1982

Numerical Computation with General Two Dimensional Domains

John R. Rice
Purdue University, jrr@cs.purdue.edu

Report Number:
81-416

Rice, John R., "Numerical Computation with General Two Dimensional Domains" (1982). *Department of Computer Science Technical Reports*. Paper 340.
<https://docs.lib.purdue.edu/cstech/340>

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries.
Please contact epubs@purdue.edu for additional information.

NUMERICAL COMPUTATION WITH GENERAL TWO DIMENSIONAL DOMAINS

John R. Rice
Mathematical Sciences
Purdue University

CSD-TR 416
October 11, 1982

ABSTRACT

Consider a bounded two dimensional domain R with boundary ∂R . Applications which involve processing R include approximation, numerical integration and the numerical solution of partial differential equations. In each case the geometric information of R and ∂R must be put into a data structure (either implicitly or explicitly) as an initial phase of the problem solution. This paper presents one approach to this problem based on overlaying R by a rectangular grid. R is assumed to be defined by parameterization of pieces of ∂R plus an orientation. Specific representations of this information are given and a data structure presented which is convenient for using the geometric information in various applications. An algorithm to produce this information is given in a companion paper, "*Algorithm: A Two Dimension Domain Processor*"; various special techniques of the algorithm are described. The application to numerical integration is discussed in some detail.

1. THE DOMAIN PROCESSING PROBLEM

We consider a bounded two dimensional domain R in the x,y plane defined by its boundary ∂R with an orientation, clockwise or counter-clockwise. The interior of R is assumed to be on the right for clockwise and on the left for counter-clockwise. For simplicity, we assume the orientation is clockwise for the remainder of the discussion as there is no essential difference between the two cases. The boundary ∂R is specified in parameteric form as follows:

$$\begin{aligned} x &= x_i(p) \text{ for } b_{1i} \leq p \leq b_{2i} \\ y &= y_i(p) \text{ for } b_{1i} \leq p \leq b_{2i} \end{aligned} \quad i = 1, 2, \dots, nb$$

This defines nb pieces of the boundary which are assumed to be joined in order of the orientation; we also assume that $b_{2i} > b_{1i}$ for all pieces. Most other schemes for defining regions can be mechanically transformed into this one, the exception is to define R as the set $f(x,y) \geq 0$ which is mathematically appealing, but not widely used in practice.

We also allow holes in R , each is defined just as R except the orientation is reversed. Holes do not introduce a major complication in domain processing or the usual applications discussed later, so their presence is tacitly assumed in the discussion unless otherwise stated. Domains with several disconnected components are not considered here although they also introduce no major complications and could be handled by the procedures discussed here.

Many applications that involve such domains process them by introducing a "regular" subdivision of R . Triangles and rectangles are usually the basis of this process; this paper considers a rectangular grid G defined by the vectors

$$\begin{aligned} xgrid &= \{ a_1 < a_2 < \dots < a_{nx} \} \\ ygrid &= \{ b_1 < b_2 < \dots < b_{ny} \} \end{aligned}$$

This is a tensor product grid and it is assumed to overlay the domain R .

The problem addressed here is to determine the relationship between the grid G and the domain R and to produce data which allow one to carry out easily some common applications involving general domains. We are especially concerned about the applications of approximation, quadrature and the discretization of differential operators.

2. THE DOMAIN, GRID AND THEIR RELATIONSHIP

We first present the information that we have found to meet the application requirements for a domain processor. We switch to Fortran notation to describe the data structures involved. The domain R is defined with NBOUND boundary pieces specified by a subroutine, called BCOORD here, of the form

SUBROUTINE BCOORD(p,x,y,i)

which returns $x = x_i(p)$, $y = y_i(p)$ for the i -th piece. The ranges for the parameters are in the array BRANGE(k,i) with $b_{1i} = \text{BRANGE}(1,i)$, $b_{2i} = \text{BRANGE}(2,i)$. The grid G is defined by two arrays XGRID(j), $j = 1, 2, \dots, \text{NGRIDX}$ and YGRID(j), $j = 1, 2, \dots, \text{NGRIDY}$. These definitions are illustrated in Figure 1.

Figure 1. Example where R has four sides with one curved and where, in each variable, the grid is uniform on $[-1,6]$ with 7 intervals.

The relationship between the grid and domain is incorporated into a data structure with two parts. The first is associated with the grid points, points where the x and y grid lines cross and the second with the boundary points, points where ∂R intersects G . The first is a two dimension array

$GTYP(i,j)$, $i = 1$ to $NGRIDX$, $j = 1$ to $NGRIDY$

which types the grid points as follows:

<p>999 $0 < k < 999$ 0 $k > 1000$</p>	<p>interior to R and not next to ∂R the k-th boundary point exterior to R and not next to ∂R interior to R and next to ∂R. The relationship is encoded by $k = nb + 1000 * l$ where nb is the lowest numbered neighboring boundary point and l has bits to note locations of all neighboring boundary points as follows:</p> <p style="margin-left: 40px;"> $l = 0001$ - boundary neighbor to north $l = 0010$ - boundary neighbor to east $l = 0100$ - boundary neighbor to south $l = 1000$ - boundary neighbor to west</p>
<p>$k < -1000$</p>	<p>exterior to R and next to ∂R. The relationship between the grid point and the boundary is encoded in k as for $k > 1000$.</p>

In Figure 1, we see that $GTYP(4,4) = 3012$ which means that there are boundary points north and east of the (4,4) point and the lowest numbered one is 12.

The second part of the data structure is a set of seven 1-dimensional arrays of length $NBNDPT$ = the number of boundary points. Let p_i be the i -th boundary point and these arrays are:

<p>$XBOUND(i)$, $YBOUND(i)$ $BPARAM(i)$ $PIECE(i)$ $BPTYPE(i)$ $BNEIGH(i)$ $BGRID$</p>	<p>Coordinates of p_i Parameter value of p_i Piece to which p_i belongs (smaller index if there are two) type of p_i with five possibilities HORZ - on a horizontal grid line VERT - on a vertical grid line BOTH - both HORZ and VERT INTE - two pieces join not on a grid line JUMP - end of ∂R with a hole following pointer to interior grid point neighbors of p_i using the same encoding as l in $GTYP$ above $i + 1000 * j$ if p_i is in the grid with lower left point (a_i, b_j).</p>
--	--

Figure 2 shows the actual values in these data structures for the grid and domain in Figure 1. Experience suggests that these two data structures provide information in a form convenient for most applications involving domains and rectangular grids. If the desired information is not present, then it is easily generated from these data structures. For example, if one wants to know whether a

grid element is interior or exterior to R , one can test the GTYPE values of the corners of the element.

THE ARRAY GTYPE WHICH TYPES THE GRID POINTS.

8 *	0	0	0	0	0	0	0	0
7 *	0	0	-4005	-4006	-4007	-4008	0	0
6 *	0	-2005	5	8	7	8	-12008	0
5 *	0	-2004	4	9004	1007	1008	3008	-8010
4 *	0	-2003	3	8003	999	999	6011	-8011
3 *	0	-2002	2	12002	999	2013	-9012	0
2 *	0	0	-1001	-3017	12016	6014	-8014	0
1 *	0	0	0	0	-1016	-1015	0	0
.....	1	2	3	4	5	6	7	8
				X				

THE SEVEN ARRAYS WHICH RELATE THE BOUNDARY TO THE GRID.

POINT	XBOUND	YBOUND	BPARAM	PIECE	BPTYPE	BGRID	BNEIGH
1	1.00000	.50000	.00000	1	VERT	2003	0
2	1.00000	1.00000	.50000	1	BOTH	3003	2
3	1.00000	2.00000	1.50000	1	BOTH	4003	2
4	1.00000	3.00000	2.50000	1	BOTH	5003	2
5	1.00000	4.00000	3.50000	1	BOTH	6003	0
6	2.00000	4.00000	1.00000	2	BOTH	6004	4
7	3.00000	4.00000	2.00000	2	BOTH	6005	4
8	4.00000	4.00000	3.00000	2	BOTH	6006	4
9	5.00000	3.31173	.68826	3	VERT	5007	4
10	5.22500	3.00000	1.00000	3	HORZ	5007	8
11	5.25000	2.00000	2.00000	3	HORZ	4007	8
12	5.00000	1.50000	2.50000	3	VERT	3007	1
13	4.67500	1.00000	3.00000	3	HORZ	3006	8
14	4.10000	.00000	4.00000	3	HORZ	2006	8
15	4.00000	-.50000	4.50000	3	VERT	1006	1
16	3.00000	-.16666	1.00000	4	VERT	1005	1
17	2.50000	.00000	1.50000	4	HORZ	2004	2
18	2.00000	.16666	2.00000	4	VERT	2004	1
19	1.00000	.50000	3.00000	4	VERT	2003	0

Figure 2. The values of the data structure for the domain shown in Figure 1. The point XGRID(1), YGRID(1) is at the lower left of the GTYPE array.

3. COMPUTATIONAL PROCEDURES

A high level description of the domain processing follows:

LOCATE FIRST BOUNDARY POINT

```
LOOP OVER PIECES OF BOUNDARY
DO WHILE NOT TO END OF BOUNDARY PIECE
    FIND NEXT BOUNDARY-GRID INTERSECTION POINT
    DETERMINE TYPE FOR INTERSECTION POINT
CHECK CONTINUITY OF BOUNDARY
```

```
CHECK CLOSING OF BOUNDARY
ADD FIRST POINT TO END OF LIST OF INTERSECTION POINTS
```

```
PASS OVER GRID TO SET TYPES OF GRID POINTS
MARK INTERIOR POINTS
    LOCATE FIRST INTERIOR POINT
    EXPAND INTERIOR FROM FIRST POINT
SET POINTERS FROM BOUNDARY TO INTERIOR
```

If the domain being processed is a hole an enclosing domain, there is a final step to merge the results of processing the hole with those of the enclosing domain.

The methods used to implement the domain processing are fairly standard. By far the most crucial decision is the choice of balance between speed and reliability. A really fast algorithm can be developed for domains bounded by straight lines, but it will not work for most other domains. At the other extreme, given an algorithm and grid, one can construct a domain where the algorithm fails. The decision made here is to achieve very high reliability for "reasonable domains" and yet have "moderate efficiency" for simple domains. The terms "reasonable" and "moderate efficiency" do not have precise definitions, we attempt to give the reader an intuitive idea of their interpretation. Moderate efficiency means that about 5 to 10 boundary points are used to locate one boundary-grid intersection for simple domains. A reasonable domain is one where

1. the boundary does not enter a grid element several times,
2. the boundary is fairly smooth on the scale of the grid,
3. the parameterization of the boundary is made by smooth, monotonic and well-behaved functions,
4. there are no corners on the boundary except at the ends of boundary pieces,
5. the coordinates and parameters are all ordinary sized numbers and of similar size.

We now comment on items in the implementation which are unusual or special.

Machine dependence and geometric tolerance. The variable EPSGRD is to be large enough to insulate the computations from machine round-off. All points and lines within EPSGRD of one another are assumed to be equal. EPSGRD should be small enough that the accuracy in the application is not affected by an uncertainty of EPSGRD in the geometry. EPSGRD should be at least 20 units in the last place as convergence tests are fractions of EPSGRD (e.g. EPSGRD/5. in the secant method).

Identification of grid element to search. The search for the next intersection point is made at the four sides of a grid element containing the most

recently located point. A small increment is made in the boundary parameter which is then steadily decreased until a starting point is found in a grid element containing the grid point. The starting point is then moved until it is "near the middle" of the grid element located. The four sides of this grid element are the targets in the search for the next intersection.

Localization of the search. Several limits are placed on the boundary parameter p during the initial phase of the search. The most important is that p is never allowed to exceed the end of the current boundary piece. A short systematic search is made initially along the boundary to identify the first of the targets to be crossed, to order the targets for a more careful search and to provide parameter values which bracket the target. The algorithm is prepared for all of this to fail, but it does provide efficiency for easy domains.

Detection of vertical or horizontal boundaries. There is a small sub-algorithm to detect when the boundary is parallel to the grid lines. Detection of this situation increases efficiency (as this occurs commonly in practice) and makes the later searches more robust.

Modified regula falsi method. The subroutine REGULA uses a modification of the modified regula falsi method. If the target has not been bracketed, then the method makes a "move" in the boundary parameter toward the beginning or ending limit of the parameter. In this non-standard phase there is also a pseudo-random perturbation made to avoid obtaining the same point twice in succession during the iteration.

Secant method convergence. The modified regula falsi method is used with loose tolerance to obtain initial points for the secant method. It is felt that making the final search with the secant method provides more reliability than the non-standard modified regula falsi method. The convergence test for the secant method is one fifth the geometric tolerance EPSGRD. If convergence does not take place with this tolerance but a point is found within EPSGRD of the target, then this point is taken. The appropriate coordinate of the intersection point is set exactly equal to the grid line.

Double crossings of grid lines. There are at least four points on each boundary where the tangent is parallel to a grid line. Near these points there is a fair probability of two crossings of a grid line close together. If both points occur on one side of a grid element, it does not matter much which is found. If two such points are in adjacent grid squares, finding the second one is probably a fatal error as it is not in the desired grid element. To protect against this occurrence, a special test is made to detect such points whenever the secant method converges to a point outside the initial grid element.

Boundary tangent to grid lines. If the boundary is tangent to a grid line then there is a relatively long segment of the boundary which is much closer than EPSGRD to a grid line. After each intersection point is found the tangency test is made:

- (a) Is this point on the same grid line as the previous one?
- (b) Are the two points closer than $EPSTAN = 1.5\sqrt{EPSGRD} + (\text{minimum grid width})/10$.

When this test is passed the previous intersection point is replaced by the current one.

Location of domain interior. The orientation of the boundary is known so the domain interior is known to be on the right (we assume a clockwise orientation for this discussion) of the boundary. Thus one merely goes along the boundary and picks a point on the right. There is a real danger that a point picked is not actually interior if one is close to a corner. To pick an interior point reliably, a

conservative criterion is used first to select a point from which to pick an interior point:

- (a) The predecessor and successor points are not ends of pieces.
- (b) This point is not the end of a piece.
- (c) The predecessor and successor points are on different grid lines (unless this point is of type BOTH).
- (d) The boundary is not "tangent" to the grid lines at this point.

The entire boundary is searched for a point which satisfies all four parts of this criterion. If such a point is found, then the point on the right is taken as interior provided the GTYPE array indicates that this point has the boundary point as a neighbor.

If no boundary point satisfies all four parts of this criterion, then part (a) is removed and the boundary searched again. This technique has proved reliable except for very coarse grids where there are only one or two interior points per boundary piece.

Short word length machine environment. It can be delicate to use the domain processor on a 32-bit machine. One has to choose EPSGRD large enough to insulate the processing from round-off effects and small enough not to perturb the geometry of the application. We recommend $2 \cdot 10^{-5}$ as EPGRD for such machines. This suffices for applications requiring 3 or 4 digits of accuracy in the geometry. If all the variables are scaled nicely and the boundary parameterizations are well-behaved, one can obtain 5 digits of accuracy in the geometry, sometimes even more. Recall that the domain processor sets coordinates of intersection points to exact grid line values.

To alleviate this problem, we experimented extensively with the use of double precision in the computations. We assumed that the boundary points are computed in single precision and then all (or part) of the remaining computation was made in double precision. This approach was not effective in increasing the accuracy obtained and was abandoned.

4. APPLICATIONS

The domain processor was originally developed for discretizing general domains in ELLPACK, a system for solving elliptic partial differential equations. Its use with finite difference approximations - using either 5 or 9 point stencils - is rather obvious. It is also used for two finite element discretizations: one with collocation on rectangular domains and one with Galerkin on triangular domains. In the Galerkin case the rectangles are halved to obtain triangles and triangles are used along the boundary. In the collocation case the boundary elements are used "exactly"; collocation points for the boundary conditions are distributed along the boundary segments inside particular elements. The data structure created by the domain processor has been developed to meet the needs of this application.

One application of the domain processor to the discretization of partial differential equations is given in [Houstis, et al, 1982]. We do not discuss this further here and instead turn to other applications. The basic one is numerical integration and we outline an algorithm for accurate integration made of three components: quadrature on rectangles (in the interior), on triangles (along the boundary) and on small, nearly 1-dimensional domains. There is a variety of accurate methods for rectangles and triangles and while they must be selected compatibly, we ignore them for the moment. The situation along the edge of a domain is illustrated in Figure 3. Assume the problem is to estimate

$$\int_R f(x,y) = \sum_G \int_{\text{element in } G} f(x,y)$$

Existing methods can be used to integrate over interior rectangular elements (numbered 1, 2 and 4 in Figure 3) and combination rectangular/triangular elements (numbered 3, 5 and 6 in Figure 3). A basic assumption for the domain processor is that the grid is reasonably fine on the scale of the local variations in the boundary. This means that the relation between the boundary and the chord lines shown in Figure 3 is realistic, in fact, the boundary usually would be much closer to the chords than shown in Figure 3.

To compute the total integral, one must also compute the integral in the small irregular strips along the boundary. We call these boundary integrals. Those are from A to B; B to C and C to D in Figure 3. The boundary parameter p already goes along the boundary through these points and we use it as the independent variable. Thus, we must estimate, for example,

$$\int_{p=p_A}^{p=p_B} \int_{t=0}^{t=t(p)} f(x,y) dt dp$$

where t is perpendicular to the chord from A to B. The analysis is simplified if a change of coordinates is made to translate the origin to point A and rotate the x -axis to be along the chord AB. This introduces local coordinates x', y' related to x and y by

$$\begin{aligned} x &= x' \cos \vartheta + y' \sin \vartheta + \alpha \\ y &= -x' \sin \vartheta + y' \cos \vartheta + \beta \end{aligned}$$

where α, β and ϑ are easily computed. To simplify the notation, we assume this change of coordinates is made and continue to use x, y as coordinate variables. The boundary integral to be evaluated is then in the form

$$\int_{p=p_A}^{p=p_B} \int_{y=0}^{y(y(p))} f(x(p), y) dy dp$$

where a point on the boundary curve has coordinates $x(p), y(p)$. This is now an integral that can be estimated safely using the composition of two 1-dimensional integration rules, call them rule 1 and rule 2.

To estimate the accuracy required of the various integrations we assume that the integrals over rectangle elements are estimated using a tensor product q -point Gauss rule. Assume the grid is uniform with x and y spacing of h , then the local quadrature error on a rectangle is $O(h^{2q+2})$. The integrals over the combination regions are assumed to be estimated with the same accuracy. Less accuracy is needed for the boundary integral since the result is $O(h)$ compared to an integral on a rectangular grid. However, using q -point Gauss rules for both rule 1 and rule 2 in the boundary quadrature is still the most efficient approach even though it gives some unneeded extra accuracy.

Once integrals can be computed then one can do least squares approximation using standard methods. While approximations can be computed for any basis; the most natural choices are piecewise polynomials based on the grid.

An important variation of the least squares approximation application is two-dimensional data smoothing. One has a large set of data points scattered in

Figure 3. Close up of the boundary ∂R and grid G . The variation of the boundary is exaggerated here, in most cases it would vary smoothly and closely follow the chord lines shown.

R and wishes to smooth and replace the data. One applies least squares piecewise polynomial fitting, but the integration procedure outlined above must be replaced by one using the discrete point set of the given data. Developing such integration procedures is not easy, but is easier if the points are already grouped by the grid into neighborhoods of related points.

REFERENCES

E. N. Houstis, W. Mitchell and J. R. Rice, Collocation software for two dimensions, to appear (1982).

J. R. Rice, Algorithm: A two-dimensional domain processor, to appear (1982).