

Estimation of discrete-event systems using interval computation

L. Jaulin, J.L. Boimond and L. Hardouin

Laboratoire d'Ingénierie des Systèmes Automatisés

62, avenue Notre-Dame du Lac 49000 ANGERS

Tel: (33) 2 41 73 04 17

[jaulin,boimond,hardouin]@univ-angers.fr

Abstract: Discrete-event systems are driven by events and generate events. To describe their evolution, the dater approach associate to each event a sequence of dates, namely a *dater*, corresponding to the dates at which the event occurs.

In this paper, we show that for a large class of discrete-event systems, the dater approach makes it possible to cast the characterization of the set of all parameters that are consistent with some collected dater, in a bounded-error context, into a set-inversion framework. Set inversion consists of characterizing the reciprocal image of a given set by a known function. Provided that an inclusion function is known for the function to be inverted, the characterization can be performed by the interval-based algorithm SIVIA. A short presentation of this algorithm is recalled in this paper. The approach is illustrated through three examples.

Keywords: Bounded-error estimation, discrete-event systems, interval analysis, set inversion.

I. INTRODUCTION

Many systems such as manufacturing, communication or transportation systems, are *event-driven* as opposed to *time-driven*, *i.e.*, their behaviors are governed only by occurrences of different types of events rather than by ticks of a clock. For example, events can be: 'the machine has just completed its task' or 'the traffic light changes to red'. Such systems are known as Discrete-Event Systems (DES), (see, *e.g.*, [5], [12], and references therein) and are often described graphically by using *Petri nets* (see, *e.g.*, [11]). When an event occurs, a process may start. When the process is finished, the event 'process ends' yields. When the processing time is known for each process of the DES, the DES is a *Timed Discrete Event System* (TDES) (see, *e.g.*, [3], [5] and [7]). Only TDES will be considered in this paper.

Among the approaches proposed in the literature to describe TDES by recurrent equations, let us quote :

- *the counter approach*: To each event of the TDES, is associated the integer $k(t)$, which represents the number of occurrence of the event at time t . The function $k(t)$ is non decreasing and is called a *counter*.
- *the dater approach*: To an event, is associated the real $x(n)$ which is the date at which the event occurs for the n th time. The sequence $x(n)$ is non decreasing and is called a *dater*.

Example 1: A machine assemble two parts A and B in δ seconds. The three events involved in our example are: e_A = 'a part A arrives', e_B = 'a part B arrives' and e_Y = 'an assembling process ends'. Associated counters are denoted $k_A(t)$, $k_B(t)$, $k_Y(t)$ and associated

daters $x_A(n), x_B(n), x_Y(n)$, respectively. The counter and the dater equations are:

$$\begin{aligned} k_Y(t) &= \min(k_A(t - \delta), k_B(t - \delta), k_Y(t - \delta) + 1) \\ x_Y(n) &= \max(x_A(n), x_B(n), x_Y(n - 1)) + \delta \end{aligned}$$

◇

In what follows, we shall only consider dater equations, but the approach can also be adapted to counters.

Considering a new definition for the additive and multiplicative operators ($a \oplus b = \max(a, b)$ and $a \otimes b = a + b$), *timed event graphs* (TEG) are TDES which can be described by the following (\oplus, \otimes) -linear state equations (see [1]):

$$\begin{cases} \vec{x}(n+1) &= A \otimes \vec{x}(n) \oplus B \otimes \vec{u}(n+1) \\ \vec{y}(n) &= C \otimes \vec{x}(n) \end{cases} \quad (1)$$

where $\vec{x}(n)$ is a state vector and $\vec{y}(n)$ is the vector of daters of interest. The system considered in Example 1 is a TEG because the dater equation can be written as

$$\begin{cases} x(n+1) &= \delta \otimes \vec{x}(n) \oplus \begin{pmatrix} \delta & \delta \end{pmatrix} \otimes \begin{pmatrix} x_A(n+1) \\ x_B(n+1) \end{pmatrix} \\ x_Y(n) &= -\infty \otimes x(n). \end{cases} \quad (2)$$

TEG are particularly suited to describe synchronization phenomena and because of their (\oplus, \otimes) -linearity, powerful tools are available to deal with them (see, *e.g.*, [2] for parameter estimation, [3] for resource optimization and [4] for control applications), but unfortunately, many TDES cannot be described by (1). Even nonlinear state space equations

$$\begin{cases} \vec{x}(n+1) &= \vec{f}(\vec{x}(n), \vec{u}(n+1)) \\ \vec{y}(n) &= \vec{g}(\vec{x}(n)) \end{cases} \quad (3)$$

cannot be used to describe TDES that are non causal with respect to n . Indeed, for a system described by equation 3, $\vec{y}(n)$ does not depend on $\vec{u}(m)$ if $m > n$ *i.e.* the system (3)

is causal. Unlike discrete time systems that can often be described by (3), many TDES are noncausal because of the special nature of n as illustrated by the following example.

Example 2: A machine assembles two identical parts A in δ seconds to produce a part B . The events involved are $e_u =$ 'a part A arrives' and $e_y =$ 'an assembling process ends'. If $u(n)$ is the input dater associated with e_u and $y(n)$ the output dater associated with e_y , we have

$$y(n) = \max((u(2n), y(n-1)) + \delta. \quad (4)$$

for $n \geq 1$ and $y(0) = -\infty$. Since $y(n)$ depends on $u(2n)$, the system is noncausal with respect to n . Therefore it cannot be described by (3). \diamond

For the sake of simplicity, dater sequences $d(k)$ are assumed to be finite and are represented by a sorted vector \vec{d} . The k th component d_k of \vec{d} corresponds to $d(k)$. Even if state space equations (3) do not exist for a general TDES, it is often easy to express the output dater vector \vec{y} with respect to the input dater vector \vec{u} as

$$\vec{y} = \vec{f}(\vec{u}) \quad (5)$$

where \vec{f} is a discontinuous function that can be represented by an algorithm. Unfortunately, the algorithm \vec{f} may involve sorting procedure, *if* statements, ... and problems like

$$\text{find } \vec{u} \text{ such that } \vec{f}(\vec{u}) = \vec{y} \quad (6)$$

cannot be solved using formal methods. Now, for a large class of functions \vec{f} and for reasonable dimensions of \vec{u} , the resolution of (6) can be performed by using interval analysis (see *e.g.* [6] and [9]).

To illustrate how interval analysis can be an efficient alternative to formal approaches when dealing with TDES, we shall consider, in this paper, the estimation of some unknown parameters of a TDES, in a bounded-error context. If \vec{Y} denotes the known box of all feasible

output data vectors, \vec{u} the known input data vector, \vec{p} the unknown parameter vector and $\vec{f}(\vec{p}, \vec{u})$ the model output data vector, then the problem of interest is to characterize the set

$$\mathcal{P} = \{\vec{p} \mid \vec{f}(\vec{p}, \vec{u}) \in \vec{Y}\}, \quad (7)$$

Since \vec{u} is known, $\vec{f}(\vec{p}, \vec{u})$ will be denoted $\vec{f}(\vec{p})$. The set to be characterized is thus defined by

$$\mathcal{P} = \vec{f}^{-1}(\vec{Y}). \quad (8)$$

The problem to be solved is therefore a set inversion problem. The algorithm SIVIA (Set Inversion Via Interval Analysis, (see [8] and [10])) able to solve it, is presented in Section II. with some basic notions of interval analysis. Illustrative examples are finally treated in Section III.

II. INTERVAL ANALYSIS AND SET INVERSION

A. Notion of interval analysis

Interval analysis [9] is based on the notions of boxes and inclusion functions that are now introduced. A *box* or *vector interval* \vec{X} of \mathbb{R}^n is denoted equally:

$$\vec{X} = ([x_1^-, x_1^+], \dots, [x_n^-, x_n^+]) = (X_1, \dots, X_n) = [\vec{x}^-, \vec{x}^+] \quad (9)$$

where $\vec{x}^- = (x_1^-, \dots, x_n^-)^T$ and $\vec{x}^+ = (x_1^+, \dots, x_n^+)^T$. The set of all boxes of \mathbb{R}^n is denoted by \mathbb{IR}^n . A *principal plane* of a box \vec{X} is a symmetry plane of \vec{X} normal to a side of maximum length. To *bisect* a box \vec{X} means to cut it along one of its principal planes. This operation generates two non-overlapping boxes \vec{X}_1 and \vec{X}_2 such that $\vec{X} = \vec{X}_1 \cup \vec{X}_2$.

Let \vec{f} be a vector function mapping \mathbb{R}^n into \mathbb{R}^m . A set-valued function \vec{F} , defined from \mathbb{IR}^n into \mathbb{IR}^m , is an *inclusion function* of \vec{f} if:

$$\forall \vec{X} \in \mathbb{IR}^n, \quad \vec{f}(\vec{X}) \subset \vec{F}(\vec{X}). \quad (10)$$

Let $w(\vec{X})$ be the width of the box \vec{X} , i.e., the length of its largest side(s). An inclusion function is *convergent* if, for any sequence of boxes $\vec{X}(k)$ of \mathbb{IR}^n ,

$$\lim_{k \rightarrow \infty} w(\vec{X}(k)) = 0 \implies \lim_{k \rightarrow \infty} w(\vec{F}(\vec{X}(k))) = 0. \quad (11)$$

An inclusion function \vec{F}^* for \vec{f} is *minimal* if, for all \vec{X} , $\vec{F}^*(\vec{X})$ is the smallest box that encloses the image set $\vec{f}(\vec{X})$. If \vec{f} is a nondecreasing function, i.e., $\vec{x} \geq \vec{y} \implies \vec{f}(\vec{x}) \geq \vec{f}(\vec{y})$, where inequalities have to be understood componentwise, then a minimal inclusion function for \vec{f} is given by [9]:

$$\vec{F}^*(\vec{X}) = [\vec{f}(\vec{x}^-), \vec{f}(\vec{x}^+)]. \quad (12)$$

Let us denote by \vec{s} the sorting function mapping \mathbb{R}^n into \mathbb{R}^n defined by

$$\vec{s}(x_1, x_2, \dots, x_n) = (x_{i_1}, x_{i_2}, \dots, x_{i_n}), \quad (13)$$

where $\{i_1, \dots, i_n\}$ is a permutation of $\{1, 2, \dots, n\}$ such that $x_{i_a} \leq x_{i_b}$ if $a < b$. For example, $\vec{s}(5, 4, 7) = (4, 5, 7)$. The function \vec{s} is nondecreasing, i.e., $\vec{x} \geq \vec{y} \implies \vec{s}(\vec{x}) \geq \vec{s}(\vec{y})$. The minimal inclusion function for \vec{s} is therefore (see equation (12)) $\vec{S}(\vec{X}) = [\vec{s}(\vec{x}^-), \vec{s}(\vec{x}^+)]$. For example,

$$\begin{aligned} \vec{S}([6, 9], [1, 7], [4, 5]) &= [\vec{s}(6, 1, 4), \vec{s}(9, 7, 5)] = \\ &= [(1, 4, 6), (5, 7, 9)] = ([1, 5], [4, 7], [6, 9]) \end{aligned} \quad (14)$$

B. Set inversion algorithm

The algorithm SIVIA [8],[10] able to characterize the solution set defined by (8) is now presented. Its principle is to partition the prior box of interest \vec{P}_0 into a set of non-overlapping boxes, namely those that are enclosed in \mathcal{P} and those that are outside \mathcal{P} . A convergent inclusion function \vec{F} associated with \vec{f} is assumed to be available. The following tests make

it possible to decide whether a given box \vec{P} is inside or outside the solution set \mathcal{P} :

$$\begin{aligned} \text{(i)} \quad \vec{F}(\vec{P}) \subset \vec{Y} &\quad \Rightarrow \quad \vec{P} \subset \mathcal{P}, \\ \text{(ii)} \quad \vec{F}(\vec{P}) \cap \vec{Y} = \emptyset &\quad \Rightarrow \quad \vec{P} \cap \mathcal{P} = \emptyset. \end{aligned} \tag{15}$$

SIVIA is a recursive routine that makes it possible to draw a bracketing of the solution set \mathcal{P} . Boxes that have been proved to belong to \mathcal{P} via test (i) are painted grey, those that have been proved to be outside from \mathcal{P} via test (ii) are painted white and those that satisfy neither (i) nor (ii) and that are too small ($< \varepsilon$) to be bisected are painted black. The first call of SIVIA is done by 'SIVIA(\vec{P}_0)', where \vec{P}_0 is the prior feasible box for the parameter vector. The box \vec{P}_0 is assumed to be large enough to contain the solution set \mathcal{P} .

SIVIA(\vec{P})

Step 1 If $\vec{F}(\vec{P}) \subset \vec{Y}$, {draw(\vec{P} , 'grey'); return};

Step 2 If $\vec{F}(\vec{P}) \cap \vec{Y} = \emptyset$, {draw(\vec{P} , 'white'); return};

Step 3 If $w(\vec{P}) < \varepsilon$, {draw (\vec{P} , 'black'); return};

Step 4 Bisect \vec{P} getting the two boxes \vec{P}_1 and \vec{P}_2 ;

Step 5 SIVIA(\vec{P}_1); SIVIA(\vec{P}_2);

Remark 1 If we denote by $\Delta\mathcal{P}$ the union of all boxes painted black and by \mathcal{P}^- the union of all boxes painted grey, the solution set \mathcal{P} is bracketed by [8]: $\mathcal{P}^- \subset \mathcal{P} \subset \mathcal{P}^- \cup \Delta\mathcal{P}$. \diamond

Remark 2 For the sake of simplicity, SIVIA has been presented as an algorithm for drawing a bracketing of a two-dimensional solution set, but it can also be used to characterize solution sets of higher dimensions [8]. \diamond

Remark 3 The set inverted by SIVIA is a box \vec{Y} and the set to be inverted in our estimation problem is an interval dater vector \vec{Y}_D . These two sets are not equal: for example, the sequence (or vector) $\vec{y} = \{3; 2\}$, which is not a dater vector (non increasing components) is inside the box $\vec{Y} = [1, 4] \times [2, 5]$, but does not belong to the interval dater vector

$\vec{Y}_D = \{[1, 4]; [2, 5]\}$, because \vec{Y}_D only contains vectors of \vec{Y} with increasing components. Now, for any \vec{p} , the vector $\vec{f}(\vec{p})$ is necessarily a dater vector because \vec{f} simulates a DES. Therefore inverting an interval dater vector or its corresponding box always yields the same set. \diamond

III. APPLICATION TO DES ESTIMATION

In this section, the behavior of SIVIA is illustrated through three test cases.

Test case 1: A machine manufactures parts located in an input stock. The input dater vector

$$\vec{u} = (0, 0, 10, 13, 16, 17, 18, 25, 25, 30)^T \quad (16)$$

contains dates at which a part is given to the input stock. The loading of a part in the machine occurs when the machine is free and when a part to be manufactured is available. The working time of the machine, assumed to be given by $\delta(k) = p_1 + \ln(p_2 k + 1)$, where k is the number of manufactured parts, depends nonlinearly on \vec{p} and k . This may correspond to a situation where the efficiency of the machine decreases as parts are proceeded. The output dater vector \vec{y} contains dates at which the machine has just completed a part. The simulator function $\vec{f}(\vec{p})$ is computed by the following algorithm:

```

Input:  $\vec{p}$ ;
 $\delta(1) := p_1 + \ln(p_2 + 1)$ ;  $f_1 := \delta(1) + u_1$ ;
For  $k := 2$  to 10 do
    {  $\delta(k) := p_1 + \ln(p_2 k + 1)$ ;
       $f_k := \delta(k) + \max(\vec{f}_{k-1}, u_k)$ ; }
Output:  $\vec{f}$ ;

```

The dater vector of experimental data

$$\vec{y} = (1.7, 3.8, 12.4, 15.6, 18.8, 21.7, 24.8, 28.2, 31.5, 34.9)^T \quad (17)$$

has been obtained by considering $\delta(k) := 1 + \ln(k+1)$, which corresponds to set $p_1 = p_2 = 1$. A model output dater vector $\vec{f}(\vec{p})$ is assumed to be acceptable if for all k , $|\vec{f}_k(\vec{p}) - y_k| < 0.3$, *i.e.*, the output dater vector $\vec{f}(\vec{p})$ falls inside the box $\vec{Y} = \vec{y} + 0.3 \cdot [-1, 1]^{10}$, where $[-1, 1]^{10}$ denotes the ten-dimensional box whose components are equal to $[-1, 1]$. For the prior feasible box $\vec{P}_0 = ([-1, 3], [0, 4])$ and for $\varepsilon = 0.01$, SIVIA brackets the posterior feasible set \mathcal{P} for the parameters, as represented by Figure 1, in less than 10 seconds on a PC DX4-100 computer. Grey boxes are proved to be inside \mathcal{P} and white boxes are proved to be outside \mathcal{P} . No conclusions have been reached for the black boxes.

Possible location for Figure 1

Test case 2: Let us consider a plane where ten acoustic transmitters are located at positions $(x_i, y_i)^T$ given by Table 1. Each transmitter emits an indiscernible impulse sound at date 0. A robot, whose position $\vec{p} = (p_1, p_2)^T$ has to be estimated, is equipped with a sound sensor and records a dater vector \vec{y} , the component of which corresponds to dates at which a sound is detected. It is assumed that ten dates have been recorded and stored into the output dater vector

$$\vec{y} = (1, 2.24, 3, 3.6, 4.12, 4.47, 5, 5.1, 5.38, 6.4)^T. \quad (18)$$

The propagation delay associated with the i th transmitter is assumed to be proportional to the distance between the robot and the transmitter:

$$\delta_i = \lambda \sqrt{(x_i - p_1)^2 + (y_i - p_2)^2}, \quad i = 1, \dots, 10. \quad (19)$$

For the sake of simplicity, the constant λ , related to the sound velocity, is assumed to be equal to 1. The set of all feasible dater vector is given by $\vec{Y} = \vec{y} + [-1, 1]^{10}$. The simulator function $\vec{f}(\vec{p})$ is computed by the following algorithm:

Input: \vec{p} ;

For $i := 1$ to 10 do $\delta_i = \sqrt{(x_i - p_1)^2 + (y_i - p_2)^2}$;

Sort the reals δ_i and store them into the dater vector \vec{f} ;

Output: \vec{f} .

For $\vec{P}_0 = ([-1, 10], [-1, 10])$ and $\varepsilon = 0.04$, SIVIA brackets the feasible set \mathcal{S} as represented on Figure 2, in less than 5 seconds on a PC DX4-100 computer.

Possible location for Table 1

Possible location for Figure 2

Test case 3: In a manufacturing system, 10 identical parts are given to the input. The input dater vector is given for example by $\vec{u} = (1, 1, 1, 3, 5, 9, 9, 15, 16, 16)^\top$. These 10 parts have to be manufactured by two machines \mathcal{M}_1 and \mathcal{M}_2 . The loading of a part in a machine occurs only when this machine is free. The working time are p_1 for \mathcal{M}_1 and p_2 for \mathcal{M}_2 . When a part is available, the machine that is chosen to deal with this part is the one that can complete it as soon as possible. The times at which the parts are manufactured are stored in the dater vector \vec{f} .

The following algorithm computes the output dater vector. The real numbers h_1 and h_2 denote times at which \mathcal{M}_1 and \mathcal{M}_2 are available.

Input: p_1 and p_2 ;

- 1 $h_1 := 0; h_2 := 0;$
- 2 For $i := 1$ to 10 do
- 3 $z_1 := \max(h_1, u_i) + p_1; z_2 := \max(h_2, u_i) + p_2;$
- 4 If $z_1 \leq z_2, \{f_i := z_1; h_1 := z_1\}$
- 5 Else $\{f_i := z_2; h_2 := z_2\}$
- 6 EndFor

Output: $\vec{f} = (f_1, \dots, f_{10});$

Line 1: At the beginning, \mathcal{M}_1 and \mathcal{M}_2 are free at time 0. Line 3: If the part i is available at time u_i , if \mathcal{M}_1 is free at time h_1 , and if the working time of \mathcal{M}_1 is p_1 , then \mathcal{M}_1 can complete the part at time z_1 . Idem for \mathcal{M}_2 . Line 4: If \mathcal{M}_1 can complete the part i before \mathcal{M}_2 , \mathcal{M}_1 is selected. The part will be completed at time f_i and \mathcal{M}_1 will then be free at time h_1 .

Unfortunately, to the best of our knowledge, because of the *if* statement, no available methodologies exists to derive an inclusion function for $\vec{f}(\vec{p})$. The algorithm SIVIA is thus unable to deal with this last test case.

IV. CONCLUSION

Problems involved in DES are generally nonlinear, nonconvex and nondifferentiable so that classical methods often fail to give reliable results. As shown in this paper, interval analysis makes it possible to deal with such problems in a global and guaranteed way thanks to the fundamental notion of inclusion function that can be computed for many algorithms involving sorting routines and nonlinear function such as max, min, ln, ...

As an illustration, the problem of characterizing the set of all parameter vectors that are consistent with all collected data, in a bounded-error context, has been considered. This

problem can be cast into a set-inversion framework, where the set of all acceptable output vectors has to be inverted by the simulator function. For this purpose, an interval-based algorithm SIVIA has been presented. SIVIA has treated two illustrative test cases for which, to the best of our knowledge, no other methods existed in the literature that could lead to comparable results. The third test-case was not treated because no inclusion function for the simulator function were available.

The main limitation of our approach is that for a large class of TDES, an inclusion function for the algorithm to be inverted is difficult to obtain and adapted methods should be developed to obtain them.

REFERENCES

- [1] F. Baccelli, G. Cohen, G.J. Olsder and J.P. Quadrat. *Synchronization and Linearity. An Algebra for Discrete Event Systems*, John Wiley & Sons, New York, 1992.
- [2] J.L. Boimond, L. Hardouin and P. Chiron. A modeling method of siso discrete event systems in max-algebra. *ECC'95, Roma*, 1995, 2023–2027.
- [3] J.G. Braker. *Algorithms and applications in Timed Discrete Event Systems*. PhD thesis, Department of Technical Mathematics and Informatics, Delft University of Technology, the Netherlands, 1993.
- [4] C.G. Cassandras, S. Lafortune, and G.J. Olsder. *Introduction to the Modelling, Control and Optimization of Discrete Event Systems*. in Trends in Control: A European Perspective, A. Isidori (ed.), 1995.
- [5] F. Dicesare, G. Harhalakis, J.M. Proth, M. Silva and F.B. Vernadat. *Practice of Petri Nets in Manufacturing*, Chapman & Hall, 1993.
- [6] E. Hansen. *Global Optimization Using Interval Analysis*. Marcel Dekker, New York, 1992.
- [7] H.P. Hillion and J.M. Proth, Performance evaluation of job-shop systems using timed-event graph, *IEEE Transactions on Automatic Control*, 34, 1989,3–9.
- [8] L. Jaulin and E. Walter. Set inversion via interval analysis. *Automatica*, 29(4), 1993, 1053–1064.
- [9] R.E. Moore. Methods and applications of interval analysis. *SIAM, Philadelphia*, 1979.
- [10] R.E. Moore. Parameter set for bounded-error data. *Math. Comput. Simulation*, 34, 1992, 113–119.

- [11] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4), 1989, 541–579.
- [12] M. Silva and R. Valette. Petri Nets and Flexible Manufacturing, *Advances in PN'89*, LNCS 84, Springer-Verlag, 1990.

Figure captions

Figure 1: Paving generated by SIVIA to bracket the solution set.

Figure 2: Paving generated by SIVIA for the characterization of all locations consistent with the bounded-error data.

Table captions

Table 1: Positions of transmitters in the plane.

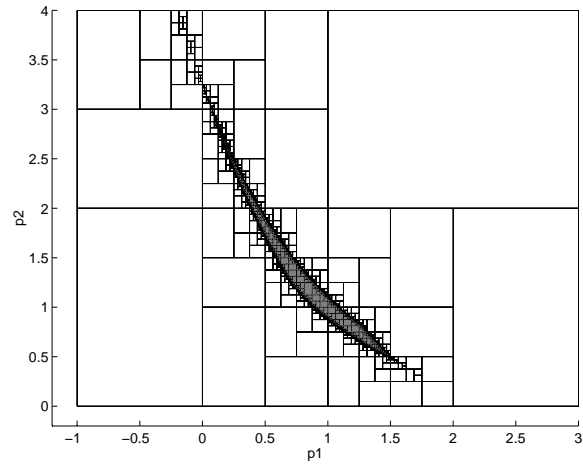


Figure 1

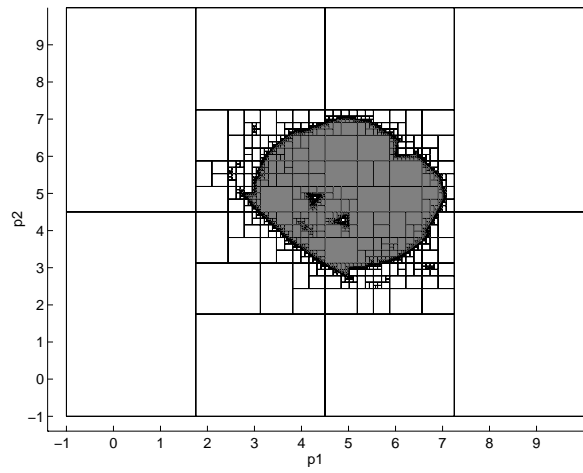


Figure 2

i	1	2	3	4	5	6	7	8	9	10
x	1	1	4	5	7	1	6	8	3	9
y	1	4	1	5	1	7	8	6	3	9

Table 1