3 used to draw the lines. In this case the points of intersection of the curves and the mesh edges are determined by the roots of a cubic polynomial, which fact should be used in finding them.

In practice the use of bicubic splines in this way leads to reasonably accurate contours, although two successive approximation techniques are involved. For most surfaces we have found that the variance of the errors in the two parts of the process are of the same order of magnitude, with the errors from the weighted least-squares interpolation usually slightly the larger. This does not, however, apply to the alternative, still faster method of reducing the problem to a (probably somewhat finer) mesh and applying hyperbolic interpolation.

6. Extensions to the systems

Provided the weight function $w(d^2)$ is well-behaved the weighted least-squares approximation fits a smooth surface to the data points. In mathematical terms all the differentials $\frac{\partial^{r+s}z}{\partial x^r \partial y^s}$ exist. If the actual surface has singularities, such as discontinuities (cliffs) or discontinuous first derivatives (escarpments), the algorithm will smooth these out, and they will appear as regions where the contours are close together, sharply bent, or otherwise untypical. If we know in advance where such discontinuities occur, the algorithm could be adapted by ignoring the contribution from the data point (x_i, y_i) to the value at the point (a, b) if the line joining these intersects the discontinuity. Thus the weight would be a function not only of the distance between two points, but the direction and location of the line joining them. As is clear from the SYMAP program, (Robertson, 1967), the computer time in such techniques can be large. A more efficient approach would be to separate the area into subregions with the discontinuities as boundaries.

The method can be easily extended in another direction to include additional information from the data points (x_i, y_i). The most useful example of this is probably the case where we have not only the height z_i but the slope at this point—the values of $\left.\frac{\partial z}{\partial x}\right]_i$ and $\left.\frac{\partial z}{\partial y}\right]_i$. This could be important if the computing or experimental (e.g. if drilling was involved) costs in examining the surface close to a point (x_i, y_i, z_i) were smaller than the costs of investigating new points. To include, for example, the information on $\left.\frac{\partial z}{\partial x_i}\right]$ we merely, for each r, s, t, u, add into the matrix element of E in the row associated with c_{rs} and the column associated with c_{tu} the term

$$r\, t\, x_i^{r+t-2}\, y_i^{s+u}\, w(d_i^2)$$

and add into the element of V corresponding to c_{rs} the term

$$r\, x_i^{r-1}\, y_i^s \left.\frac{\partial z}{\partial x}\right]_i w(d_i^2) \ .$$

There is scope for considerable flexibility in devising different weight functions for this derivative information from the weight function for the original height information. However experiments indicate that the accuracy of the result is not very sensitive to changes in the relative weights. They also indicate that using the height and first derivatives at n points gives, on average, about the same accuracy as using the height information alone at $2\frac{1}{2}n$ points spread more densely over the same region. In other words it is only marginally better to have information at each of $3n$ points than to have 3 times the information at each of only n points.

Appendix 1 Distance-weighted least-squares quadratic approximation

```
real procedure weighted least squares approximation (a, b, x, y,
  z, npts);
value npts; integer npts; real a, b; array x, y, z;
comment If we are given the value of z[i] at each of the arbitrary
  data points (x[i], y[i]), i = 1, . . ., npts, this procedure
  estimates the value of z at the point (a, b), using the method of
  Section 2;
begin integer i, j;
  real xi, yi, x2, y2, term, xterm, yterm, xxterm, yyterm,
    xyterm, zterm;
  array e[1:6, 1:6], coef, v[1:6];
  for i := 1 step 1 until 6 do
  begin v[i] := 0·0;
    for j := i step 1 until 6 do e[i,j] := 0·0;
  end;
  for i := 1 step 1 until npts do
  begin xi := x[i]; yi := y[i]; x2 := xi↑2; y2 := yi↑2;
    term := w((xi − a)↑2 + (yi − b)↑2);
  comment w is the weighting function as described in
      Section 6;
    xterm := xi × term; yterm := yi × term;
    xxterm := x2 × term; yyterm := y2 × term;
    xyterm := xi × yterm;
    e[1, 1] := e[1, 1] + term;
    e[1, 2] := e[1, 2] + xterm;
    e[1, 3] := e[1, 3] + yterm;
    e[1, 4] := e[1, 4] + xyterm;
    e[1, 5] := e[1, 5] + xxterm;
    e[1, 6] := e[1, 6] + yyterm;
    e[2, 4] := e[2, 4] + x2 × yterm;
    e[2, 5] := e[2, 5] + x2 × xterm;
    e[2, 6] := e[2, 6] + y2 × xterm;
    e[3, 6] := e[3, 6] + y2 × yterm;
    e[4, 4] := e[4, 4] + x2 × yyterm;
    e[4, 5] := e[4, 5] + x2 × xyterm;
    e[4, 6] := e[4, 6] + y2 × xyterm;
    e[5, 5] := e[5, 5] + x2 × xxterm;
    e[6, 6] := e[6, 6] + y2 × yyterm;
    zterm := z[i] × term;
    v[1] := v[1] + zterm;
    v[2] := v[2] + xi × zterm;
    v[3] := v[3] + yi × zterm;
    v[4] := v[4] + xi × yi × zterm;
    v[5] := v[5] + x2 × zterm;
    v[6] := v[6] + y2 × zterm;
  end of contributions from data points;
  e[2, 2] := e[1, 5]; e[2, 3] := e[1, 4]; e[3, 3] := e[1, 6];
  e[3, 4] := e[2, 6]; e[3, 5] := e[2, 4]; e[5, 6] := e[4, 4];
  for i := 1 step 1 until 5 do
  for j := i + 1 step 1 until 6 do e[j, i] := e[i, j];
  Gauss (6, e, v, coef);
  comment This call to Gauss is assumed to solve the 6 linear
    equations $\sum_{y=1}^{6}$ e[i, j] coef[j] := v[i] for the 6 unknown
    coef[j];
  weighted least squares approximation :=
  coef[1] + a × (coef[2] + b × coef[4] + a × coef[5])
                        + b × (coef[3] + b × coef[6]);
end of procedure;
```

Acknowledgement. Much of the detail of the above was developed by K. L. Williams of University College, Cardiff.

Appendix 2 A contouring procedure using the method of Section 3

In this procedure to draw a contour using the (global) height function $f(x, y)$, the existence of two procedures *movepento* (a, b) and *graphstep* (*xstep*, *ystep*) to control the plotter is assumed. The first moves the pen in the up position to (a, b), and the second moves the pen in the down position a distance of *xstep* in the x direction and *ystep* in the y direction.

A procedure *findroots* (*function*, *x*, *xmin*, *xmax*, *roots*, *nroots*) to find the *nroots* real roots of the polynomial *function* (x) between *xmin* and *xmax*. In practice it would seem sensible to replace this general root finding procedure by one specifically for the degree of the polynomial involved. If the bicubic spline method is being followed the CACM algorithm 326, Nonweiler (1968) is suitable. In that case the calls of *findroots* should evaluate the coefficients of the polynomial which would then be called by value. The other method is followed here for generality.

```
procedure drawcontour (contour, xmin, xmax, ymin, ymax, step);
value contour, xmin, xmax, ymin, ymax, step;
real contour, xmin, xmax, ymin, ymax, step;
comment draws the contour line for height contour in the rectangle
xmin ≤ x ≤ xmax, ymin ≤ y ≤ ymax, using a plotter step of
size step;
begin
  integer i, n, nstep, nexits, nroots, maxnsteps;
  real x, y, epsilon, lbx, ubx, lby, uby, xstep, ystep, temp, min;
  array xexit, yexit[1:50], root[1:10], xinc, yinc[2:3];
  procedure drawcurvesintersecting side (u, v, w, wmin, wmax,
    xstep1, ystep1, xinitial, yinitial);
  value wmin, wmax, xstep1, ystep1;
  real u, v, w, wmin, wmax, xstep1, ystep1, xinitial, yinitial;
  comment draw curves intersecting side wmin ≤ w ≤ wmax,
  where w is either x or y, and whichever of x or y is not w being
  constant. If translating into FORTRAN, note the use of the
  ALGOL call by name facility to control whether x or y is being
  varied. A corresponding FORTRAN program might include a
  LOGICAL parameter to indicate the direction of the side, with
  IF statements at the appropriate points in the subroutine, where
  the variables are here referenced by name;
  begin
    real x, y;
    findroots(f(u,v)-contour,w,wmin,wmax,root,nroots);
    for n := 1 step 1 until nroots do
    begin
      x := xinitial; y := yinitial;
      for i := 1 step 1 until nexits do
      if abs(x − xexit[i]) + abs(y − yexit[i]) < 5 × step
      then go to ENDLINE;
      comment contour is to be drawn;
      movepento(x, y); xstep := xstep1; ystep := ystep1;
      for nstep := 1 step 1 until maxnsteps do
      begin
        if xstep ≠ 0 ∧ ystep ≠ 0 then
        begin xinc[2] := xstep; yinc[3] := ystep;
          yinc[2] := xinc[3] := 0
        end else
      begin xinc[2] := yinc[3] := xstep + ystep;
        xinc[3] := xstep − ystep; yinc[3] := −xinc[3]
      end;
      min := abs(f(x + xstep, y + ystep) − contour);
      for i := 2, 3 do
      begin temp := abs(f(x + xinc[i], y + yinc[i]) −
        contour);
        if temp < min then
        begin min := temp;
          xstep := xinc[i]; ystep := yinc[i]
        end
      end of finding which of 3 directions to take;
      x := x + xstep; y := y + ystep;
      graphstep(xstep, ystep);
      if x < lbx ∨ x > ubx ∨ y < lby ∨ y > uby then
      begin nexits := nexits + 1;
        xexit[nexits] := x − xstep; yexit[nexits] :=
          y − ystep;
        go to ENDLINE
      end
    end nstep;
ENDLINE: end n
  end of drawcurvesintersecting side;
  nexits := 0; maxnsteps := 2 ×
    (xmax + ymax − xmin − ymin)/step;
  epsilon := step/10; comment to allow round-off error;
  lbx := xmin − epsilon; ubx := xmax + epsilon;
  lby := ymin − epsilon; uby := ymax + epsilon;
drawcurvesintersectingside(x, ymin, x, xmin, xmax, 0·0, step,
  root[n], ymin);
drawcurvesintersectingside(x, ymax, x, xmin, xmax, 0·0, −step,
  root[n], ymax);
drawcurvesintersectingside(xmin, y, y, ymin, ymax, step, 0·0,
  xmin, root[n]);
drawcurvesintersectingside(xmax, y, y, ymin, ymax, −step, 0·0,
  xmax, root[n])
end;
```

Appendix 3 Bicubic spline coefficients

This procedure calculates the coefficients of the bicubic spline approximation to a function given on a rectangular mesh. The procedure applies the conventional one-dimensional spline approximations first in the x direction, then in the y direction. The one dimensional procedure *splinecoeffs* used and given below is an immediate adaption of Splinefit, Späth's Algorithm 40 (1969), the adaptation being necessary to return the cubic spline coefficients rather than the values.

```
procedure bicubic splinecoefficients (x, nx, y, ny, z, c);
value nx, ny; integer nx, ny;
array x, y, z, c;
comment When given the values z[i, j] of a function at the point
  x[i], y[j] on a rectangular mesh with 1 ≤ i ≤ nx, 1 ≤ j ≤ ny,
  this procedure calculates in c[i, j, r, s] the coefficients of $x^r y^s$
  in the bicubic spline approximation to z, valid for the rectangle
  x[i] ≤ x ≤ x[i + 1], y[j] ≤ y ≤ y[j + 1]. The array de-
  claration for c should be (at least) c[1:nx − 1, 1:ny, 0:3, 0:3]
  although the answer is only given for the second subscript in the
  range 1:ny − 1. If converting to FORTRAN it is suggested
  that, because of the absence of call by name, before the two
  calls to splinecoeffs the input vectors x[i, j] for i varying and
  c[i, j, r, 0], varying j respectively should be copied into arrays.
```

```
    Similarly the output arrays c[i, j, r, 0] varying i, r, and
    c[i, j, r, s] varying j, s, respectively, should be copied from
    arrays after the call;
begin integer i, j, r, s;
  procedure splinecoeffs (y, m, x, n, coeff, r);
  value n; integer m, n, r;
  real y, coeff; array x;
  comment y is really an input array—it is a function of m, and
    coeff is really an output array the element selected depending
    on m and r;
  begin
    integer i, j, k, n1, n2; real oldy, sk, sum, sumx, px, z;
    array f2, h, dy[1:n], s[1:n − 1], e[1:n − 2];
    n1 := n − 1; n2 := n − 2;
    m: = 1; oldy := y;
    for m := 2 step 1 until n do
    begin h[m − 1] := x[m] − x[m − 1];
      dy[m − 1] := y − oldy; oldy := y
    end;
    f2[1] := f2[n] := 0;
    for i := 1 step 1 until n1 do
    f2[i] := dy[i] − dy[i − 1];
    z: = 0·5/(h[1] + h[2]); s[1] := −h[2] × z;
    e[1] := f2[2] × z; k := 1;
    for i := 2 step 1 until n2 do
    begin
      j := i + 1; z := 1·0/(2·0 × (h[i] + h[j]) + h[i] ×
        s[k]);
      s[i] := −h[j] × z; e[i] := (f2[j] − h[i] × e[k]) ×
        z;
      k := i
    end;
    f2[n1] := e[n2];
    for i := n2 step − 1 until 2 do
    begin
      k := i − 1; f2[i] := s[k] × f2[i + 1] + e[k]
    end;
    for m := 1 step 1 until n1 do
    begin
      i := m + 1; sk := (f2[i] − f2[m])/h[m];
      r := 3; coeff := sk;
      sum := f2[i] + 2·0 × f2[m] − x[m] × sk;
      sumx := x[m] + x[i]; px := x[m] × x[i];
      r := 2; coeff := sum − sumx × sk;
      r := 1; coeff := dy[m] − sumx × sum + px × sk;
      r := 0; coeff := y − x[m] × dy[m] + px × sum
    end
  end of splinecoeffs
  start of main procedure;
  for j := 1 step 1 until ny do
  splinecoeffs (z[i, j], i, x, nx, c[i, j, r, 0], r);
  for i := 1 step 1 until nx − 1 do
  for r := 0 step 1 until 3 do
  splinecoeffs (c[i, j, r, 0], j, y, ny, c[i, j, r, s], s)
end of bicubic spline coefficients;
```

Appendix 4 Experimental results

The table below summarises a numerical assessment performed on an ICL 1907 computer of a variety of interpolation methods applied to five different mathematically defined surfaces.

Surface	*Equation*	*Comment*
S1	$(x - 5{\cdot}5)^2 + (y - 5{\cdot}5)^2 + z^2 = 64$	A part of a sphere
S2	$z = exp - ((x - 5)^2 + (y - 5)^2)$	A steep hill rising from a plain
S3	$z = exp - \frac{1}{4}((x - 5)^2 + (y - 5)^2)$	A less steep hill
S4	$z = exp - ((x + y - 11)^2 + (x - y)^2/10)$	A long narrow hill
S5	$z = tanh\,(x + y - 11)$	A plateau and plain separated by a steep cliff

For each of these surfaces 100 points were generated on a regular 10×10 grid ($x, y = 1, \ldots, 10$), and a variety of interpolation methods used to estimate the surface heights at 50 intermediate points. The average deviations from the true heights are tabulated for each of the following methods, the first six of these only being applicable because the data were on a regular grid:

M1:	The 'classical' polynomial interpolation
M2:	Degree 3 spline
M3:	Hyperbolic interpolation, the weighted average of the four nearest points
M4:	Quadratic interpolation, fitting a quadratic in x and y exactly through the four nearest points, and approximately to the next eight.
M5:	Quadric interpolation as suggested by the referee, described in Section 2.
M6:	Weighted quadric interpolation, using a weighted average of four results applying M5 to the four nearest points, to ensure continuity of the contours
M7-M15:	Distance weighted least squares polynomial fit using polynomials and weighting functions $w(d^2)$ as described.

	polynomial	*weighting function*
M7	a (Shepard's method)	
M8	$a + bx + cy$ (all linear terms)	
M9	$a + bx + cy + dxy$	
M10	$a + bx + cy + dxy + ex^2 + fy^2$	e^{-d^2}/d^2
M11	quadratic terms $+gx^2y + hxy^2$	
M12	all 10 terms up to degree 3	
M13	all 15 terms up to degree 4	
M14	all quadratic terms	$1/d^2$
M15	all quadratic terms	$1/d^4$

	Surface				
Interpolation method	S1	S2	S3	S4	S5
M1	0·00000	0·02887	0·00077	0·09546	0·01292
M2	0·00111	0·00136	0·00064	0·00395	0·00416
M3	0·01541	0·01309	0·01069	0·03258	0·02400
M4	0·00029	0·00955	0·00364	0·02483	0·00942
M5	0·00000	0·01305	0·01051	0·02242	0·02142
M6	0·00000	0·00768	0·01293	0·01771	0·01789
M7	0·02030	0·01678	0·01920	0·03185	0·02813
M8	0·01712	0·01212	0·00944	0·02933	0·02603
M9	0·01718	0·01196	0·00950	0·02956	0·02575
M10	0·00034	0·00832	0·00539	0·01800	0·00811
M11	0·00029	0·00767	0·00500	0·01568	0·00757
M12	0·00026	0·00734	0·00277	0·01877	0·00835
M13	0·00002	0·00563	0·00158	0·01064	0·00448
M14	0·00305	0·02979	0·03033	0·04980	0·06840
M15	0·00057	0·01256	0·01687	0·02750	0·01977

Average deviation found between original and interpolated surface

We may conclude that in general, for these surfaces, the bicubic spline interpolation method M2 is the most accurate if it can be applied, i.e. if the data are regularly spaced. Of the distance-weighted least-squares methods M7-M15, the method M10 recommended in the paper, using a quadratic polynomial with weighting function e^{-d^2}/d^2 appears satisfactory; although some others (e.g. M13) are slightly more accurate, this gain is not felt to justify the extra computing involved.

References

AHLBERG, J. H. (1970). Spline approximation and computer-aided design, Alt, F. L. and Rubinoff, M. *Advances in computers*, Vol. 10, Academic Press, p. 275.

BENGTSSON, B. E., and NORDBECK, S. (1964). Construction of isarithms and isarithmic maps by computers, *BIT*, Vol. 4, p. 87.

BOOR, C. DE. (1962). Bicubic spline interpolation, *J. Math. and Phys.*, Vol. 41, p. 212.

COONS, S. A. (1969). *Surfaces for computer aided design of space forms*, MAC-TR-41, Massachusetts Institute of Technology.

COTTAFAVA, G., and LE MOLI, G. (1969). Automatic contour map, *CACM*, Vol. 12, p. 386.

GREVILLE, T. N. E. (1967). Spline functions, interpolation and numerical quadrature, Ralston, A. and Wilf, H. S. *Mathematical methods for digital computers*, Vol. 2, Wiley, p. 156.

MORSE, S. P. (1969). Concepts of use in contour map processing, *CACM*, Vol. 12, p. 147.

NONWEILER, T. R. F. (1968). Algorithm 326. Roots of low-order polynomial equations, *CACM*, Vol. 11, p. 269.

PARTRIDGE, M. F. (1968). Algorithm for drawing ellipses or hyperbolae with a digital plotter, *The Computer Journal*, Vol. 11, p. 119.

PITTEWAY, M. L. V. (1967). Algorithm for drawing ellipses or hyperbolae with a digital plotter, *The Computer Journal*, Vol. 10, p. 282.

RAE, A. I. M. (1966). A program to contour Fourier maps by use of an incremental CRT display, *Acta Crystallography*, Vol. 21, p. 618.

RIVLIN, T. J. (1969). *An introduction to the approximation of functions*, Blaisdell, Waltham Mass.

ROBERTSON, J. C. (1967). The Symap program for computer mapping, *Cartographic J.*, Vol. 4(2), p. 108.

SCHMIDT, A. J. (1966). *Symap: A user's manual*, Tri-County Regional Planning Commission, Lansing, Michigan.

SHEPARD, D. (1965). A two-dimensional interpolation function for irregularly spaced data. *Proc. 23rd Nat. Conf. ACM*, p. 517.

SPATH, H. (1969). Spline interpolation of degree three, *The Computer Journal*, Vol. 12, p. 198.

Book review

The Art of Computer Programming, *Volume 3*, *Sorting and Searching*, by Donald Knuth, 1973; 722 pages. (*Addison-Wesley*, £9·30.)

This thick volume is the third of a projected series of seven. Like the two earlier volumes the exposition is heavily weighted towards mathematics, but there are perhaps fewer examples of results being introduced into the text solely for their mathematical interest. A better title to the series might have been 'A Mathematical Approach to Computer Programming'; certainly the title must not be taken as implying that the author considers that, in the common phrase, 'programming is more an art than a science'.

The first half of the book deals with sorting. This was the subject of much research in the period around 1960-65. With the results of this research scattered in the literature and requiring cross evaluation before they could be used, a compendium was much needed. Now anyone with a sorting problem to solve would be foolish not to look in this book first. As Dr. Knuth tells us: 'unfortunately there is no known "best" way to sort; there are *many* methods, depending on what is to be sorted on what machine for what purpose'.

Methods of internal sorting are dealt with first; by this is meant the sorting of information wholly contained within the high speed memory. Twenty-five different algorithms are described and evaluated. In each case an English language description of the algorithm is given, followed by a flow diagram and a program written in MIX, the author's synthetic assembly language. Finally there is an example of numbers sorted according to the algorithm. No use is made of high level languages for describing algorithms.

Methods discussed under the heading of internal sorting are, by themselves, virtually useless when the amount of information to be sorted is greater than can be held in the high speed memory. It is then that merging comes into its own and most of the section in the book on external sorting is concerned with variations on the theme of internal sorting followed by merging. People who are accustomed to use such methods in business data processing may, perhaps, be surprised to find how much theoretical discussion is necessary to evaluate the various alternatives.

The second half of the book is concerned with searching. The author remarks that this section might have been given the more pretentious title 'Storage and Retrieval of Information' or the less pretentious title 'Table Look-up'. The sub-headings are: sequential searching; searching by comparison of keys; digital searching; hashing; retrieval on secondary keys. The problem discussed is simply stated; it is how to find data that has been stored with a given identification. 'Searching is the most time consuming part of many programs and the substitution of a good search method for a bad one often leads to a substantial increase in speed'. The author was right in choosing a non-pretentious title for this part of the book, since he was concerned with the design of algorithms and not with database technology in any general sense. There is much more to the design of management information systems, for example, than is discussed here; in particular, the reader will find no reference to the work of the CODASYL Data Base Task Group or to relational data bases. A similar remark can be made about systems for information retrieval, as that subject is understood in the world of library and information science.

The book is liberally provided with exercises classified according to the amount of mathematical knowledge needed and according to difficulty. They are intended for self study as well as for classroom study and the author has devoted 130 pages at the end of the book to giving solutions. The book is to be strongly recommended to systems programmers and implementers of application packages. It will be indispensible to those who have to plan courses in computer science that cover the topics of which it treats. As a source of inspiration for lecturers and their abler students it could hardly be bettered.

M. V. WILKES (Cambridge)