A NUMERICAL METHOD FOR LEVEL CURVE PLOTTING

Jouni Freund

Rakenteiden mekaniikka Vol. 22
No 3 1989 s. 3...24

SUMMARY: A numerical method for calculation of level curves is presented. In the method the level curves are divided into a set of straight line segments, which can be plotted easily. The division is made in such a way, that a smoothness criterion is satisfied with as few line segments as possible. General aspects of finding level curves in an arbitrarily shaped region are discussed. Thereafter three toolbox algorithms, which can be used in solving the problem, are presented. The toolbox approach makes it possible to use the algorithms also in related problems, as in plotting the element mesh in finite element applications. Finally two simple plotting problems are solved using a program based on the algorithms.

INTRODUCTION

Examples of level curves can be found in maps in the form of contour lines ; i.e. curves with equal distance from the sealevel. Clearly the level curves serve as a simple and easily comprehensible way to present three dimensional forms in two dimensions. A formal definition would read : A level curve for a function of two variables is the set of points, where the function has a given value.

If the level curve equation is linear in the independent variables, level curves can be obtained easily by solving for one of the vari-

ables. In practice it is not always possible to find the level curve equation for a given function in analytical form. The reason is simply that there exists no general method for solving nonlinear equations. Other line of difficulties comes from the fact that analytical solutions - if obtained - are often multiple valued. That is, for a given function value and one coordinate value there are several other coordinate values satisfying the equation. Thus one must face the difficult task of choosing the right one. Furthermore it is often demanded that the level curves are confined to a given region. Because of these complications one is assured that finding an analytical solution in a complicated region is beyond the reach. However, numerical methods can be employed no matter the function or region type. The nonlinear nature of the problem prohibits the construction of a foolproof method, but with a proper algorithm and with enough computational power one can calculate level curves for almost any function.

Two different numerical methods are presented in references / 1 / and / 2 /. The method in / 2 / is based on dividing the region of interest into smaller regularly shaped subregions. The division is refined until the level curves intersects the boundaries of each subregion just twice. Once a subdivision with the above mentioned properties is found, the first approximation of the level curves is obtained easily by connecting the intersection points in each subregion with a line segment. The other method referred to is based on direct tracing of the curve. Namely if one point on the curve is known one can proceed along the curve by taking into account the fact that the gradient of the function is orthogonal to the curve. Just take a step perpendicularly to the gradient and another point is found, save a small error, which can be corrected as indicated in / 1 /.

The scheme to be adopted here is basically similar to the tracing method in / 1 /. In spite of the same underlying idea the algorithms are quite different because the present method is intended to handle problems of a more general type. The main features not found in / 1 / are: 1) Automatic step length selection, which allows smooth curve representation with minimum data. 2) Proper handling of difficult situ-

ations encountered in tracing. 3) Avoidance of multiple tracing of same curves by two simple rules.

PROBLEM DEFINITION

The aim is to construct an algorithm, which can be used in plotting level curves for an arbitrary function in an arbitrarily shaped bounded region in the plane described with rectangular Cartesian x,y-coordinates. To make the problem easier to handle the region of interest is divided into possibly overlapping subregions called elements or *patches* to avoid possible confusion with elements in the finite element method sense, in which subregions are in general nonoverlapping. Furthermore in every patch the *curves* are divided into straight *line segments*. The primitive object is thus a straight line segment.

For each patch a local s,r-coordinate system is defined so that in the s,r-plane every pair of points can be joined with a straight line, which does not intersect the boundaries. Shortly patches are convex in the s,r-plane. In the x,y-plane the patch can be of any shape depending on the form of transformation between the two coordinate systems. It is tacitly assumed that there exists a mapping, which maps the original patch in the x,y-plane into a convex one in the s,r-plane. It is well known that for example conformal mapping provides a method for mapping almost arbitrarily shaped x,y-patch onto a unit square, which certainly is convex. In practice one can use for instance isoparametric mappings of the finite element literature.

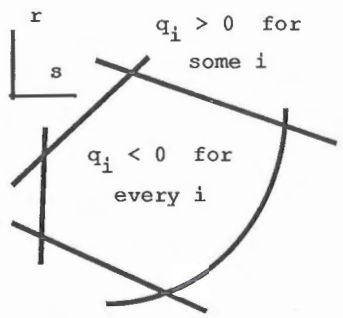

Figure 1. Typical patch.

A typical patch is shown in Figure 1. The boundary consists of piece-

wise smooth curves. Corresponding to each curve one defines a so called *boundary function* $q_i(s,r)$ in such a way that inside the patch the function value is strictly negative, outside the patch strictly positive and on the boundary exactly zero. The patch can be thus identified as the set of points where every boundary function has a negative value. Conversely the outside region can be identified as the set of points where at least one boundary function has a positive value. The boundary consists of all the points where at least one boundary function has zero value and every other boundary function a negative value.

From the point of view of numerical calculations it is advantageous that the both coordinate values are of the same order of magnitude in the patch. Thus before numerical treatment one should cast the problem into a dimensionless form with properly selected reference values. The dimensionless problem definition for one patch and one curve is as follows: Given functions

$$f = f(s,r) \quad , \tag{1}$$

$$x = x(s,r) \quad , \tag{2}$$

$$y = y(s,r) \quad , \tag{3}$$

$$q_i = q_i(s,r) \quad , \quad i = 1,2... \tag{4}$$

find a curve, on which function f has a given value f_0. All functions are supposed to possess first order derivatives inside a patch. The presentation is natural with functions calculated by the finite element method, but with a little effort almost any level curve plotting problem can be cast into this form.

Example. Let us consider a simple problem in which the patch in the x,y-plane is bounded by two rays emanating from the origin and two ori-

gin centered circles. The function to be plotted is $f = xy$. The region of interest is nonconvex whatsoever the angle between the rays, and a transformation must be used in order to cast the problem into a suitable form. One possibility is to identify the s,r-system as polar coordinates. Thus an equation set

$$f = r^2 \sin(s)\cos(s) \ , \tag{5}$$

$$x = r \sin(s) \ , \tag{6}$$

$$y = r \cos(s) \ , \tag{7}$$

$$q_1 = r - r_2 \ , \tag{8}$$

$$q_2 = r_1 - r \ , \tag{9}$$

$$q_3 = s - s_2 \ , \tag{10}$$

$$q_4 = s_1 - s \tag{11}$$

is obtained, where the meaning of the notations is obvious. For simplicity the equations are not made dimensionless as recommended above. Mapping (6) and (7) transforms the patch onto a rectangle in the s,r-plane. Substitution of expressions (6) and (7) into the original function to be plotted gives a representation of the form (1). The boundary functions (8) to (11) are easily obtained as the region is a rectangle in the s,r-plane. The boundary functions are not unique. For instance any boundary function (8) to (11) multiplied with a strictly positive function is still a boundary function but not necessarily as simple as the original one.

SOLUTION METHOD

Before proceeding to the algorithms the solution idea is discussed in detail. To simplify the presentation, it is assumed that the boundary of each patch consists of straight line segments in the s,r-plane. As a consequence the data needed in the construction of the boundary functions consists simply of a table of corner point values of the local coordinates. A typical boundary function is in this case the coordinate measured from the corresponding line segment to the outward normal direction to the boundary.

The level curves are *traced* by working first only in the s,r-plane as in / 1 /. Thereafter the curves in the s,r-plane are *converted* into the x,y-plane by using transformation functions (2) and (3). Thus in the finite element applications one can for example calculate the level curves for various actual x,y-patch shapes with just one time consuming number crunching in the s,r-plane. In some cases, as plotting patch boundaries, the presentation in the s,r-plane is known in advance and the tracing part can be skipped.

General

The notations used are given in Figure 2. The subscripts of the different local coordinate pairs refer to *first, old, intermediate* and *new.* The first point is simply the first point obtained on the curve. The old point is the origin of a local auxiliary rectangular Cartesian u,v-coordinate system, which is kept fixed until

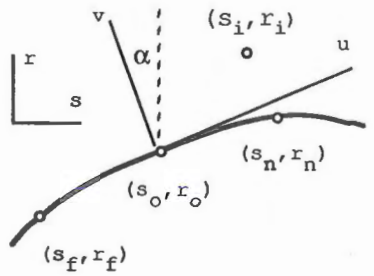

Figure 2. How to trace a curve; some notations.

the new point is found and then moved to a new position. In the beginning the first point and the old point coincide. The direction of the axes of the curve fixed coordinates u and v are in the tangential and the normal directions to the curve, respectively. The intermediate point is an intermediate result in the search for the new point on the curve somewhere onwards from the first point and the old point.

The connection between the s,r-system and the auxiliary u,v-system is

$$s = s_o + \cos(\alpha)u - \sin(\alpha)v \quad , \tag{12}$$

$$r = r_o + \sin(\alpha)u + \cos(\alpha)v \quad , \tag{13}$$

where the notations refer to Figure 2. Using the fact that the positive v-axis is selected to point into the direction of the gradient of function f, the sine and cosine of angle α can be written as

$$\cos(\alpha) = \left(\frac{\partial f}{\partial r}\right)_o \Big/ \sqrt{\left(\frac{\partial f}{\partial s}\right)_o^2 + \left(\frac{\partial f}{\partial r}\right)_o^2} \quad , \tag{14}$$

$$\sin(\alpha) = -\left(\frac{\partial f}{\partial s}\right)_o \Big/ \sqrt{\left(\frac{\partial f}{\partial s}\right)_o^2 + \left(\frac{\partial f}{\partial r}\right)_o^2} \quad . \tag{15}$$

The solution idea is as follows: First one point on the curve is searched for. The search is performed between two given points inside the patch. Because the convexity assumption the search cannot lead outside the patch. If the point is found the tracing starts into the positive u-direction. A step is made by giving the u-coordinate some predefined value. Then an iteration is performed along the v-direction in order to correct the error possibly made. Once a new point on the

curve is found it becomes an old point and the same procedure is repeated as far as possible. The points obtained on the curve are joined finally with linesegments.

The two points used in the search of the first point are selected with some general knowledge of function f in mind. In most cases there can be several level curves in the patch and the procedure must be repeated several times. Usually one directs the searches at least along the boundaries by taking two successive corner points of the boundary as starting and end points. Because of the tracing procedure (always to positive u-direction) curves intersecting the boundaries twice are, however, traced just once. One must only make sure that any curve can intersect a search line only once. The requirement can be satisfied easily at least in conventional finite element applications. If the function can form closed level curves one must use some search lines inside the patch too. The strict avoidance of multiplicity of the level curves is not simple but can be covered in most cases by the rule: Discard any curve ending at the boundary.

New v when u is fixed

For a fixed u-value the v-value, which corresponds to the point on the curve, is determined using Newton's method. Thus a new better value is obtained through the iteration

$$v := v - \left(f - f_0\right) / \left(\frac{\partial f}{\partial v}\right) , \tag{16}$$

in which notation := means substitution and f_0 is the function value to be traced. The expression for the partial derivative of f with respect to v is

$$\frac{\partial f}{\partial v} = - \frac{\partial f}{\partial s} \sin(\alpha) + \frac{\partial f}{\partial r} \cos(\alpha) \quad . \tag{17}$$

The method will converge to the right v-value if the initial guess for v is good enough. The choice of the origin of the u,v-coordinate system makes it possible to use a good fixed initial value v=0.

New u when v is fixed

The value of u is updated during the iteration. The updating is done partly because of the iteration scheme. Namely if the guessed point is too far from the curve, one can not be sure of the convergency any more. On the other hand the smoothness of the curve in the s,r-plane determines the suitable step length u. It is obvious that a small u-value should be used if the curvature is large and conversely a large u will do with more or less straight curves.

The absolute value of the cross product of the unit normal of the curve at the intermediate point and at the old point is taken as a measure of the direction change of the curve:

$$g(u,v) = \left| \vec{n}_o \times \vec{n}_i \right| \quad . \tag{18}$$

Actually the intermediate point is initially not necessarily on the curve but at some point nearby and the related normal direction given by the gradient is therefore only an estimation. For small function values the value of function g is approximately the same as the absolute value of the angle between the normals in the s,r-plane. If the value is between given lower and upper bounds during the iteration nothing is done. However, if the value is outside the bounds an interpolation

$$u := u \, \frac{g_l + g_u}{2 \, g(u)} \tag{19}$$

is performed in order to get a better value for u. The indices refer to lower and upper. The interpolation seeks for a u, which gives the

function the mean value of the bounding values. In the case of a straight curve the denominator is zero and the interpolation is modified slightly by replacing the zero function value by a small positive value. Obviously the interpolation works well provided the curvature changes of the curve are not considerable. However, one can easily sketch curves for which the updating scheme fails. Therefore the linear interpolation is overtaken by a foolproof method

$$u := u/2 \tag{20}$$

after some nonsuccessfull interpolations. This emergency method leads finally to a g-value below the upper bound because at the old point (corresponding to u = 0) the value of g is zero. Every time u is updated, v is given zero value and the iteration starts from scratch.

Out of the patch condition

Function f is defined only in the patch, which is determined by the boundary functions. Therefore when function f is evaluated one must make sure that the argument values are legal and correct them properly if the iteration procedure for v or the guess for u gives noncorrect values for the local s,r-coordinates.

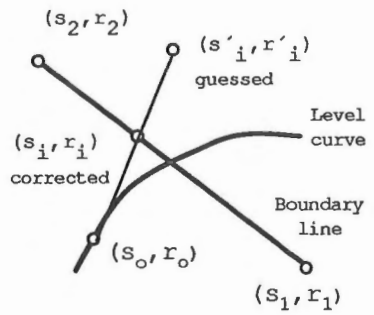

Figure 3. Intersection with boundaries.

If the calculated point is outside the patch, new s,r and u,v-values corresponding to the boundary are determined. Geometrically one moves backwards on a line between the old point (Figure 3) and the intermediate point. In practice each boundary function is checked in turn. Once a positive value

is detected, a correction is made using the following interpolation equations

$$s_i = (1-t)\ s_1 + t\ s_2 \ , \tag{21}$$

$$r_i = (1-t)\ r_1 + t\ r_2 \ , \tag{22}$$

$$s_i = (1-w)\ s_o + w\ s'_i \ , \tag{23}$$

$$r_i = (1-w)\ r_o + w\ r'_i \ , \tag{24}$$

in which the meaning of the different indices can be seen from Figure 3. Parameters t and w are scaled to make the values zero and one correspond to the end points of the lines. If the parameter values satisfying the equations are between zero and one, an intersection point has been found. Once the correction is made, further checking of the boundary functions can be rejected because the goal is achieved.

Curve end conditions

The curve is traced as far as possible. However, the boundary may prevent from going on, a saddle point may break the iteration or the curve may be extremely badly behaving at some point etc. All these cases have the common feature that the old point and the new point are very close together. Therefore the closeness of these two points is taken as one stopping criterion.

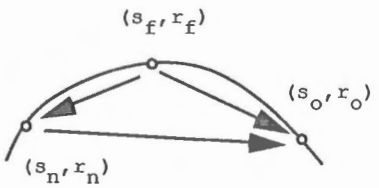

Figure 4. Closed curve condition

In order to keep the number of points needed in the description of

a curve at minimum, one must have some kind closed curve criterion too. Otherwise the tracing of the curve can be in principle a never-ending task. A simple method employed here is described in Figure 4. The curve is taken as closed if the lengths of the vectors in the figure joining the first point and the new point and the first point and the old point are both smaller than the length of the vector joining the new point and the old point. Altogether the stopping conditions are

$$\left|\vec{D}_{no}\right| \leq \varepsilon_{no} \ , \tag{25}$$

$$\left|\vec{D}_{fn}\right| < \left|\vec{D}_{no}\right| \quad \wedge \quad \left|\vec{D}_{fo}\right| < \left|\vec{D}_{no}\right| \quad . \tag{26}$$

In the equations $\left|\vec{D}\right|$ stands for the distance and ε_{no} is a small predefined parameter.

Conversion between s,r- and x,y-systems

The basic data to be converted is a straight line segment in the s,r-system. Depending on the coordinate transformation functions (2) and (3), however, the straight line is more or less curved in the x,y-plane. Therefore the linesegment given as

$$s = (1-w)\ s_1 + w\ s_2 \ , \tag{27}$$

$$r = (1-w)\ r_1 + w\ r_2 \quad . \tag{28}$$

is divided into smaller pieces. The indices in coordinates refer to the starting and end points of the segment and $w \in [0,1]$ is the curve parameter. The purpose is to find the w-values which provides a smooth enough representation in the x,y-plane. Because no assumptions con-

cerning the coordinate transformation (2) and (3) is made one must rely on an iterative method.

The conversion starts from the point labeled by 1, which is introduced in the beginning as the old point too. A new point labeled by n is calculated by giving the parameter w a small increment. The determination of a proper w value follows the lines sketched in the tracing part. A new better value for w is calculated from equations similar to (18) to (20) until the value of g function is satisfactory. If the value of function g is between the given lower and upper bounds, the new point becomes the old point and so on. Notations are shown in Figure 5. The indices in the x,y-coordinates refer to the values corresponding to the s,r-coordinates with the same indices.

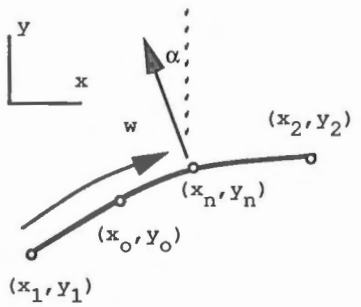

Figure 5. Conversion between the local and the global systems.

The conversion is stopped once the curve parameter has the value one corresponding to the point two. Of course the normal directions in equation (18) must be calculated in the x,y-plane. Using the chain rule one obtains

$$\frac{dx}{dw} = \frac{\partial x}{\partial s}(s_2 - s_1) + \frac{\partial x}{\partial r}(r_2 - r_1) \quad , \tag{29}$$

$$\frac{dy}{dw} = \frac{\partial y}{\partial s}(s_2 - s_1) + \frac{\partial y}{\partial r}(r_2 - r_1) \quad . \tag{30}$$

The sine and cosine of the angle α , which are the same as the components of the unit normal vector shown in the figure, can be now written as

$$\sin(\alpha) = -\left(\frac{dy}{dw}\right) / \sqrt{\left(\frac{dx}{dw}\right)^2 + \left(\frac{dy}{dw}\right)^2} \quad , \tag{31}$$

$$\cos(\alpha) = \left(\frac{dx}{dw}\right) / \sqrt{\left(\frac{dx}{dw}\right)^2 + \left(\frac{dy}{dw}\right)^2} \quad . \tag{32}$$

ALGORITHMS

The method described in the previous chapter is programmed using the FORTRAN language. The program consists of three toolbox subroutines which handle the number crunching part of the algorithm. The toolbox approach is preferred over a complete program because the form of the latter depends highly on the application.

Correction algorithm

The correction subroutine consists of a loop-escape construction over the boundary lines as shown in Figure 6. Once a boundary line is found, which intersects the line between the old point and the intermediate point, an escape is made and new corrected values for the coordinates are calculated.

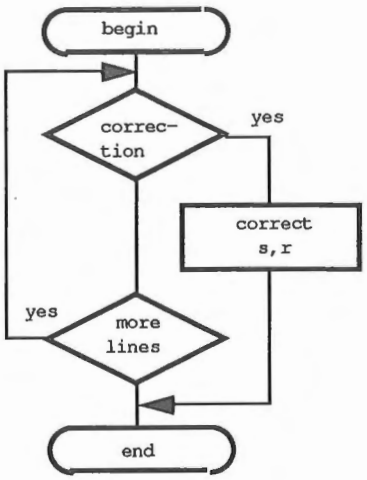

Figure 6. Correction algorithm.

The decision about the correction is made using the calculated t and w values from equations (21) to (24). In practice one must solve a linear set of equations for two variables. Obviously the correction should be made only if the values of t and w are both between zero and one.

Tracing algorithm

The tracing algorithm is shown in Figure 7. The algorithm consists of two main parts. In the first part a point is looked for, where the

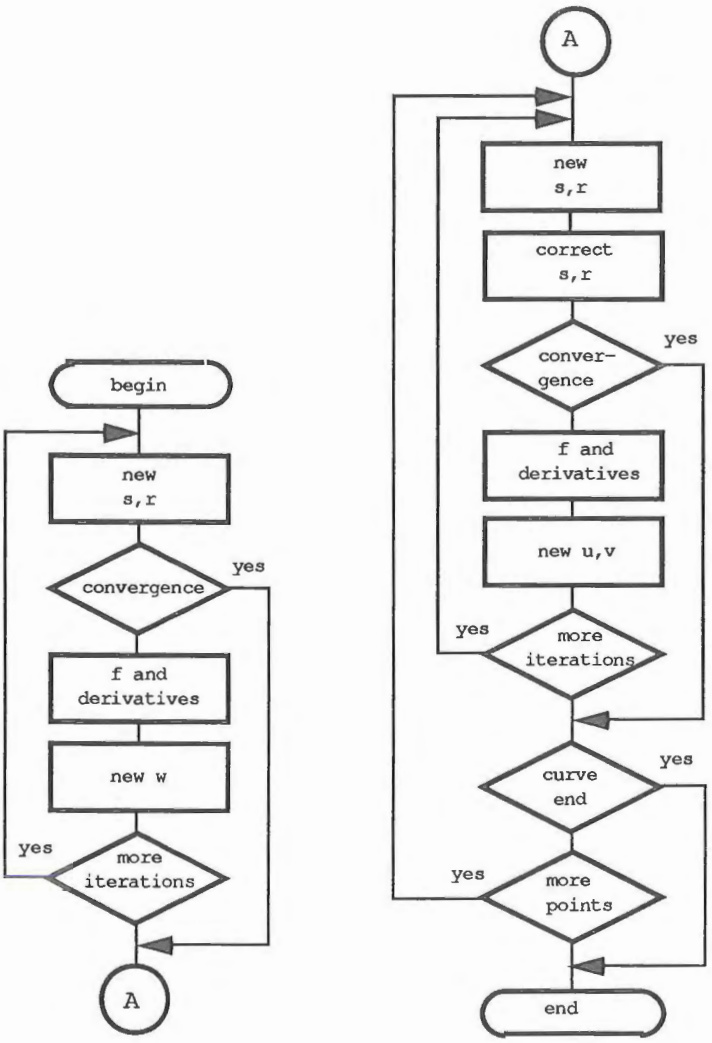

Figure 7. The tracing algorithm.

function has the given value. The search is made between two given points using Newton's method. If the point is found the tracing part of the subroutine is invoked but otherwise the second part is skipped. The first part consists of one loop escape construction over the maximum number of iterations. From the beginning: The new local coordinate values are evaluated using a linear representation as in equations (27) and (28) . The function value is obtained from equation (1). Finally a new value for w is calculated according to Newton's method. If the local coordinate values are more or less the same as the previous ones an escape is made, otherwise the iteration continues. The range of possible w values is confined between zero and one corresponding to the end points of the search line.

In the tracing part of the algorithm the local coordinate values are calculated from equations (12) and (13). The values obtained are checked and possibly corrected using the correction algorithm described in the previous chapter. Thereafter new function f value is calculated from equation (1). A decision is made whether the curve is smooth enough or should one decrease the step length. If the curve is regarded smooth enough a better v is calculated according to equation (16). Otherwise v is given the value zero and u is decreased appropriately using equations (18) to (20). In that unfortunate case iteration starts from the beginning. Convergence means that new values obtained for s and r are more or less equal to the previous values. If this happens more iterations are not needed. A curve end is detected if the proceeding is prevented somehow or the curve is closed. The check is made using inequalities (25) and (26).

Conversion algorithm

The algorithm used for the conversion between the s,r-system and the x,y-system is shown in Figure 8. The main loop is over the consecutive s,r-coordinate pairs given as input in the form of a table. Every successive coordinate pair in the table defines a line segment, which

is divided by the algorithm into smaller pieces if necessary.

From the beginning: The new local coordinate values are calculated from equations (27) and (28). The corresponding global coordinate values are obtained from equations (2) and (3). If the global coordinate values thus obtained meet the smoothness requirements, an escape is made. Otherwise a new better w value is calculated and iteration for acceptable x,y-coordinate pair continues. Once the end point of the line is detected (the curve parameter value is one), the next line segment is taken to be converted.

The calculated x,y-coordinate pairs are saved by the subroutine as a file in standard format. The output is buffered with buffer size given in the parameter block. The file consists of variable number of blocks. Every block consists of a title record, which is followed by the data. The title record itself consists of three integer numbers: Number of data records in the data block, number of data in a data record (fixed for a given block) and finally data type. The number, which defines the data type can be used in final processing of the plot data. One can, say, define element bound-

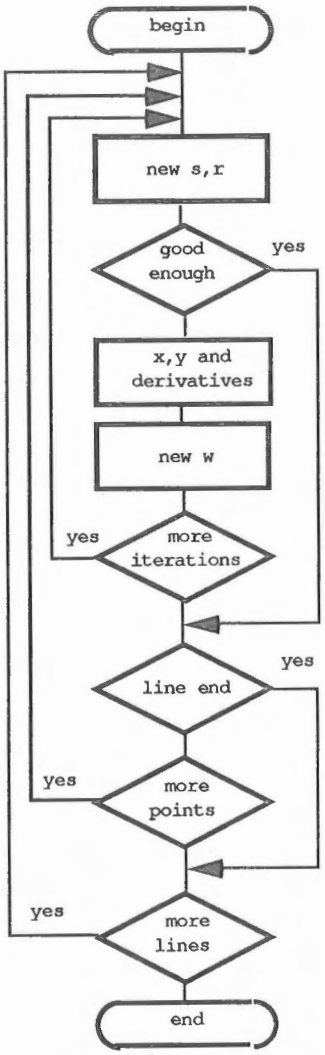

Figure 8. Conversion algorithm.

ary as different type of data than level curves and use different linetype in plotting.

EXAMPLES

Two simple plotting problems are solved using the subroutines based on the algorithms. The toolbox routines TRACER and CONVER are assembled into two higher level subroutines called LEVC02 and FRAM02. The names refer to level curve and frame drawing for element type 02. The numbering 02 refer to the fact that there can be any number of these subroutines, each specialized to handle some type of elements. The subroutines FRAM02 consists of a data statement, in which the table of corner point values in the s,r-system of a patch is given and a call for subroutine CONVER. The subroutine LEVC02 consist of two data statements and a loop over the search lines given in the other data statement. The other data statement is the same as in the frame drawing routine and the other one defines the search lines. Inside the loop is call for the tracing subroutine followed by call for the conversion subroutine.

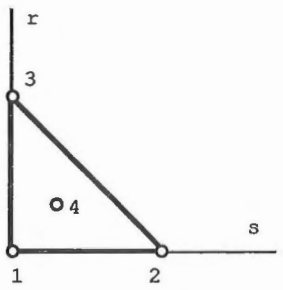

Figure 9. Definition of a patch.

The main programs used in the examples are very simple consisting only of a block, in which the data fixing functions (1) to (3) are read from the input file, and calls for subroutines FRAM02 and LEVC02. Actually the subroutines are meant to be an integral part of a finite element program. The output file generated by subroutine CONVER is plotted with the help of the DISSPLA subroutine library / 3 /. Smoothing or options like that available in DISSPLA are not used.

The shape of an element of type 02 is a triangle in the s,r-plane

as shown in Figure 9. At the corner points labeled 1,2 and 3 the values of the local coordinates are zero or one. The numbered points are called nodes or nodal points. The definition of this coordinate system and the following functions are standard in finite element method literature and the details concerning the representation will be skipped. In the both problems dealt with the functions involved are

$$f = (1-r-s)\ f_1 + s\ f_2 + r\ f_3 + rs(1-r-s)\ 27\ f_4 \quad , \tag{33}$$

$$x = (1-r-s)\ x_1 + s\ x_2 + r\ x_3 + rs(1-r-s)\ 27\ x_4 \quad , \tag{34}$$

$$y = (1-r-s)\ y_1 + s\ y_2 + r\ y_3 + rs(1-r-s)\ 27\ y_4 \quad . \tag{35}$$

Parameters f_i , x_i and y_i are the function and the x,y-coordinate values at the nodal points. The functions multiplying these nodal values are called shape functions. By varying the nodal values one can obtain different kind of level curves from straight to highly curved. In this case the search lines are from node one to node two, from node two to node three, from node three to node one and finally from node one to node four. Any curve can be found with the use of these four search lines.

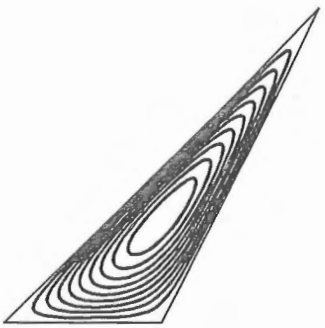

Figure 10. Closed level curves in just one element.

The first example is the calculation of nine closed level curves for f_0 values ranging from 0.1 to 0.9 and the boundary curve. Parameters x_i and y_i were given such values that the shape of the resulting triangle is rather irregular and the curvature changes in the level curves are considerable. The only nonzero func-

tion parameter is f_4 , which is given the value one. The parameters defining the smoothness of the curve were given values $g_l = 0.05$ and $g_u = 0.1$.

As can be seen from Figure 10 the behavior of the curves near the corners is abrupt, which makes the tracing of the curves not easy at all. The total number of calls for function f was 1520, and the total number of s,r-pairs 799. That makes about 2 evaluations for a point. The total number of x,y-pairs was 803. The number of points needed in the description of each curve was between 85 and 94 in the s,r- and x,y-planes. The near match of the number of points for each curve in the s,r-plane is not a surprise if one takes into account the u updating procedure. In fact every closed curve with nonreversing sign of the curvature will be described in principle with $2\pi/0.075 \approx 84$ points. The divider is the mean value of the lower and upper bounds given for function g.

In the second example the region consists of four nonoverlapping triangles of the type 02. The resulting level curves as well as element boundaries for certain selection of nodal values are shown in Figure 11. Each curve has a smooth representation within the elements and abrupt changes of direction are located at the element boundaries. The total number of function f evaluations was in this case 1591 and the total number of s,r-pairs needed in description of the curves 720. That makes about two evaluations for a point as in the first example. The total number of x,y-pairs was 523. The number of x,y-points is smaller than the number of s,r-points because some curves are traced twice but discard-

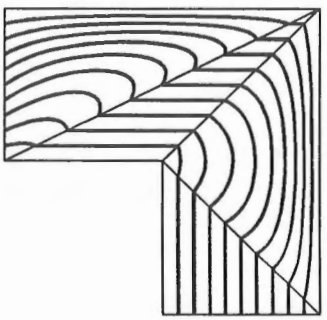

Figure 11. Level curves in the case of four elements.

ed because the curve found in the search from node two to node four has ended at the boundary. Every straight curve confined to one element is described by two points. The curved ones are described by 10 to 36 points in the s,r- and x,y-planes.

CONCLUDING REMARKS

With the information gained from the examples one can conclude that in the method described the deciding factor determining the computational effort is the complexity of the function rather than the number of patches used. Thus to plot a simple function in a complicated region takes less time than to plot a complicated function in a simple region.

After all the algorithm proved to work well with simple as well as with complicated functions. The computational effort measured with calls of function f is as good as one could hope. No optimization of the performance of the subroutines was done and probably the execution could be speeded up somewhat. The optimization, unfortunately, has usually an unwanted side effect of making the program hard to read.

The listings of the programs described above are obtainable from the writer on request.

ACKNOWLEDGEMENTS

I would like to thank Associate Professor Eero-Matti Salonen for valuable advice during the iterative writing process.

REFERENCES

[1] Akin, J. E. , Application and implementation of Finite Element method, Academic Press, 1982.

[2] Köliö, J. , Rakenteiden Mekaniikka, Vol. 21 No 3, 1988.

[3] Integrated Software Systems Corporation , DISSPLA.

Jouni Freund, assistentti, Teknillinen Korkeakoulu, Mekaniikan laboratorio.