Algorithms for Refinement

Zhaoji Wang

2018.2.1

Here I list two refinement methods based on $MKtest$. Using different order of the test to exclude the box and choose correct candidate, The two show in-and-out efficiency with respect to specified input.

1. ϵ: **Size threshold of output box region.**
2. P: **Queue of isolated root boxes.**
3. $OUTPUT$: **Queue of refined root boxes.**

Algorithm 1 pops an isolated box and pushes it to an empty queue Q firstly; Then it splits each box which fails in C_0test until the largest width of box in the Q(suppose all are square boxes) is smaller than ϵ; Finally, it starts doing $MKtest$: Outputting if a box in queue could pass, and then "Break"(because each isolated box contains and only contains one root); continuing splitting if a box in queue Q fail in $MKtest$. This algorithm goes to end if every box in P has been refined, that is to say, we output a fair number of boxes with the input queue.

Algorithm 2 pops an isolated box and pushes it to an empty queue Q as well; Differently, it executes $MKtest$ for boxes in Q for which fails C_0test and discards all the box in Q if one could pass $MKtest$(there is only one root located in boxes of Q); Additionally, outputting and "Break" if the width of this box is smaller than ϵ or pushing subboxes to Q after splitting if its too big. It does splitting as well if the box fails in $MKtest$. This algorithm goes to end if all isolated boxes have been refinded.

I detect that $\underline{MKtest \text{ could not discern the root at the boundary of a box}}$, so aiming to include roots at boundary as well, it's necessary to dilate subboxes with a scalar β when splitting. For both algorithm, we do dilation when it needs to split after failing $MKtest$. Fortunately, we need not change all splitting procedure to "dilated splitting" since $\underline{C_0test \text{ will not exclude box with root at there boundary}}$.

Theoretically, algorithm 2 is more efficient than algorithem 1, because it frequently empty the Queue Q based on the information delivered by other boxes from same ancestor(Isolated box has one and only one root in it.) That is to say, focusing on an isolated box A, other sub-sub$\cdots$sub box could be excluded if a sub-sub$\cdots$sub box of A have passed $MKtest$, so next step is to search root in A. However, test for examples suggest algorithm 1 do better when polynomial system is relatively simple.

For a box B, we denote by $\mathcal{Z}_f(B)$ the set of roots of polynomial system F contained in B.

$C_0(B)$**:** $\exists$ **an integer** $i \in [1, n]$ **such that** $0 \notin \square f_i(B) \quad \Rightarrow \mathcal{Z}_f(B) = 0$**;**

$C_1(B)$**:** $MKtest$ **succeeds over** $B \quad \Rightarrow \mathcal{Z}_f(B) \geq 1$**;**

Algorithm 1: Refinement of isolated root boxes(version **1**)

Input: 1. ϵ 2. P

Output: $OUTPUT$

Using this method to refine the root boxes;

β is a real number in $(1, 2)$, which represent the scalar enlargement to avoid root on the boundary of box being ignored;

while $P \neq \emptyset$ **do**

    $B \leftarrow P.pop()$;

    Initialize queue $Q \leftarrow B$;

    while $Q \neq \emptyset$ **do**

        $B_1 \leftarrow Q.pop()$;

        if $C_0(B_1)$ *fails* **then**

            if B_1.width*()*$>\epsilon$ **then**

                Split B_1 into 2^n congruent subboxes and add them to Q;

            else

                if $C_1(B_1)$*succeeds* **then**

                    $OUTPUT.push(B_1)$;

                    Break;

                else

                    Split B_1 into 2^n congruent subboxes;

                    Dilate all subboxes with β and add them to Q;

Algorithm 2: Refinement of isolated root boxes(version **2**)

Input: 1. ϵ 2. P

Output: *OUTPUT*

Using this method to refine the root boxes;

β is a real number in $(1, 2)$, which represent the scalar enlargement to avoid root on the boundary of box being ignored;

while $P \neq \emptyset$ **do**

    $B \leftarrow P.pop()$;

    Initialize queue $Q \leftarrow B$;

    while $Q \neq \emptyset$ **do**

        $B_1 \leftarrow Q.pop()$;

        if $C_0(B_1)$ *fails* **then**

            if $C_1(B_1)$ *succeeds* **then**

                <u>Discard all boxes in Q</u>;

                if B_1.width*()*$< \epsilon$ **then**

                    OUTPUT.*push*(B_1);

                    Break;

                else

                    Split B_1 into 2^n congruent subboxes, and add them to Q;

            else

                Split B_1 into 2^n congruent subboxes;

                <u>Dilate all subboxes with β</u> and add them to Q;