Journal of Sound and Vibration (1998) **209**(3), 443–472

EVALUATING BASINS OF ATTRACTION IN NON-LINEAR DYNAMICAL SYSTEMS USING AN IMPROVED RECURSIVE BOUNDARY ENHANCEMENT (RBE)

N. A. Alexander

Department of Civil Engineering, University of East London, Barking Campus, Dagenham RM8 2AS, England

(*Received* 16 *October* 1996, *and in final form* 23 *July* 1997)

An improvement to the computational algorithm known as Recursive boundary enhancement (RBE) is described. This updated algorithm produces global stability phase space diagrams in periodically forced differential systems. These equations being derived from the dynamics of engineering structures with non-linear responses. The algorithm uses a process of boundary grid refinement to produce a greatly enhanced procedure which is accurate and less computationally expensive than the standard *grid of starts* (GOS) method. The algorithm focuses on the boundaries of the catchment basins which need the most attention. This concentration on the boundaries cannot be made in an *a priori* manner as the boundaries are initially unknown. While the algorithm is proceeding concepts of parent cells, child cells and cell division are used to determine the location of the boundaries. The role of cell neighbourhood comparison is modified, in the improved algorithm, to provide a handle to control accuracy and computational speed. The necessity for recursion in the algorithm is discussed. The procedure is valid for both non-fractal and fractal boundaries. A comparison of the old and new RBE algorithms and other methods such as GOS, SCM and ICM mapping methods are made to evaluate computational efficiency and accuracy.

1. INTRODUCTION

In non-linear dynamics, the analysis of global instability phenomena such as the invariant manifold tangency induced boundary explosions [1–4] can result in events of far more relevance to engineers than the accurate knowledge of local bifurcation events. In a real life situation initial conditions are never precisely known and transient motions within a perhaps chaotic and fractal paradigm require a knowledge of catchment regions of various attracting sets. Thus it could be argued that events such as capsizes of boats [5] and other marine structures, such as floating *compliant* drilling platforms, owe more to the destruction of coherent global boundaries than to the local bifurcating demise of the attracting sets. The ideas presented here represent an automotive strategy to produce accurate portraits of the basins or regions of attraction with a great computational saving. The improvements to the basic algorithm presented in [6] represent an investigation of the role that the *comparative module* plays in the accuracy and computational speed of the Recursive Boundary Enhancement (RBE) algorithm.

0022–460X/98/030443 + 30 $25.00/0/sv971234

2. PRELIMINARIES

Consider a general non-linear harmonically driven differential oscillator

$$\ddot{x} + h(\dot{x}, x) = F \sin(\omega t).$$

This can be expressed as the following simultaneous differential equations

$$\dot{\underline{x}} = \underline{q}(\underline{x}, t), \qquad \underline{q}(\underline{x}, t) = \underline{q}(\underline{x}, t + T) \tag{1}$$

where $\underline{x}$ is a 2-tuple of state variables, $\underline{q}$ is a vector function of period T and t is the independent variable. The Poincaré section P can be defined as

$$P = \{(x, \dot{x}, t) \in R^3 : t = t_0 + iT, i \in Z\}.$$

From this periodic sampling of the solution space to (1) the Poincaré map can be defined as the following vector equation:

$$\underline{x}_{i+1} = \underline{f}(\underline{x}_i). \tag{2}$$

A periodic cycle is now represented by a point or set of points dependent on its periodicity. Thus, a period one solution point is one at which consecutive iterates of equation (2) result in identical values $x_{i+1} = x_i$.

The *grid of starts* (GOS) [1, 7] creates a rectangular grid of points across the Poincaré section P. Each of these grid points is used as an initial condition for the Poincaré map (2). Each iterate of (2) requires the differential system to integrate numerically using a Runga–Kutta type technique. Now if $\underline{x}_1 = \underline{f}(\underline{x}_0)$ and $\underline{x}_2 = \underline{f}(\underline{x}_1) = \underline{f}(\underline{f}(\underline{x}_0))$ which will be notated as $\underline{x}_2 = \underline{f}^{(2)}(\underline{x}_0)$, each point $\underline{x}_0$ is iterated n times to give

$$\underline{x}_n = \underline{f}^{(n)}(\underline{x}_0). \tag{2a}$$

In P each point $\underline{x}_0$ is mapped until it converges to some stable solution or infinity. Period one solutions can be located by this method while period s solutions require every sth iterate to be compared thus

$$\underline{x}_{sn} = \underline{f}^{(sn)}(\underline{x}_0). \tag{2b}$$

Chaotic (non-periodic solution) require their own special procedures for identification [7, 8]. Every stable solution can be numbered and thus each grid point can be coloured according to which solution it is attracted to. When this is completed for the entire grid, the resulting diagram describes the basins of attraction of the different attracting sets. The finer the grid the more accurate the resultant picture becomes, however the computational effort increases inversely with the square of the distance between the grid points.

3. BASIC CONCEPTUAL FRAMEWORK

The basic idea arises from its progenitor the GOS method. The basic procedure is described below.

(i) The region of phase space P that is of interest is divided into a coarse mesh of cells. The bottom left-hand corner of each cell is used as an initial condition. Equation (2b) is evaluated for each start and a pattern of catchment basins is built up in the standard way. If each attracting set is numbered then each cell can also be numbered according to which catchment basin it appears in.

(ii) Each cell, considered in a row by row fashion, is systematically compared with its neighbouring cells. The *basic comparative module* defines neighbouring cells as ones directly above and below, left and right. More complex comparative modules, which include

diagonal neighbours, are discussed in section 6 and constitute the main modification to the RBE algorithm. If neighbouring cells are in dissimilar catchment basins they are considered to be *on a boundary* and are marked as *boundary cells*. Cells that are in the same catchment basin as all their neighbours are considered *non-boundary cells*.

(iii) All cells are quartered: all cells can be thought of as parent cells producing children. Consider Figure 1: the 1a child cell takes its value directly from its parent requiring no evaluation of equation (2b). Child cell 1a is considered a good child cell. Child cells 1b, 1c and 1d however have to be evaluated somehow. For *non-boundary parent cells*, child cells 1b, 1c and 1d are approximated by taking exactly the same value as the parent cell. No evaluation of equation (2b) is performed here. In this case child cells 1b, 1c and 1d are considered bad child cells. For *boundary parent cells*, child cells 1b, 1c and 1d are calculated from equation (2b), they are not approximated and hence good child cells.

(iv) Each child of the *boundary parent cells* is systematically, in a row by row fashion, compared with its neighbouring child cells. If neighbouring child cells are in dissimilar catchment basins they are both considered to be *on a boundary* and are marked as *boundary cells*. Cells that are in the same catchment basin as all their neighbours are considered *non-boundary cells*. Children of *non-boundary parent cells* are considered to remain as *non-boundary cells* if not otherwise modified.

Basically each child cell has three properties: (1) whether it is *boundary* or *non-boundary*; (2) whether it is good or bad; (3) to which catchment basin it belongs.

(v) Children go on to become parents in their own right. Loop round to (iii) until the required resolution of the catchment basins and boundaries is achieved. The only caveat is when a bad *non-boundary child cell* becomes a parent all four of its children will be bad.

The great computational saving is made by the presence of bad child cells which are given approximated values for equation (2b). The greater the percentage of bad child cells the greater the computational saving over the ordinary grid of starts method.

In Figure 2(a) the four parent cells produce 16 child cells in Figure 2(b). As cells 2, 3 and 4 are boundary cells all child cells $2x$, $3x$ and $4x$, where $x \in$(a, b, c, d), are calculated from (2b), thus are good child cells. Only cells 1b, 1c and 1d are bad child cells. At stages (v) and (iii) only the 7 *boundary cells* are divided into four reliable good cells the other *non-boundary cells* are divided but approximate values of equation (2b) are used.

By the third generation of cells the results would be exactly the same as the GOS method with 64 cells. However only 34 cells would have been actually calculated, a 46% saving in the number of evaluations of equation (2b). This is just at the third generation, as the refinement of the cell mesh is increased, the saving over the conventional grid of starts is increased.

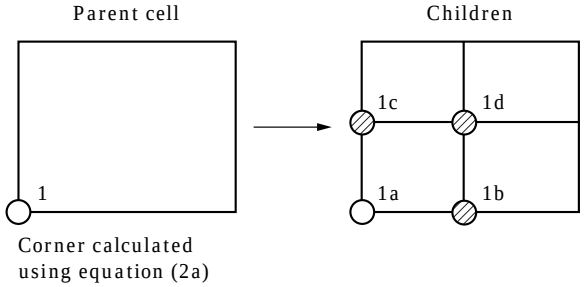

Figure 1. Cell division.

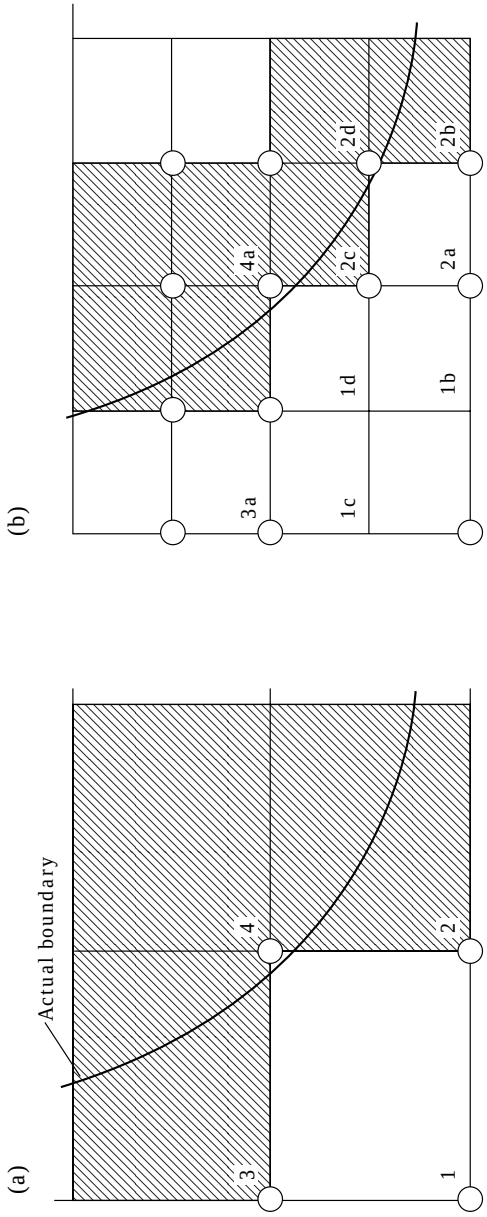

Figure 2. Cell generations. (a) 4 Parent cells, 4 good cells, 0 bad cells, 3 first generation boundary cells 2, 3 and 4. (b) 16 Child cells, 15 good cells, 3 bad cells, 7 second generation boundary cells.

4. NECESSITY OF A RECURSIVE ALGORITHM

The procedure in section 3 will produce a good approximation to the boundaries of a catchment basin but suffers from an algorithmic problem for certain boundary topologies.

Consider Figure 3(a). Six parent cells are evaluated and 3 *boundary cells* result from neighbourhood comparisons. Because of the nature of the boundary and the edge of the grid across region Poincaré section P, the parent cell 6 is not marked as a *boundary cell* (as cell 6's lower-left corner is on the same side of the boundary as its neighbours 5 and 4). In Figure 3(b) problems arise about the boundary region in both parent cells 6 and 5. Remember only cells marked as boundary cells are compared with their neighbours.

Observe child 4d and compare it with its neighbour 6b. Cell 6b parent 6 was a *non-boundary cell* thus 6b is a bad cell, with an approximated value. In fact 6b is considered to be in the same catchment basin as 6. Thus good cell 4d and bad cell 6b are thought to be on different sides of the boundary and hence 6b is marked as a *boundary cell* but for the wrong reason because cells 4d and 6b are in reality on the same side of the boundary. It would have been more beneficial if 6b was evaluated accurately by using equation (2b); converting a bad cell into a good cell. However even this modification is not sufficient because the 4d–6b comparison would now result in 6b being marked as a *non-boundary cell* as they are on the same side of the boundary. If the comparison 6a–6b was made, both 6a and 6b would be marked as *boundary cells*. However the 6a–6b comparison is never made as both 6a and 6b were marked as *non-boundary cells* because their parent 6 was a *non-boundary cell*.

Thus it is clear that the basic algorithm will suffer badly in certain situations. The problems with the above algorithm are: some cells are not marked as boundary cells and hence neighbourhood comparison is not made, when in fact this comparison is required; the presence of bad cells in the comparison is a source of potential error.

In the ideal scenario the 6a–6b comparison would be made, 6c, 6d and 5d would all be converted to good cells and all neighbourhood comparisons would be made for the identification of the next generation boundary cells. How can this be achieved?

Consider the following addition to the algorithm in section 3. This is now the basis of the RBE algorithm.

(iv.a) If two neighbouring cells are in dissimilar catchment basins but one cell is in fact a bad child cell (i.e., a cell with an approximate value for equation (2b)) then the bad child cell is evaluated exactly using equation (2b) and thus it is converted into a good child cell. Then the comparison which will identify the second/next generation *boundary cells* is always based on a comparison between good cells.

(iv.b) The comparison between cells in similar catchment basins and hence resulting in a non-boundary cell could be based on the good cell–bad cell comparison which may be in error. This good–bad cell comparison is algorithmically necessary as if it were not included then all the cells would be compared with their neighbours and be converted to good cells resulting in an algorithm very similar to the much simpler GOS method.

(iv.c) All newly calculated child cells are marked as *boundary cells* and immediately compared with all their neighbouring cells. This immediate comparison is required so that the sequence in which the boundary cells are compared is not a factor in the algorithm. In fact this means a *recursive* type procedure. As one cell becomes a *boundary cell*, it is immediately compared with its neighbourhood. Other cells may become *boundary cells* subsequently. Thus the process of addition of newly calculated cells will spread recursively until all necessary cells are identified and calculated correctly.

In Figure 3(b) cells 6b, 6c, 6d and 5d would all be converted to good cells and 6a, 6b, 6c and 5d marked as boundary cells by the above modification to the algorithm.

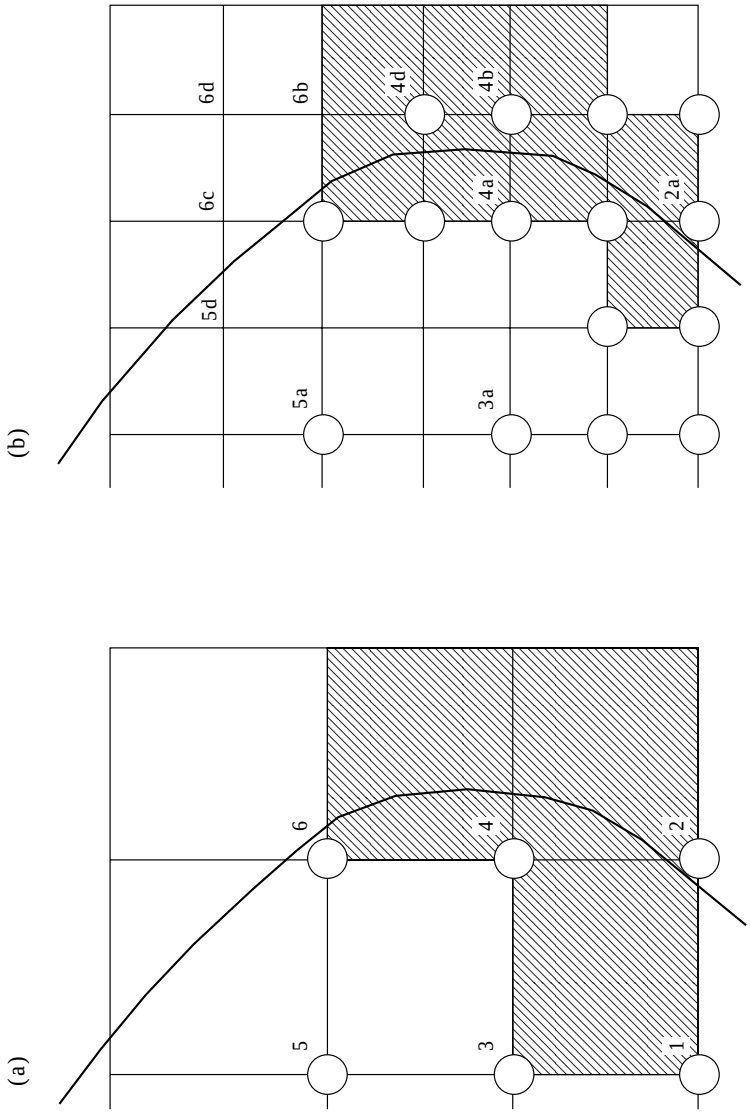

Figure 3. Problem topologies. (a) 6 Parent cells, 3 boundary cells. (b) 24 Child cells, 15 good cells, 9 bad cells? 8 + ? boundary cells.

5. FRACTAL AND NON-FRACTAL BOUNDARIES

For non-fractal boundaries and ones that are simply connected within the problem domain (the grid across the Poincaré section P) the RBE algorithm will produce exactly the same results as the GOS method at a fraction of the computational effort. This is providing one pair of boundary cells are located on each boundary. To achieve this does, in practice, require some knowledge of the dynamic system under investigation. This means defining the initial cell size for the RBE at such a value that this condition is satisfied. If the initial cell size for the RBE is large then there will be a larger number of cell generations and hence divisions resulting in a greater computational saving over GOS. However too large an initial cell size will result in missing certain boundaries which are not simply connected to other identified boundaries within the problem domain. This is where the initiation procedure cannot be totally automated. However it is worth noting that the real computational saving increases as the cell size reduces. Thus it makes more sense to err on the side of caution and start off with at least an 8×8 or even 16×16 grid. (Remember that this cannot be a general rule but is a guide and has been used for all the present studies.)

For fractal boundaries, ones that are not simply connected within the problem domain, the RBE algorithm works very well but suffers from certain problems when thin whiskers are present. Considering Figure 4(a), only cell 1 is actually inside the whisker boundary. In Figure 4(b) consider the cell 11d. This cell is also, in fact, inside the whisker boundary but because its parent 11 was not a boundary cell it is not evaluated. Note that the recursive procedure of section 4 would not cause it to be evaluated because cells 6a, 6b, 6d, 11a etc., cells between the known boundary cells and 11d, are all just on the wrong side (algorithmically) of the boundary and hence the algorithm will not proceed at this resolution to evaluate 11d. However as the cell's size is reduced this whisker would be successfully described. Simply speaking if the whisker width is less than the cell width then RBE may not pick up isolated cells such as 11d in Figure 4(b). Thus there will be a small difference (1/64 in Figure 4(b)) in the description of whiskers between the grid of the start method and RBE for a prescribed cell size when there are fractal boundaries in the problem domain.

6. RBE COMPARATIVE MODULES IN NUMERICAL STUDIES

The question of whom should be the "neighbouring cells" to a particular cell is a matter for discussion. In sections 3, 4 and 5 the rather arbitrary definition of those cells directly above and below, left and right is used and shall be subsequently known as the comparative module 1. Essentially the old RBE algorithm [6]. Flowchart Figure 12 shows the three modules investigated. Module 3 defines "neighbouring cells" far more widely than module 1 thus leading to a greater number of boundary cells. Module 2 includes diagonal cells in the neighbourhood but is a halfway-house between modules 1 and 3.

Numerical studies have been performed to assess the computational accuracy and efficiency of the various modules used in defining "neighbouring cells" in the RBE algorithm. Hence the modified RBE is compared with the GOS method and simple cell mapping method (SCM) of reference [9] for various modules. The dynamic system used for the testing is the Henon map:

$$x_{i+1} = A - x_i^2 - By_i$$

$$y_{i+1} = x_i. \tag{3}$$

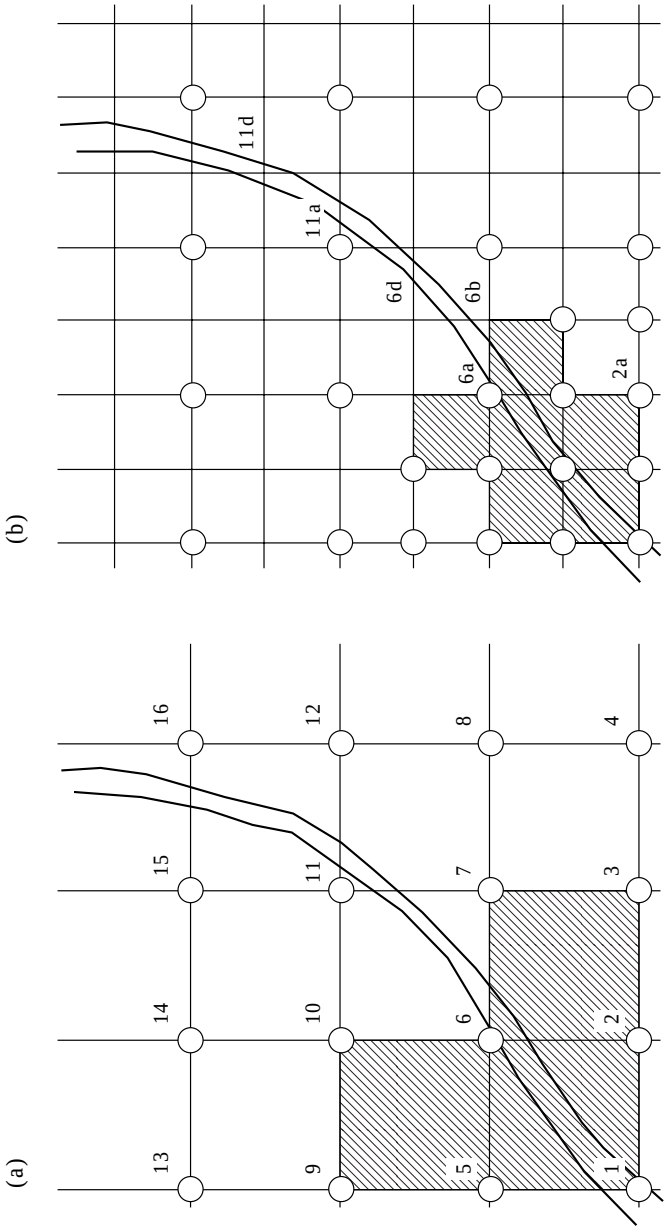

Figure 4. Thin whiskers. (a) 16 Parent cells, 3 boundary cells. (b) 64 Child cells, 25 good, 39 bad, 6 boundary cells.

The problem domain was $(-2{\cdot}5, -2{\cdot}5)$ to $(2{\cdot}5, 4)$. A bound to restrict exponentially large phase space co-ordinates was used at $|x_i| + |y_i| \leqslant 10$. Points that map outside this bound are considered to pass to an attractor at infinity. The Henon A parameter was 0·9. For RBE and GOS methods 10 iterates of (3) define the basin of attraction of the non-infinity attracting set. While for SCM 10 iterates of the cell mapping derived from (3) were used.

6.1. WHAT IS THE EFFECT OF FRACTAL AND NON-FRACTAL BOUNDARIES ON ACCURACY?

Consider Figure 5(a) the percentage error in the RBE modules 1, 2 and 3 and SCM methods compared with the GOS method is monitored across a non-fractal to fractal region of parameter space. A 512×512 cell GOS was performed and then a 512×512 cell RBE and SCM over the problem domain for a fixed parameter (A, B) set. Each cell of the RBE and SCM is compared with the equivalent GOS cell and the percentage error, which is basically the percentage deviation from the GOS, was calculated. The fractal boundary caused by a homoclinic tangency was located using [8]. In the non-fractal boundary region both SCM and RBE produce quite good results. RBE modules 1, 2 and 3 are in fact exactly the same as the GOS method here. In the fractal boundary region the percentage error in the SCM becomes much greater, increasing ten-fold, to about 3% across the problem domain. For RBE, in the fractal boundary region, the percentage error also increases but still remains quite small at less the 0·5% across the problem domain. Figure 5(b) indicates the variation in error of the RBE modules versus the GOS drawn at a different scale. The errors in the RBE are mainly in the fractal regime. Module 1 is the least accurate and module 3 the most accurate. Module 3 here produces an error of less than 0·05% across the parameter range. Figure 6 displays an instance in the fractal regime of: (a) the catchment basin; (b) the errors in the SCM method; (c) the errors in RBE module 1; and (d) the errors in RBE with module 3. Note that while the errors in the SCM represent only a 3·06% deviation from the GOS across the problem domain graphically it appears that there is quite a noticeable error. The errors for SCM seem to be predominately located around the tips of large tongues which intrude into the catchment basin and at the boundaries of the catchment basin. The generally accepted result [9] that SCM is in fact not very good at describing fractal boundaries accurately is confirmed. While the errors in RBE are located in very fine whiskers which are generally on the extreme edges of the catchment basin, module 3 is almost ten times more accurate than module 1 of the RBE methods.

6.2. WHAT IS THE EFFECT OF FRACTAL AND NON-FRACTAL BOUNDARIES ON COMPUTATIONAL SPEED?

Consider Figure 7 which results from the same analysis as Figure 5. The number evaluations of equation (3) for RBE modules 1, 2 and 3 and SCM are compared with GOS across a problem domain. The time taken to complete a portrait of a catchment basin is proportional to the number of iterates of equation (3). For differential systems, which require numerical integration, each iterate is expensive and this is major factor in the overall computational time taken. For the Henon map each iterate is cheap computationally so other methodological overheads associated with RBE and SCM are important for equation (3). However the main aim of this paper is directed towards differential systems where these overheads have less effect on the overall time taken, thus the number of iterates is important. The percentage of function evaluations is a comparison of the number of iterates of equation (3) for RBE and SCM against GOS. In the non-fractal region SCM requires less than 15% of the computational effort of the GOS, while RBE modules 1, 2 and 3 only require about 5%. In the fractal boundary region the two methods become more comparable and SCM moves up to 22% and RBE module 1

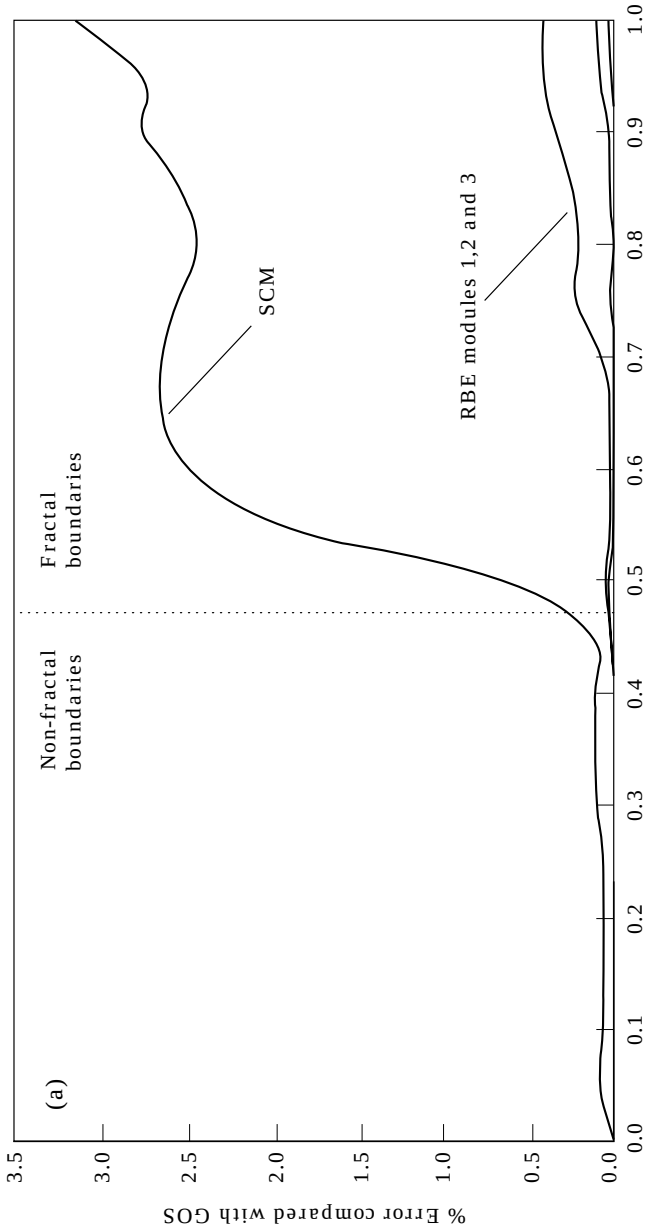

Figure 5(a)—(*Caption on facing page*)

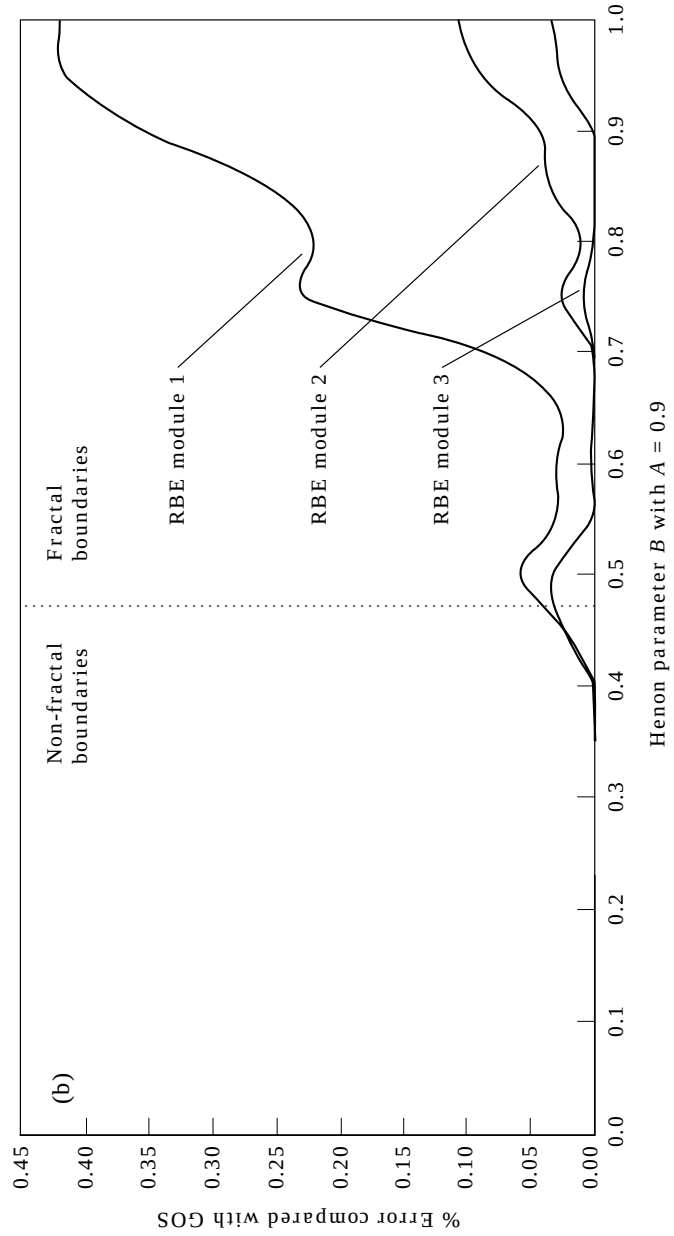

Figure 5. (a) Comparison of errors of SCM and RBE versus GOS at 512×512 resolution. (b) Comparison of errors of RBE with modules 1, 2 and 3 versus GOS at 512×512 resolution.

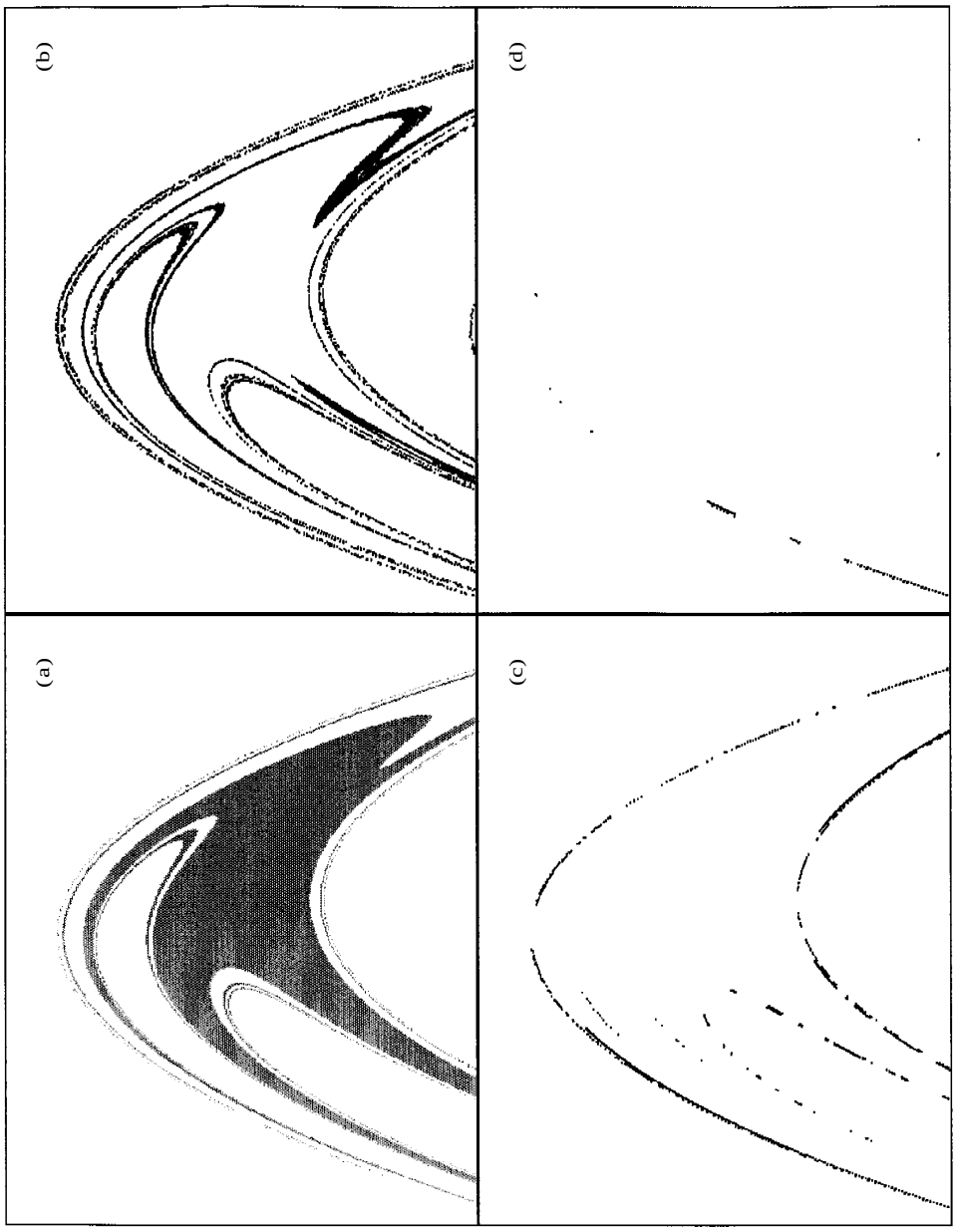

Figure 6. Graphical comparison of errors of SCM and RBE versus GOS. Henon parameter $A = 0{\cdot}9$, $B = 0{\cdot}95$ (512×512 resolution). (a) RBE module 3 (almost exactly the GOS result). (b) Error in SCM (this represents a 3·06% deviation from the GOS result). (c) Error in RBE module 1 (this represents a 0·45% deviation from the GOS result). (d) Error in RBE module 3 (this represents a 0·045% deviation from the GOS result).

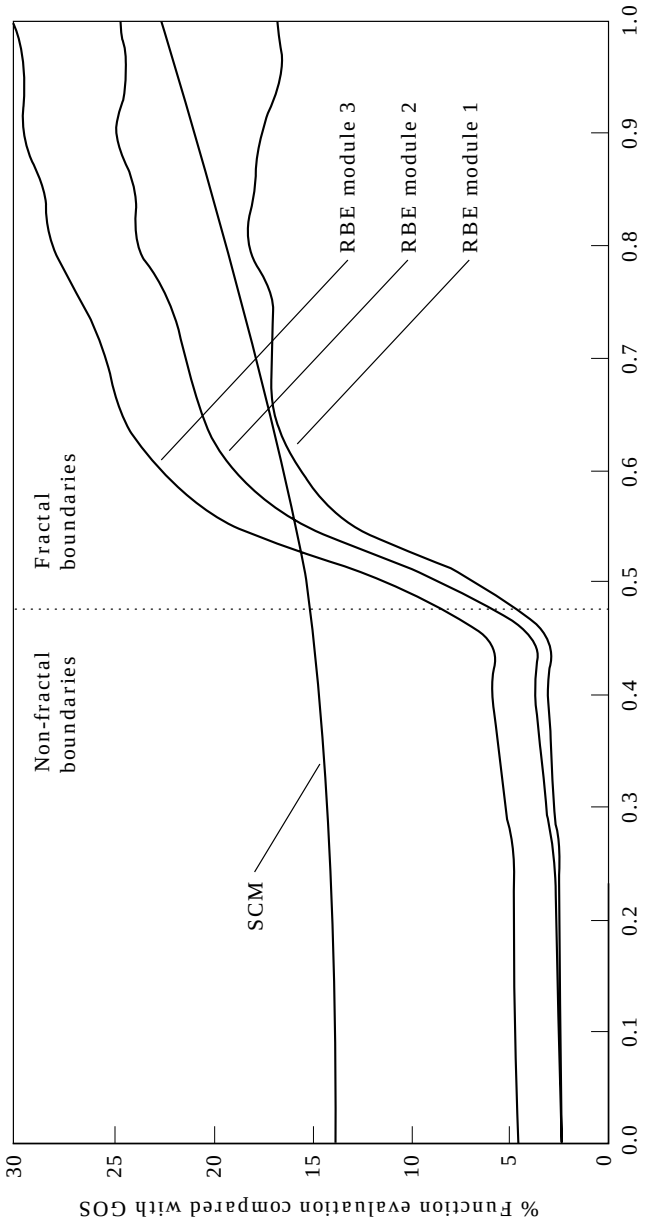

Figure 7. Comparison of computational effort of SCM and RBE versus GOS at 512×512 resolution.

about 16% of GOS. RBE modules 2 and 3 are computationally more expensive than SCM in the fractal regime at this resolution. Clearly both SCM and RBE represent a considerable computational saving over GOS. Note that RBE module 3 represents only 30% of the computational effort of GOS with at worst a 0·05% deviation in the present studies.

6.3. WHAT IS THE EFFECT OF CELL RESOLUTION ON ACCURACY?

Consider Figure 8(a): this diagram compares the percentage error (averaged across the parameter range tested in Figures 5 and 7) in evaluations for RBE and SCM with increasing number of cells. While the averages cannot be defined very precisely and are dependent on the sampling chosen, they do indicate the nature of general trends. The first point on all graphs is at 64×64 cell resolution across the Poincaré section P. In this figure both SCM and RBE errors reduce as the number of cells increase. Note the RBE modules 1, 2 and 3 are never that inaccurate always being less than 1% error from GOS regardless of cell size and boundary type. Figure 8(b) is drawn at a different scale indicating the comparative accuracy of RBE modules 1, 2 and 3.

6.4. WHAT IS THE EFFECT OF CELL RESOLUTION ON COMPUTATIONAL EFFICIENCY?

In Figure 9 the percentage of function evaluations for SCM remain constant at about 20% of GOS. Increasing the number of cells has no effect on computational saving over GOS. For RBE modules 1, 2 and 3, increasing the number of cells reduces the relative effort, computationally. The saving over GOS increases as the number of cells is increased. Under these fractal boundaries SCM is computationally less expensive than RBE module 1 until a 512×512 cell resolution. At a 1024×1024 cell resolution, RBE modules 1 and 2 are computationally less expensive than SCM. What is clear from this figure is that the use of RBE is only worth considering when quite high resolution portraits are needed when the computational saving over GOS is apparent.

6.5. NUMERICAL STUDIES IN DIFFERENTIAL SYSTEMS

Figure 9 is from Thompson's escape equation [1, 5, 8] (a harmonically forced system with non-linear softening stiffness)

$$\ddot{x} + \beta\dot{x} + x - x^2 = F \sin(\omega t + \phi). \tag{4}$$

Figure 10 shows the gradual refinement process of RBE module 1 for the escape equation parameters ($\beta = 0{\cdot}1$, $\omega = 0{\cdot}85$, $F = 0{\cdot}075$, $\phi = 0{\cdot}0$) with a Poincaré section displayed from ($x = -0{\cdot}9$, $\dot{x} = -0{\cdot}9$) to ($x = 1{\cdot}2$, $\dot{x} = 0{\cdot}9$). A classical Runge–Kutta algorithm is used to solve equation (4). The dots displayed in this diagram represent the position of a good cell; i.e., there has been an accurate evaluation of equation (2b) at every dot. White space indicates area where equation (2b) has been approximated. As should be expected all the dots are congregated around the fractal basin boundaries of this problem.

7. ALGORITHMIC COMPARISONS

One way of considering the SCM method is in terms of the way in which it approximates the Poincaré map (2). SCM basically divides the P into a grid of cells where each cell is given a constant value of the Poincaré map (2) within that cell. This approximate constant cell value becomes more accurate as an average across each cell as the cell size decreases. However as the cell size decreases the computational effort increases. It is clear that in some regions of the map (2) which are relatively flat, $\partial \underline{f}(\underline{x})/\partial \underline{x}$ is small and the accuracy of the SCM will be good. While in other regions of P where $\partial \underline{f}(\underline{x})/\partial \underline{x}$ is large (e.g., boundary

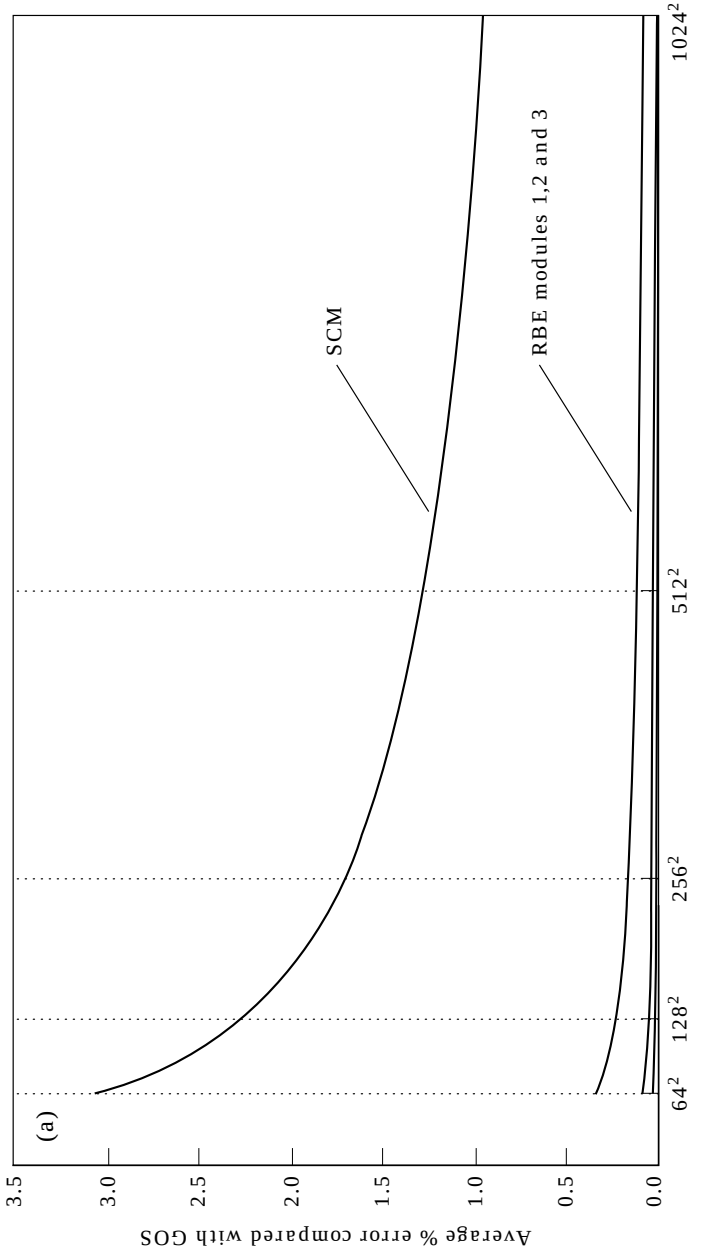

Figure 8(a)—(*Caption on following page*)

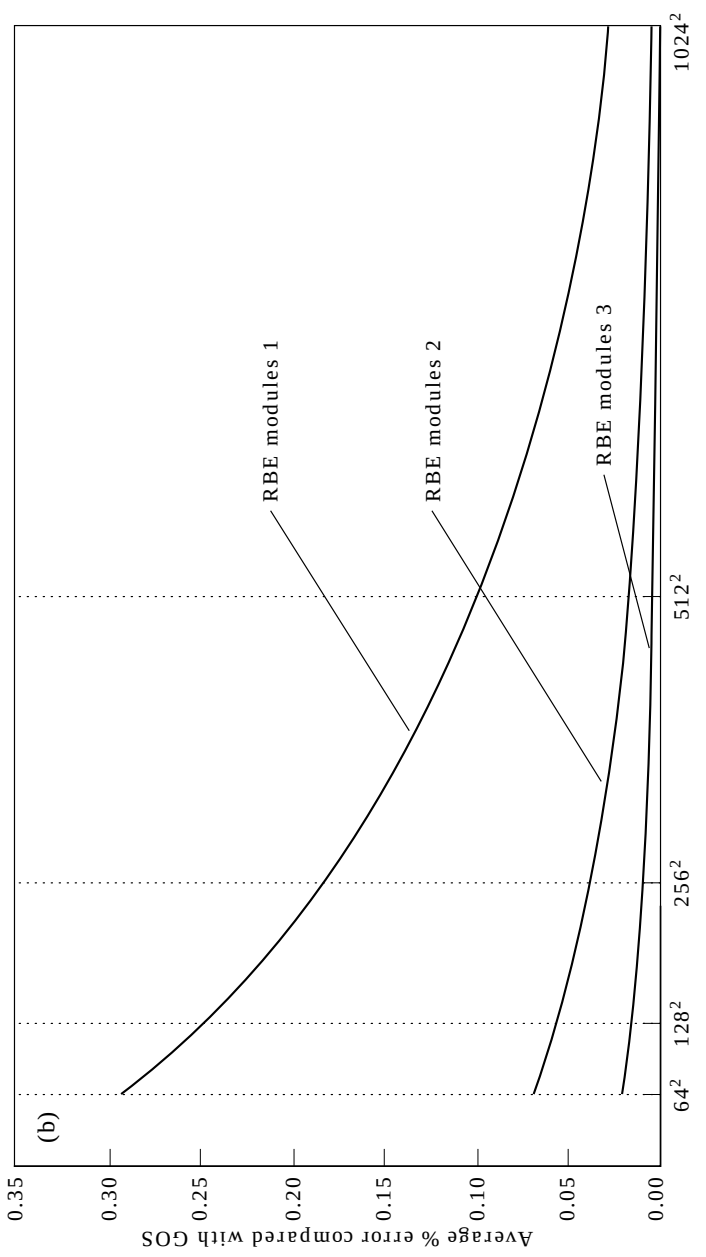

Figure 8. (a) Effects of cell size on average errors (average including fractal and non-fractal boundaries). (b) Effects of cell size on average errors (average including fractal and non-fractal boundaries).

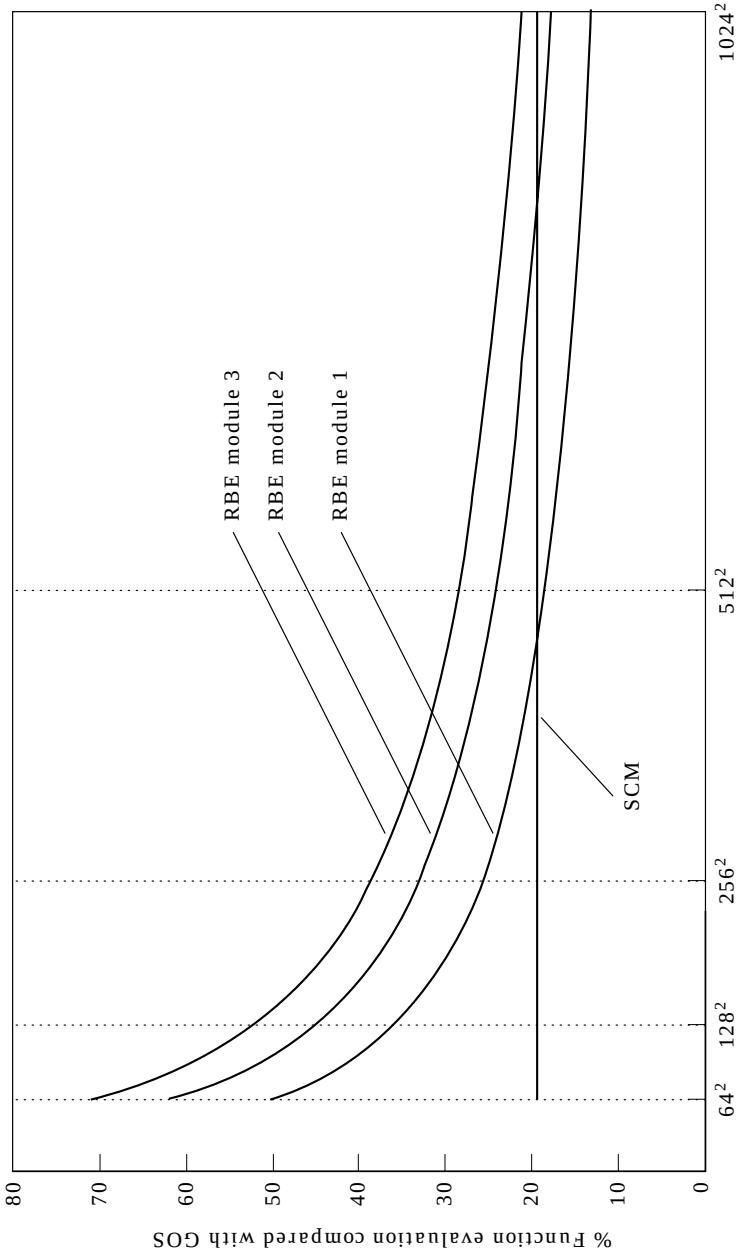

Figure 9. Effects of cell size on computational speed. Henon parameter $A = 0{\cdot}9$, $B = 0{\cdot}8$.

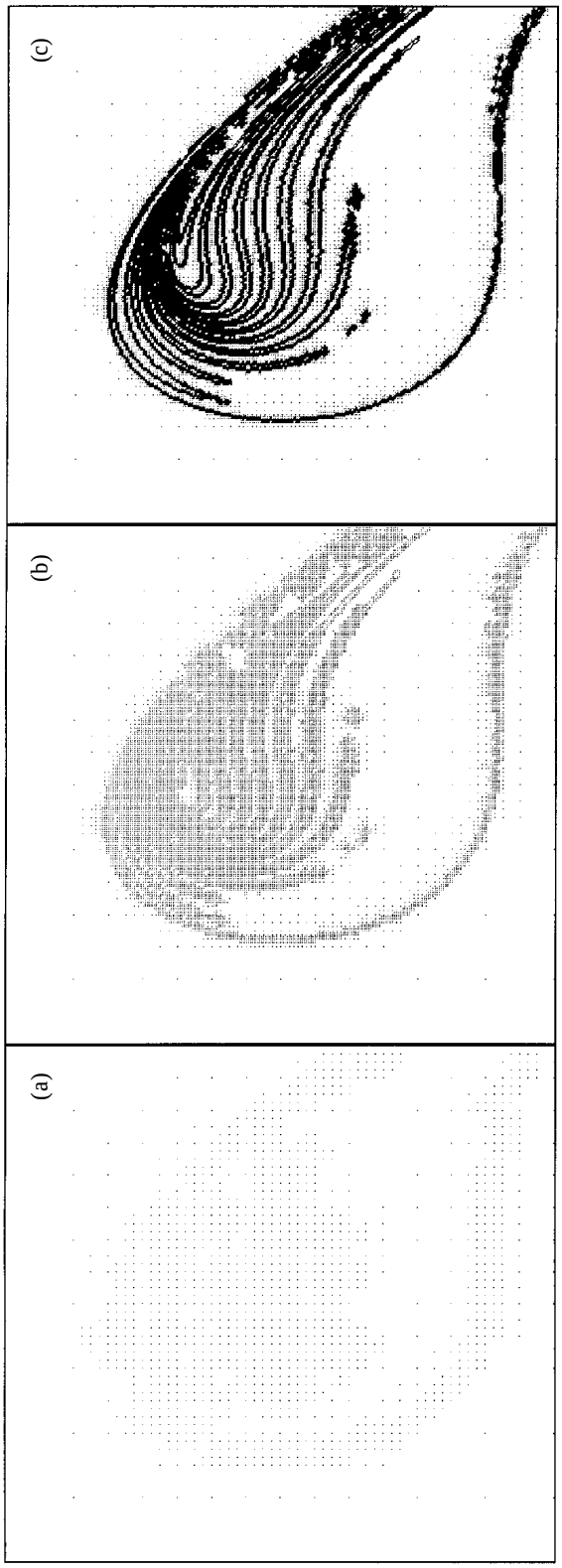

Figure 10. Distribution of "good cells" across Poincaré section: (a) 64×64 cells; (b) 256×256 cells; (c) 512×512 cells.

regions) the SCM will not be so accurate. Thus SCM is highly sensitive to the cell size at the boundaries of catchment basins. The results of this algorithmic problem are that the described catchment basin boundary at one cell size can be substantially different to that at a smaller cell size.

The ICM method [10] is a natural extension from Hsu's SCM in that it seeks to perform some linear interpolation from the centre to centre of cells so that in effect the Poincaré map (2) is approximated by a piecewise bi-linear function with slope discontinuities at the centre of the cells. (SCM is in a sense a piecewise constant function with step discontinuities at the cell edges.) One could in fact go the whole way and introduce a bi-cubic spline interpolation across the whole Poincaré section, thus including function and slope continuity. However the main problem of large slope and curvature of Poincaré map (2) is ignored in all these methods because the regions of high slope of (2) are initially unknown. If it were possible to have more small cells in regions of high slope and curvature of (2), a consistent accuracy across a Poincaré section could be achieved. While SCM, ICM and any higher order interpolation method represent considerable computational saving over the GOS, one would have to either compare the results with the GOS method to verify accuracy which of course defeats the whole purpose of using such an approximate method, or introduce some cell reduction procedure for self-validation. This repetitive cell reduction and comparison is not a standard part of these methods. Note also Figure 6(b) which shows that for SCM to prove accurate at the boundaries, a very large amount of small cells would have to be used.

One of the main advantages of the GOS method from a researcher's point of view, apart from its extreme simplicity, is that the investigation of a sub-region Q of the Poincaré section P can be achieved with practically no special configuration of the algorithm. The problem with SCM, ICM etc. is that cells outside the region of interest may be required by a solution trajectory i.e., one which may pass outside the sub-region Q and then back inside Q in subsequent iterates of (2). Thus the smallest region that can be algorithmically selected is that region which contains all the catchment basins of all the main attracting sets, computationally almost the entire main area of interest of the Poincaré section. There is considerable computational saving over the GOS in the case of where the region of particular interest Q is almost the complete main area of interest of P. However when the sub-region Q is for example only a small part of a catchment basin in P the region algorithmically required (for the grid of cells) is often still almost the entire main area of interest of the Poincaré section. Hence to achieve a similar picture resolution over a small sub-region Q may require a vast number of cells outside this region of interest to be calculated. The computational saving of SCM, ICM etc. becomes much more marginal over the GOS in this case.

RBE is an extension and modification of the GOS method. It does not attempt to approximate (2) but attempts to approximate the repeated iterate of (2) i.e., equation (2b). RBE attempts to respond to the major difficulties in the cell mapping methods described above.

First, since RBE is based on repetitive cell division a process of self-validation is already built into the algorithm. It can keep a record of the percentage change in the areas of catchment basins with each cell division.

Second, the good cells (one for which equation (2b) has been exactly evaluated) are all located at the boundaries. In effect equation (2b) is evaluated predominantly at the location where equation (2) is varying most rapidly and unpredictably. Bad cells tend to be located away from the boundaries where (2) is relatively flat. Thus approximations are made in the regions of P where the errors induced are likely to be small.

Finally, any sub-region Q of the Poincaré section P can be investigated in a similar fashion to the GOS method. This is because each evaluation of equation (2b) is in essence the same as the GOS method. Cell mapping techniques cannot easily work with sub-regions of P.

8. CONCLUSIONS

The improved RBE algorithm is a computational efficient algorithm for the description of basins of attraction in non-linear dynamical systems. In the present studies the use of a comparative module for defining "boundary cells" is discussed. The general conclusions are that RBE module 3 produces results that are almost identical to the conventional GOS method but at a great computational saving for differential systems. It is also clear that this computational saving is increased as the cell resolution is increased. Hence if very high resolution portraits of basins of attraction are required, RBE is the algorithm to use. RBE modules 1 and 2 provide a method increasing computational speed at the expense of loss of accuracy. Note, however, that RBE with any comparative module is never that inaccurate. Studies in section 6 have indicated that RBE modules 1 and 2, in fractal and non-fractal regimes, regardless of cell resolution, have a maximum deviation from the GOS result of less that 0·3%. Also RBE modules 1 and 2 are generally exact under a non-fractal regime. The only drawback of RBE is the complexity of the algorithm which is a problem to implement. The Fortran 90 code and flowcharts are presented in the Appendix. It is set up from Thompson's escape equation. The compiled code for a PC can be found by emailing N.A.Alexander@uel.ac.uk

While the GOS method is always going to be the researcher's preferred solution when computational speed is not paramount, RBE does provide an accurate and fast alternative. In the comparison between RBE and the cell mapping method SCM and ICM, RBE provides a much more accurate solution at equivalent computational efficiency. When global stability events are investigated by monitoring the evolution of basins of attraction under parameter change, RBE may prove a very useful tool.

ACKNOWLEDGMENTS

The author would like to thank Professor J. M. T. Thompson of University College London for his guidance and advice. He would also like to thank the University of East London for their support.

REFERENCES

1. J. M. T. Thompson and H. B. Stewart 1986 *Nonlinear Dynamics and Chaos*. Chichester: John Wiley.
2. S. W. McDonald, C. Grebogi, E. Ott and J. A. Yorke 1985 *Physica* **17D**, 125–153. Fractal basin boundaries.
3. Y. Ueda 1980 in: *New Approaches to Nonlinear Problems in Dynamics* (P. J. Holmes, editor). Philadelphia, SIAM 311.
4. N. A. Alexander 1989 *Journal of Sound and Vibration* **135**, 63–77. Production of computational portraits of bounded invariant manifolds.
5. M. S. Soliman and J. M. T. Thompson 1991 *Applied Ocean Research* **13**, 82–92. Transient and steady state analysis of capsize phenomena.
6. N. A. Alexander 1995 *Proceedings of the 6th International Conference on Civil and Structural Engineering Computing*, 29–35. Volume Developments in Computational Techniques for Structural Engineering. Edinburgh, UK: Civil-Comp Press. Recursive boundary enhancement

(RBE) an algorithm for computational portraits of basin of attraction in non-linear dynamic systems.
7. T. S. Parker and L. O. Chua 1989 *Practical Numerical Algorithms for Chaotic Systems*. New York: Springer-Verlag.
8. N. A. Alexander 1989 *PhD Thesis*, *University College*, *London*. Computational algorithms for the global stability analysis of driven oscillators.
9. C. S. Hsu 1987 *Cell-to-Cell Mapping*: *A Method of Global Analysis for Nonlinear Systems*. New York: Springer-Verlag.
10. B. H. Tongue 1987 *Physica* **28D**, 401–408. On obtaining global nonlinear system characteristic through ICM.

APPENDIX A: FLOW DIAGRAMS AND CODE DESCRIPTION

The presented RBE code was written in Microsoft Fortran Powerstation and is as near as possible to standard Fortran. Figures A1–A3 represent the flow schematic of the algorithm. The graphics routines have been omitted since these are specific to the MS compiler. This means that the user must supply the subroutine PLOT which graphically displays the array ARRAY. ARRAY spans the region of phase space defined by the rectangle (sXMIN,sYMIN) to (sXMAX,sYMAX). Each element of ARRAY is a cell and for the escape equation an element value 0 indicates a solution trajectory leading to escape while element value 1 indicates a stable solution. To customize the code to other dynamical systems the subroutine EVAL2B must be supplied. The italicized COMMON and type declaration statement will also have to be altered to supply the user defined EVAL2B with the system variables.

APPENDIX B: MS FORTRAN POWERSTATION CODE

```
C ---------------------------------------------------------------
C Recursive Boundary Enhancement
C ver 2.0  f32 Dr. N.A.Alexander 1994
C ---------------------------------------------------------------
      PROGRAM RBE
C recursive data list
      PARAMETER(MAXSTORE = 40000)
      INTEGER*2 RX(MAXSTORE),RY(MAXSTORE)
      INTEGER*4 DATACOUNT
      COMMON /RECLIST/RX,RY,DATACOUNT
C general vars
      INTEGER*2 S,I,J
      INTEGER*4 KK
C arrays  ARRAY .. cell array containing
C                        info on catchment basins
C           FLAGS .. cell array containing
C                        good/bad non/boundary info
      PARAMETER(SS = 512)
      INTEGER*1 ARRAY(SS,SS),FLAGS(SS,SS)
      COMMON /ARRAY/ARRAY
      COMMON /FLAGS/FLAGS
      INTEGER MODE
      COMMON /MODULE/MODE
C define system vars
```

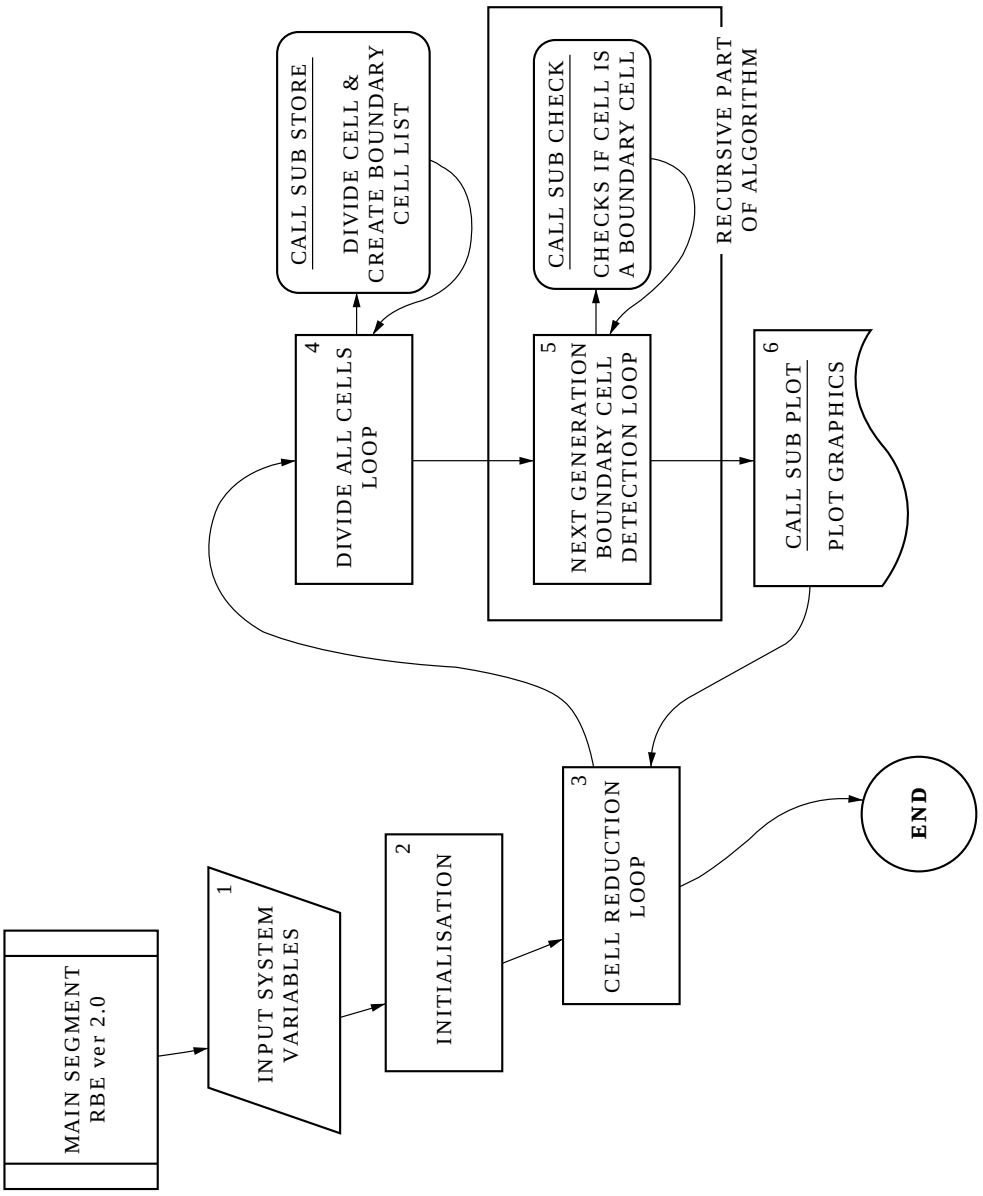

Figure A1. Main segment of algorithm.

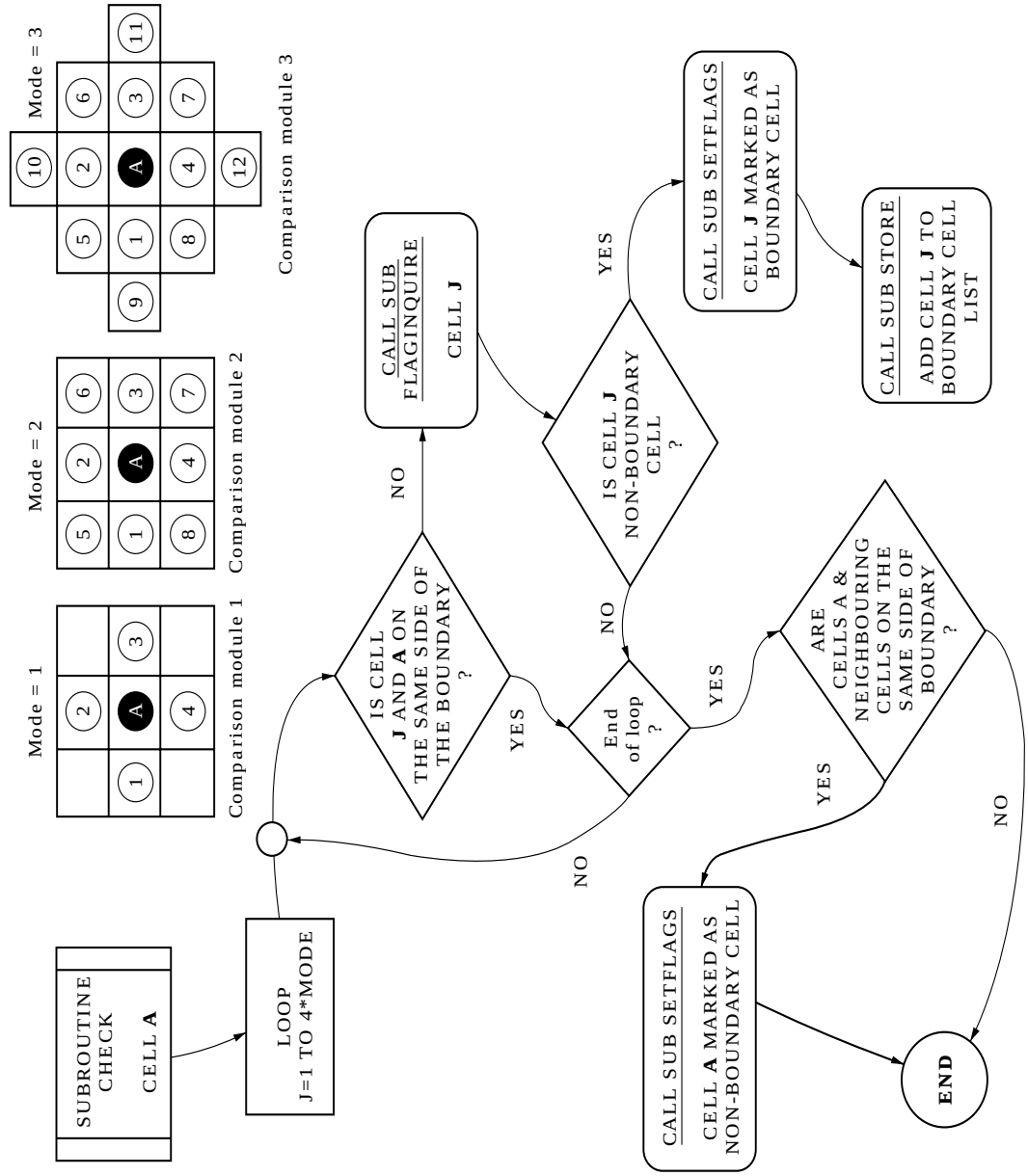

Figure A2. Subroutine CHECK.

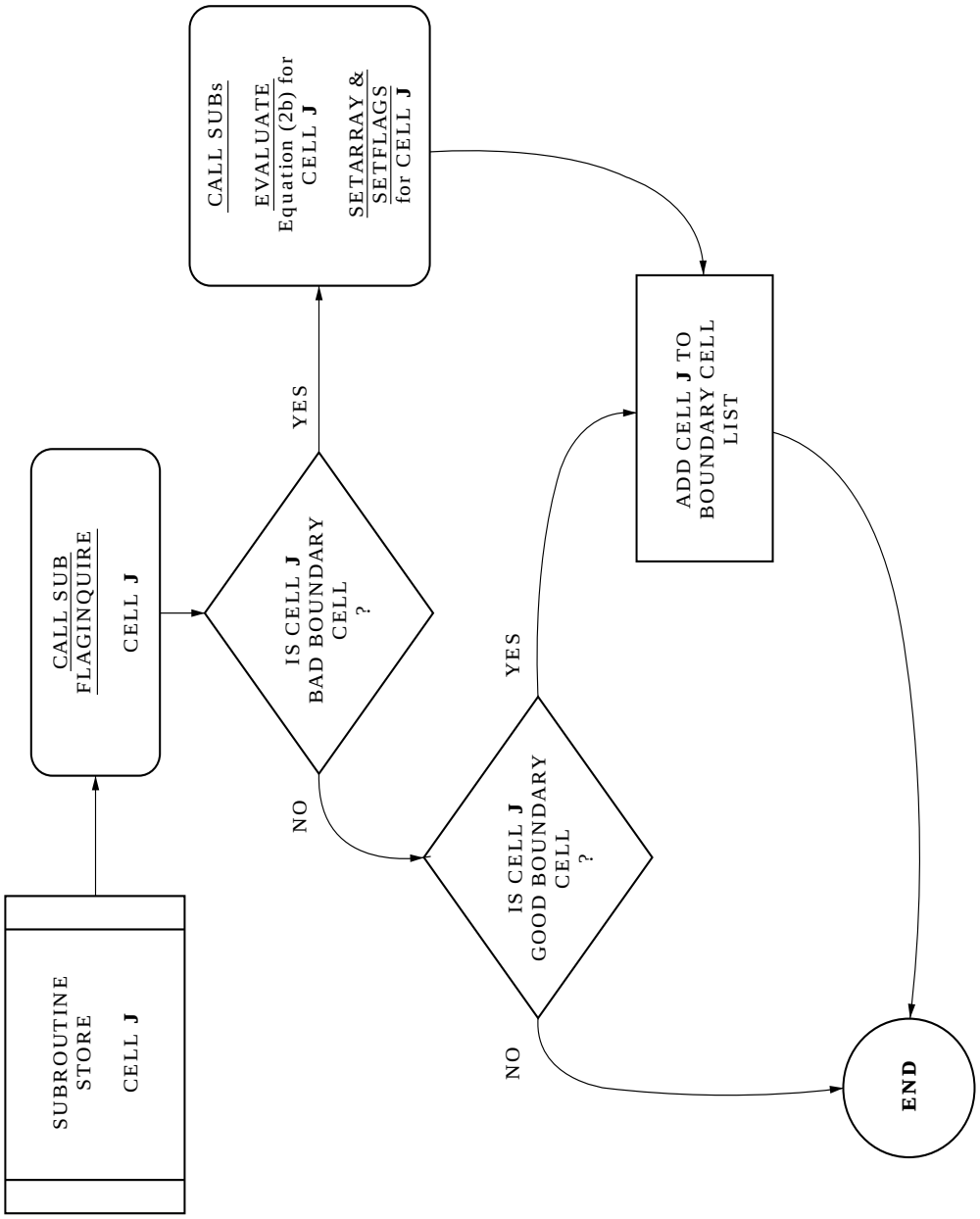

Figure A3. Subroutine STORE.

```
      PARAMETER(PI = 3.141592654D0)
      REAL*8 AA,BB,sXMIN,sYMIN,sXMAX,sYMAX,DT
      COMMON/SYSTEM/AA,BB,sXMIN,sYMIN,sXMAX,
     *                sYMAX,DT
C (1) Input System Variables
C     Variable for escape equation
C     AA = 0.08D0
C     BB = 0.85D0
C     sXMIN = -0.9D0
C     sYMIN = -0.9D0
C     sXMAX = 1.2D0
C     sYMAX = 0.9D0
C     DT = (2.D0*PI/BB)/40.D0
C     MODE = 3          <-- Comparative Module 3
C (2) Initiatise cell size and set all cells
C     to be Boundary cells
C
      S = INT(SS / 8)
      CALL SETARRAY(INT2(1),INT2(1),INT2(SS),
     *                      INT2(1))
      CALL SETFLAGS(INT2(1),INT(1),INT2(SS),
     *                      INT2(1), INT2(0))
C
C (3) Cell Reduction Main Loop
C
      DO WHILE (S.GT.1)
        DATACOUNT = 0
C
C (4) Divide all cells loop
C
      DO I = 1,SS - S + 1,S
        DO J = 1,SS - S + 1,S
          CALL STORE(I, J, S)
        END DO
      END DO
C
C (5) CHECK that cells in recurcive list
C     are actual boundary cells
C
      KK = 0
      DO WHILE (KK.LT.DATACOUNT)
        KK = KK + 1
        CALL CHECK(RX(KK),RY(KK),S)
      END DO
C
C (6) User Define Sub For Graphics Output
C
      CALL PLOT(ARRAY,S,SS)
```

```
C Divide Cell by two and goto (3)
      IF (S.GT.1) THEN
        S = S/2
      ELSE
        S = -1
      ENDIF
     END DO
     END
C -------------------------------------------------------------
      SUBROUTINE CHECK(X,Y,S)
C -------------------------------------------------------------
      INTEGER*2 X,Y,S
      INTEGER*2 ACTUAL,BOUNDARY,J,CHECK1
      INTEGER*2 XP(4),YP(4)
      PARAMETER(SS = 512)
      INTEGER*1 ARRAY(SS,SS)
      COMMON /ARRAY/ARRAY
      INTEGER MODE
      COMMON /MODULE/MODE
      XP(1) = X + S
      YP(1) = Y
      XP(2) = X - S
      YP(2) = Y
      XP(3) = X
      YP(3) = Y + S
      XP(4) = X
      YP(4) = Y - S
      XP(5) = XUP
      YP(5) = YUP
      XP(6) = XUP
      YP(6) = YDOWN
      XP(7) = XDOWN
      YP(7) = YUP
      XP(8) = XDOWN
      YP(8) = YDOWN
C
      XP(9) = XUP + S
      YP(9) = Y
      XP(10) = XDOWN - S
      YP(10) = Y
      XP(11) = X
      YP(11) = YUP + S
      XP(12) = X
      YP(12) = YDOWN - S
C
      CHECK1 = 0
      DO J = 1,4*MODE
        IF((XP(J).LE.SS).AND.(XP(J).GE.1).AND.
     *    (YP(J).LE.SS).AND.(YP(J).GE.1)) THEN
          IF (IABS(ARRAY(X, Y)).NE.
```

```
     *            IABS(ARRAY(XP(J),YP(J)))) THEN
                CHECK1 = CHECK1 + 1
                CALL FLAGINQUIRE(XP(J), YP(J), BOUNDARY,
     *                                          ACTUAL)
                IF((BOUNDARY.EQ.INT2(0)).AND.
     *                          (ACTUAL.EQ.INT2(0)))THEN
C Re-evaluate cell
                  CALL SETFLAGS(XP(J), YP(J), S, INT2(1),
     *                                INT2(0))
                  CALL STORE(XP(J), YP(J), S)
                ELSEIF((BOUNDARY.EQ.INT2(0)).AND.
     *                          (ACTUAL.EQ.INT2(1)))THEN
C flag all squares in cell as boundary cell
                  CALL SETFLAGS(XP(J), YP(J), S, INT2(1),
     *                                INT2(1))
                  CALL STORE(XP(J), YP(J), S)
                ENDIF
              ENDIF
            ENDIF
          END DO
C not a boundary cell
          IF (CHECK1.LT.1) THEN
            CALL FLAGINQUIRE(X, Y, BOUNDARY, ACTUAL)
            CALL SETFLAGS(X, Y, S, INT2(0), ACTUAL)
          ENDIF
          RETURN
          END
C -------------------------------------------------------------
          SUBROUTINE EVALUATE(I,J,S,EVAL)
C -------------------------------------------------------------
          INTEGER*2 I,J,S,II,F,EVAL
          PARAMETER (SS = 512)
          INTEGER*1 ARRAY(SS,SS)
          COMMON /ARRAY/ARRAY
          REAL*8 XI,YI,XS,YS
          REAL*8 AA,BB,sXMIN,sYMIN,sXMAX,sYMAX,DT
          COMMON /SYSTEM/AA,BB,sXMIN,sYMIN,sXMAX,
     *                        sYMAX,DT
C scale variables to window
          XI = (I + 0.5) - 1
          YI = (J + 0.5) - 1
          XS = sXMIN + (sXMAX - sXMIN) * XI / DBLE(SS)
          YS = sYMIN + (sYMAX - sYMIN) * YI / DBLE(SS)
C Evaluate Eqn (2b)
          CALL EVAL2B(XS,YS,F)
          EVAL = F
          RETURN
          END
C -------------------------------------------------------------
          SUBROUTINE FLAGINQUIRE(X,Y,BOUNDARY,ACTUAL)
```

```
C ---------------------------------------------------------------
      INTEGER*2 X,Y,BOUNDARY,ACTUAL
      PARAMETER (SS = 512)
      INTEGER*1 FLAGS(SS,SS)
      COMMON /FLAGS/FLAGS
C
C test first two bits of flag variable
C (bitwise storage reduces memory needed)
C bit 1   0 ... not boundary cell
C bit 1   1 ... boundary cell
C
C bit 2   0 ... not actual computation
C bit 2   1 ... actual computation
C
      BOUNDARY = FLAGS(X, Y).AND.INT1(1)
      ACTUAL = FLAGS(X, Y).AND.INT1(2)
      IF(ACTUAL.GT.0)ACTUAL = 1
      RETURN
      END
C ---------------------------------------------------------------
      SUBROUTINE SETARRAY(X,Y,S,II)
C ---------------------------------------------------------------
      INTEGER*2 X,Y,S,II,I,J
      PARAMETER (SS = 512)
      INTEGER*1 ARRAY(SS,SS)
      COMMON /ARRAY/ARRAY
      DO I = X,X + S - 1
        DO J = Y, Y + S - 1
          ARRAY(I,J) = II
        END DO
      END DO
      RETURN
      END
C ---------------------------------------------------------------
      SUBROUTINE SETFLAGS (X,Y,S,BOUNDARY,ACTUAL)
C ---------------------------------------------------------------
      INTEGER*2 X,Y,S,BOUNDARY,ACTUAL,I,J
      PARAMETER (SS = 512)
      INTEGER*1 FLAGS(SS,SS)
      COMMON /FLAGS/FLAGS
      DO I = X,X + S - 1
        DO J = Y,Y + S - 1
          FLAGS(I, J) = BOUNDARY
        END DO
      END DO
C flag corner only with actual calculation mark.
      FLAGS(X, Y) = BOUNDARY + ACTUAL * INT2(2)
      RETURN
      END
C ---------------------------------------------------------------
```

```
      SUBROUTINE STORE(X,Y,S)
C -------------------------------------------------------------
      INTEGER*2 X,Y,S,BOUNDARY,ACTUAL,EVAL
      PARAMETER(MAXSTORE = 40000)
      INTEGER*2 RX(MAXSTORE),RY(MAXSTORE)
      INTEGER*4 DATACOUNT
      COMMON /RECLIST/RX,RY,DATACOUNT
C
C subroutine to mark cells that have been changed
C to boundary cells and add to recursive list
C
      CALL FLAGINQUIRE(X, Y, BOUNDARY, ACTUAL)
C
      IF ((BOUNDARY.EQ.INT2(1)).AND.
     *          (ACTUAL.EQ.INT2(0)))THEN
        DATACOUNT = DATACOUNT + 1
        IF(DATACOUNT.GT.MAXSTORE)THEN
          STOP'DATASTORE ARRAY LIMIT EXCEEDED'
        ENDIF
        RX(DATACOUNT) = X
        RY(DATACOUNT) = Y
        CALL EVALUATE(X, Y, S, EVAL)
        CALL SETARRAY(X, Y, S, EVAL)
        CALL SETFLAGS(X, Y, S, 1, 1)
      ELSEIF((BOUNDARY.EQ. INT2(1)).AND.
     *                (ACTUAL.EQ.INT2(1)))THEN
        DATACOUNT = DATACOUNT + 1
        IF(DATACOUNT.GT.MAXSTORE)THEN
          STOP'DATASTORE ARRAY LIMIT EXCEEDED'
        ENDIF
        RX(DATACOUNT) = X
        RY(DATACOUNT) = Y
      ENDIF
      RETURN
      END
C -------------------------------------------------------------
C     C SUBROUTINE EVAL2B(X,Y,F)
C -------------------------------------------------------------
C This sub must be supplied by the user given a
C start (x,y) it calculates equation (2b) and F is the
C numbered catchment basin (x,y) belongs to. The
C following is given for the escape equation
      INTEGER I,F,IFAIL
      REAL*8 X,Y
      F = 1
      DO I = 1,10
        CALL RUNGA(X,Y,IFAIL)
        IF(IFAIL.EQ.1)THEN
          F = 0
          EXIT
```

```
        ENDIF
      END DO
      RETURN
      END
C     SUBROUTINE RUNGA(XN,YN,IFAIL)
      REAL*8 XN,YN,TN,X,Y,T,K1,M1,K2,M2,K3,M3,K4,M4
      REAL*8 AA,BB,sXMIN,sYMIN,sXMAX,sYMAX,DT
      INTEGER*2 I,IFAIL
      COMMON /SYSTEM/AA,BB,sXMIN,sYMIN,sXMAX,
     *                sYMAX,DT
      IFAIL = 0
      TN = 0.D0
      DO I = 1,40
        X = XN
        Y = YN
        T = TN
        CALL EQUAT (X, Y, T, K1, M1)
        X = XN + M1 * 0.5D0
        Y = YN + K1 * 0.5D0
        T = TN + DT * 0.5D0
        CALL EQUAT (X, Y, T, K2, M2)
        X = XN + M2 * 0.5D0
        Y = YN + K2 * 0.5D0
        CALL EQUAT (X, Y, T, K3, M3)
        X = XN + M3
        Y = YN + K3
        T = TN + DT
        CALL EQUAT (X, Y, T, K4, M4)
        YN = YN + (K1 + 2.D0*K2 + 2.D0*K3 + K4)/6.D0
        XN = XN + (M1 + 2.D0*M2 + 2.D0*M3 + M4)/6.D0
        TN = TN + DT
        IF (DABS(XN) + DABS(YN).GT.10.D0)THEN
          IFAIL = 1
          EXIT
        ENDIF
      END DO
      RETURN
      END
C     SUBROUTINE EQUAT (X, Y, T, K, M)
      REAL*8 X,Y,T,K,M
      REAL*8 AA,BB,sXMIN,sYMIN,sXMAX,sYMAX,DT
      COMMON /SYSTEM/AA,BB,sXMIN,sYMIN,sXMAX,
     *                sYMAX,DT
      K = DT*(AA*DSIN(BB*T) - 0.1D0*Y - X + X*X)
      M = DT*Y
      RETURN
      END
```