PoSSo-RealSolving

(Simultaneous Inequalities)

by
Fabrice Rouillier and Guadalupe Trujillo

1 Introduction

2 Simultaneous Inequalities Algorithm

Let J be an ideal, and define Z as the set of real roots of J. Let $Q_1, \ldots, Q_n$ be a list of polynomials. A polynomial Q_i is said to realize a sign condition $\varepsilon \in (+, 0, -)$ at a point if it evaluates to a positive, zero or negative number. Given $Q_1, \ldots, Q_n$ and $\varepsilon = \{\varepsilon_1, \ldots, \varepsilon_n\}$, the realization of the sign pattern ε at the solutions of J is:

$$R_\varepsilon(J; Q_1, \ldots, Q_n) = \{x \in Z \mid (\forall i)\ Q_i(x)\varepsilon_i\}.$$

The aim of the algorithm is to compute the number of elements of Z giving to $Q_1, \ldots, Q_n$ the sign pattern ε:

$$c_\varepsilon(J; Q_1, \ldots, Q_n) = \mathrm{card}(R_\varepsilon(J; Q_1, \ldots, Q_n))$$

Our algorithm is based on the fact that it is possible to compute:

$$V(J; Q) = c_{\{+\}}(J; Q) - c_{\{-\}}(J; Q)$$

Definition 1 *A* pseudo-partition *of Z is a list $C = [C_1, \ldots, C_n]$ of n subsets of Z whose union is Z and whose intersections two by two are empty (the difference with a partition is that some C_i may be empty). A list of polynomials $H = [H_1, \ldots, H_m]$ is* adapted *to a pseudo-partition $C = [C_1, \ldots, C_n]$ of Z if on each subset of C the signs of the polynomials of H are fixed. If H is adapted to C, the* matrix of signs of H on C, *$A(H, C)$ is the $n \times m$ matrix whose (i, j)'s entry is the sign of H_j on C_i.*

For example $[1, Q, Q^2]$ is adapted to $[R_{\{0\}}(J; Q), R_{\{+\}}(J; Q), R_{\{-\}}(J; Q)]$ and the matrix:

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & -1 \\ 0 & 1 & 1 \end{pmatrix}$$

is the matrix of signs of $[1, Q, Q^2]$ on $[R_{\{0\}}(J; Q), R_{\{+\}}(J; Q), R_{\{-\}}(J; Q)]$.

We denote by $c(C)$ the list of cardinals of the C_is, by $\mathrm{sign}(H(x))$ the list of signs of $H_j(x)$ and by $V(J; H)$ the list of integers $V(J; H_j)$, $j \in \{1, \ldots, m\}$.

Proposition 2.1 *Given a pseudo-partition C of Z, and a list of polynomials H adapted to C:*

$$A(C,H)\cdot c(C) = V(J;H)$$

Proof : It is obvious since the j-the row of $A(C,H)$ is the list of signs of H_j on the list C. □

Corollary 2.2 *Given a polynomial Q:*

$$A\cdot\begin{pmatrix} c_{\{0\}}(J;Q)\\ c_{\{+\}}(J;Q)\\ c_{\{-\}}(J;Q)\end{pmatrix} = \begin{pmatrix} V(J;1)\\ V(J;Q)\\ V(J;Q^2)\end{pmatrix}$$

Proof : We have seen above that A is the matrix of signs of $[1,Q,Q^2]$ on $[R_{\{0\}}(J;Q), R_{\{+\}}(J;Q), R_{\{-\}}(J;Q)]$. □

Notation 2.3

- *If $C=[C_1,\cdots,C_n]$ and $C'=[C'_1,\cdots,C'_{n'}]$ are two pseudo-partitions of Z, we denote by $C\cap C'$ the pseudo-partition of Z defined as:*

$$[C_1\cap C'_1,\cdots,C_n\cap C'_1,\cdots,C_1\cap C'_{n'},\cdots,C_n\cap C'_{n'}]$$

- *If $H=[H_1,\cdots,H_m]$ and $H'=[H'_1,\cdots,H'_{m'}]$ are two lists of polynomials, we denote by $[H\cdot H']$ the list of mm' polynomials:*

$$[H_1\cdot H'_1,\cdots,H_n\cdot H'_1,\cdots,H_1\cdot H'_{n'}\cdots H_n\cdot H'_{n'}]$$

- *If A and A' are two $n\times m$ and $n'\times m'$ matrices, we denote by $A\otimes A'$ the $nn'\times mm'$ matrix obtained in replacing the entry $a'_{i,j}$ of A' by the matrix $a'_{i,j}A$.*

Proposition 2.4 *Consider two pseudo-partitions C and C' of Z and two lists of polynomials H and H' such that H (resp. H') is adapted to C (resp. C'). Then:*

$$A(H,C)\otimes A(C',H') = A(C\cap C', H\cdot H')$$

Proof : This follows immediately from the definitions. □

Corollary 2.5 *Consider two pseudo-partitions C and C' of Z , and two lists of polynomials H and H' such that H (resp. H') is adapted to C (resp. C'). Then:*

$$A(H,C)\otimes A(C',H')\cdot c(C\cap C') = V(J;H\cdot H')$$

2.0.1 Simultaneous inequalities

The aim of the algorithm we shall describe is to determine the number of roots of J giving fixed signs to $Q_1, \ldots, Q_n$.

Let us consider the case $n = 1$. Let $C^{(Q_1)}$ be the pseudo-partition of Z $[R_{\{0\}}(J; Q_1), R_{\{+\}}(J; Q_1), R_{\{-\}}(J; Q_1)]$, and $H^{(Q_1)} = [1, Q_1, Q_1^2]$. Since we know a function V such that:

$$V(J; Q_1) = \#\{x \in Z \mid Q_1(x) > 0\} - \#\{x \in Z \mid Q_1(x) < 0\}$$

we can compute $c_{\{0\}}(J; Q_1)$, $c_{\{+\}}(J; Q_1)$, $c_{\{-\}}(J; Q_1)$ from these values and the linear relation:

$$A \cdot c(C^{(Q_1)}) = V(J; H^{(Q_1)})$$

Combining the matrix equalities corresponding to Q_1 and Q_2 we have:

$$A \otimes A \cdot c(C^{(Q_1)} \cap C^{(Q_2)}) = V(J; H^{(Q_1)} \cdot H^{(Q_2)})$$

More generally, for $l = 1, \ldots, n$, we have the following equality:

$$A_l \cdot c(C^{(Q^{(l)})}) = V(J; H_l)$$

where A_l, H_l, $Q^{(l)}$, $C^{(Q^{(l)})}$ are recursively defined as follow:

- $A_l = A_{l-1} \otimes A = \begin{pmatrix} A_{l-1} & A_{l-1} & A_{l-1} \\ 0 & A_{l-1} & -A_{l-1} \\ 0 & A_{l-1} & A_{l-1} \end{pmatrix}$
- $H_l = H_{l-1} \cdot H^{(Q_l)}$
- $Q^{(l)} = [Q_1, \ldots, Q_l]$
- $C^{(Q^{(l)})} = C^{(Q^{(l-1)})} \cap C^{(Q_l)}$

Using the precedent results, we see that we can obtain all the sign conditions realized by the polynomials $Q_1, \ldots, Q_n$ solving linear systems recursively defined:

$$A_n \cdot c(C^{(Q^{(n)})}) = V(J; H_n)$$

(Note that the matrices we obtain is each step are nonsingular, since they are the Kronecker product of two nonsingular matrices).

If we continue this process iteratively, then, without some strategy, the dimensions of the matrices involved in our calculus will grow in an exponential way. We will need to solve a linear system of 3^n equations in 3^n variables, and 3^n calls to the V function will be necessary. To avoid this exponential behavior, the algorithm applies the strategy of Ben-Or, Kozen and Reif ([BKR86]). The idea is to remark that, if the ideal J has k real roots, then the number of different sign conditions realised by $Q_1, \ldots, Q_n$ at the real solutions of J, is bounded by k. Using this idea, at any step of the algorithm, we reduce the system by dropping the information corresponding to the sign conditions that are not realized by a solution of the ideal. Next section presents the algorithm in more detail.

3 Implementation

Input:

- An ideal J, given by an Univariate Polynomial (in the univariate case), or by the Multiplication table corresponding to the problem (in the multivariate case).
- A list of polynomials constraints, $Q_1, \ldots, Q_n$.

Output:

- An element of the class $SIout$, which consists of:
  - $SIMatrix$: A non singular matrix.
  - $lsig$: The list of sign conditions which are realised at the roots of the ideal.
  - $lsol$: A vector containing the number of roots of J that verify each sign condition set of $lsig$.
  - $lpol$: The list of polynomials, products of powers of the initial set $Q_1, \ldots, Q_n$, which are present in the reduced system.

- $lRSV$: The vector $V(J; lpol)$.
- $SIdeal$: The ideal J, represented as in the input.
- Dim: The dimension of $SIMatrix$, which coincides with the number of distinct sign conditions.
- $Nroots$: The number of real roots of the ideal J.

At each step of the algorithm, the following relation holds:

$$SIMatrix \cdot lsol = lRSV$$

The algorithm begins with an initialisation step followed by an iterative process which computes the sign conditions over the set of polynomials $Q_1, \ldots, Q_{l+1}$ from the output corresponding to $Q_1, \ldots, Q_l$.

Initialisation Step

The program starts by computing $s = V(J; 1)$, the number of real roots of the ideal J, and creating an element of the class $SIout$ by a call to the constructor of the class. The output of this initialisation step is given by:

$$\begin{aligned} SIMatrix &= (1) \\ lsig &= \{\{\}\} \\ lsol &= \{s\} \\ lpol &= (1) \\ lRSV &= (V(J; 1)) \\ SIdeal &= J \\ Dim &= 1 \\ Nroots &= s \end{aligned}$$

Note that the identity $SIMatrix \cdot lsol = lRSV$ holds because $(1)(s) = (V(J; 1))$.

If $s = 0$ (i.e. if the ideal has not real roots), the program exits. If not, the iterative process, $SIadd$, is applied for each polynomial in the list of constraints.

Iterative Process: SIAdd

The input of this step is the output of the previous one, an element of the class $SIout$, and a new polynomial constraint, Q_{l+1}. We start constructing a new system as a tensor product:

$$A = SIMatrix \otimes \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & -1 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} SIMatrix & SIMatrix & SIMatrix \\ 0 & SIMatrix & -SIMatrix \\ 0 & SIMatrix & SIMatrix \end{pmatrix}$$

$$X = lsol \otimes \begin{pmatrix} c_{\{0\}}(J; Q_{l+1}) \\ c_{\{+\}}(J; Q_{l+1}) \\ c_{\{-\}}(J; Q_{l+1}) \end{pmatrix} = \begin{pmatrix} c_{\varepsilon \cup \{0\}}(J; Q^{(l+1)})_{\varepsilon \in lsig} \\ c_{\varepsilon \cup \{+\}}(J; Q^{(l+1)})_{\varepsilon \in lsig} \\ c_{\varepsilon \cup \{-\}}(J; Q^{(l+1)})_{\varepsilon \in lsig} \end{pmatrix}$$

$$B = lRSV \otimes \begin{pmatrix} V(J; 1) \\ V(J; Q_{l+1}) \\ V(J; Q_{l+1}^2) \end{pmatrix} = \begin{pmatrix} V(J; H)_{H \in lpol} \\ V(J; H \cdot Q_{l+1})_{H \in lpol} \\ V(J; H \cdot Q_{l+1}^2)_{H \in lpol} \end{pmatrix}$$

Next, we call the $SISolve$ function (see next section for more details) to solve the system $A \cdot X = B$, and we reduce it in this way:

- Let $ind_1, \ldots ind_{newDim}$ be the sequence of indices corresponding to the non zero elements of the vector X.
- Drop the columns of the matrix A corresponding to the zeros of the vector X.
- Drop the rows of the resulting matrix needed in order to obtain an square matrix of full rank. This operation is performed by a call to the $MaxInvert$ function (see next section).
- Let $u_1, \ldots u_{newDim}$ be the sequence of indices of the rows of the matrix A which appears in the reduced matrix.
- Reconstruct $lRSV$ taking from B the elements corresponding to the lines of the matrix A we have chosen:

$$lRSV = (B[u_1], \ldots, B[u_{newDim}])$$

- Reconstruct *lpol* taking the polynomials corresponding to the elements of $lRSV$. *lpol* is given by the elements of indices $u_1, \ldots u_{newDim}$ in the list H defined by:

$$\begin{array}{ll} H(i) = lpol(i) & i = 1, \ldots, Dim \\ H(i + Dim) = lpol(i) \cdot Q_{l+1} & i = 1, \ldots, Dim \\ H(i + 2 \cdot Dim) = lpol(i) \cdot Q_{l+1}^2 & i = 1, \ldots, Dim \end{array}$$

- Reconstruct *lsol* with the non zero elements of the vector X:

$$lsol = \{X(ind_1), \ldots, X(ind_{newDim})\}$$

- Reconstruct *lsig* as the list of sign conditions corresponding to *lsol*, taking the elements of indices $ind_1, \ldots ind_{newDim}$ from the list L defined by:

$$\begin{array}{ll} L(i) = lsig(i) \cup \{0\} & i = 1, \ldots, Dim \\ L(i + Dim) = lsig(i) \cup \{+\} & i = 1, \ldots, Dim \\ L(i + 2 \cdot Dim) = lsig(i) \cup \{-\} & i = 1, \ldots, Dim \end{array}$$

In each step, some tests can be applied in order to minimize the number of calls to the V function (see [G89]):

- We compute $V(J; Q_{l+1})$:
  - *Case 1:* $V(J; Q_{l+1}) = Nroots$.
    All the roots of J take a positive value if evaluated in Q_{l+1}. We append " + " at the end of each list of signs. No modifications are needed for the rest:

$$lsig \rightarrow \{\varepsilon \cup \{+\}\}_{\varepsilon \in lsig}$$

  - *Case 2:* $V(J; Q_{l+1}) = -Nroots$.
    All the roots of J take a negative value if evaluated in Q_{l+1}. We append " − " at the end of each list of signs:

$$lsig \rightarrow \{\varepsilon \cup \{-\}\}_{\varepsilon \in lsig}$$

    The rest, as in the previous case, remains unchanged.

- If $V(J;Q_{l+1}) \neq Nroots$ and $V(J;Q_{l+1}) \neq -Nroots$, then we compute $V(J;Q_{l+1}^2)$:

  - *Case 3:* $V(J;Q_{l+1}^2) = 0$.
    All the roots of J are also roots of Q_{l+1}. We append "0" at the end of each list of signs:

    $$lsig \rightarrow \{\varepsilon \cup \{0\}\}_{\varepsilon \in lsig}$$

  - *Case 4:* $V(J;Q_{l+1}^2) = V(J;Q_{l+1})$.
    Each root of J is a root of Q_{l+1} or takes a positive value. Next relation follows:

    $$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} c_{\{0\}}(J;Q_{l+1}) \\ c_{\{+\}}(J;Q_{l+1}) \end{pmatrix} = \begin{pmatrix} V(J;1) \\ V(J;Q_{l+1}) \end{pmatrix}$$

    In this case, the system we construct starting $SIadd$ process is given by:

    $$A = SIMatrix \otimes \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

    $$X = lsol \otimes \begin{pmatrix} c_{\{0\}}(J;Q_{l+1}) \\ c_{\{+\}}(J;Q_{l+1}) \end{pmatrix}$$

    $$B = lRSV \otimes \begin{pmatrix} V(J;1) \\ V(J;Q_{l+1}) \end{pmatrix}$$

    Next, we solve the system and we reduce it as described before.

  - *Case 5:* $V(J;Q_{l+1}^2) = -V(J;Q_{l+1})$.
    Each root of J is a root of Q or takes a negative value.

    $$\begin{pmatrix} 1 & 1 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} c_{\{0\}}(J;Q_{l+1}) \\ c_{\{-\}}(J;Q_{l+1}) \end{pmatrix} = \begin{pmatrix} V(J;1) \\ V(J;Q_{l+1}) \end{pmatrix}$$

  - *Case 6:* $V(J;Q_{l+1}^2) = Nroots$.
    No root of J is at the same time a root of Q_{l+1}. All the roots of the ideal take a positive or negative value when evaluated in Q_{l+1}:

    $$\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} c_{\{+\}}(J;Q_{l+1}) \\ c_{\{-\}}(J;Q_{l+1}) \end{pmatrix} = \begin{pmatrix} V(J;1) \\ V(J;Q_{l+1}) \end{pmatrix}$$

– *Case 7:* General case.

Each root of J is a root of Q_{l+1} or takes a positive or negative value:

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & -1 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} c_{\{0\}}(J; Q_{l+1}) \\ c_{\{+\}}(J; Q_{l+1}) \\ c_{\{-\}}(J; Q_{l+1}) \end{pmatrix} = \begin{pmatrix} V(J;1) \\ V(J; Q_{l+1}) \\ V(J; Q_{l+1}^2) \end{pmatrix}$$

3.1 Some specialized Linear Algebra Tools

This section deals with the strategies used for :

- Solving the linear system $SIMatrix \cdot lsol = lRSV$
- Extracting from $SIMatrix$ an invertible sub-matrix with a fixed rank, using choosen columns of $SIMatrix$ (those that correspond to non null coordinates of $lsol$).

These two algorithms are called frequently in the algorithm, they have to be as efficient as possible.

Since $SIMatrix \cdot lsol = lRSV$ is a linear system of diophantine equations, a first method consists in using Bareiss-like methods to prevent from using rational numbers.

3.1.1 IntSISolve

IntSISolve takes a square matrix A and a vector V and returns a vector X such that $AX = V$ assuming that A is invertible.

Using Bareiss relations (see [BAR68]), we put A into a row-echelon form using elementary fraction free manipulations on its lines and doing the same operations on V. We obtain then a triangular system $A'X = V'$ that have exactly the same solution than the original one. Since it is a system of diophantine equations , we know that the divisions involved to solve the triangular system are exact and so we can solve the system without using rational numbers.

3.1.2 IntMaxInvert

IntMaxInvert takes as arguments, a square matrix A with integer coefficients, a list of indices $cind$ and an integer d, and returns an invertible matrix B of rank d extracted from the rectangular matrix $A[i][ind[j]]$, and the list of $rind$ row indices of $A[i][ind[j]]$ that has been used to construct B.

Also, if $A \cdot X = V$ then we will obtain: $B \cdot X' = V'$ where:

- $X'[i] = X[cind[i]]$
- $V'[i] = V[dind[i]]$

Since, in the SI algorithm, the coordinates of V correspond to the computation of $V(J; H)$ for a given polynomial H, this polynomials must have degrees as small as possible to optimize the algorithm and so, in IntMaxInvert, this means that we have to take first the lines of $A[i][ind[j]]$ that have the smallest indices to construct B.

This can be done using an appropriate strategy for choosing a new pivot in the row-echelon algorithm, searching first in the current line instead of in the current column.

Combining these operations with Bareiss relations, we obtain a fraction-free specialized algorithm.

Unfortunately, such methods implies the use of a multi-precision arithmetic for the computations because of the growth of the coefficients in the Bareiss method.

A second idea is to use modular arithmetic. Let's study the size of the involved data:

- The coordinates of $lsol$ represent the number of real roots realized by some sign conditions, so they are positive and their sum is bounded by the total number of real roots.
- In the same way, we can remark that each coordinate $lRSV[i]$ of $lRSV$ verify the following relation:

  $$-Nroots \leq lRSV[i] \leq Nroots$$

  where $Nroots$ is the total number of distincts real roots.

- The entries of $SIMatrix$ stand in $\{0, 1, -1\}$

So, let suppose that p is a prime integer so that $p > 2 \cdot Nroots$ and study $SIMatrix \cdot lsol = lRSV$ as a linear system with p-modular coefficients (denoting by $\overline{SIMatrix}$ (resp. $\overline{lsol}$, $\overline{lRSV}$) the expression of $SIMatrix$ (resp. $lsol$, $lRSV$) reduced modulus p).

If $\overline{SIMatrix}$ is invertible, then there is an unique solution $\overline{lsol}$ giving a family of potential solutions $lsol$ so that $lsol[i] = \overline{lsol}[i] + k_i \cdot p$, $i = 1 \ldots dim(SIMatrix)$. But because all the coordinates of $lsol$ are positive and bounded by p, taking $k_i = 0 \ \forall i \in 1 \ldots dim(SIMatrix)$, we obtain the expected solution.

To be sure that $\overline{SIMatrix}$ is invertible at each step, we use the following recursive method:

At the first step,

$$SIMatrix = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & -1 \\ 0 & 1 & 1 \end{pmatrix}$$

so $det(SIMatrix) = 2$ implies that $SIMatrix$ is invertible modulus $p > 2$ and we can use the method precedently described to solve $SIMatrix \cdot lsol = lRSV$. Since $\overline{SIMatrix}$ is invertible, we can find a sub-matrix B of full rank that is invertible modulus p using an algorithm similar to $IntMaxInvert$, working with modular arithmetic (using the classical gaussian reduction for example).

At the next step, $SIMatrix$ is defined as the tensor product: $M \otimes B$ where

$$M \in \{(1), \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 0 & -1 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & -1 \\ 0 & 1 & 1 \end{pmatrix}\}$$

Since, in each case, $det(M)$ is a power of 2, the relation :

$$det(M \otimes B) = det(M)^{dim(B)} \cdot det(B)^{dim(M)}$$

shows that $M \otimes B$ is invertible modulus p and so the modular solver described before can be used at each step.

These different results give two algorithms **ZpSISolve** and **ZpMaxInvert** that can replace **IntSISolve** and **IntMaxInvert**.

4 Real Algebraic Numbers Coding: RAN Algorithm

The algorithm we present in this section has been introduced by Coste and Roy ([CR88]). It is based on the following fact:

Let P be a polynomial of degree d with integer coefficients, $P', \ldots, P^{(d-1)}$ its derivatives. Let α and β be two real roots of P. If $sign(P^{(i)}(\alpha)) = sign(P^{(i)}(\beta))$ $(i = 1, \ldots, d-1)$, then $\alpha = \beta$.

This result is a consequence of Thom's lemma (see [BCR87]) and provides a caracterization of a real root of a polynomial P by the sequence of signs it takes when evaluated in the derivatives of P.

4.1 Algorithm RAN

Input:

- A polynomial P.

Output:

- An element of the class *SIout* where *lsig* contains the coding of the real roots of P.

The algorithm consists on applying the SI algorithm to the derivatives of P. Moreover, some strategies are applied (see [G89] or [RS90]), in order to reduce the computational cost:

- The algorithm stops when all the elements of *lsol* are "1". (That means that there is exactly one root of P which realize each sign condition, and then all the roots are distinguised and we do not need to compute the signs realized over the rest of derivatives).

- The list of derivatives is given in decreasing order: $P^{(d-1)}, \ldots, P'$. Like this, we start the computation with the polynomials of smaller degree.

- We replace P^i by $\frac{P^i}{i!}$, $(i = 1, \ldots, d-1)$, reducing in this way the coefficients of the polynomials we deal with.

5 Examples

Example 1: Univariate case. Caracterization of the zeroes of the polynomial:

$$P = 21x^5 + x^4 - 2x^3 + 162x^2 - 13x - 1$$

Output of RAN :

```
RAN( {21*T**5,1*T**4,-2*T**3,162*T**2,-13*T**1,-1} )


{ {+,+} , 1 }
{ {-,+} , 1 }
{ {-,-} , 1 }

User Time: 0,110000 sec
```

This means that there are tree zeroes, α_1, α_2 and α_3 such that:

$$\begin{aligned}\alpha_1 &\equiv [P = 0, P^{(4)} > 0, P^{(3)} > 0] \\ \alpha_2 &\equiv [P = 0, P^{(4)} < 0, P^{(3)} > 0] \\ \alpha_3 &\equiv [P = 0, P^{(4)} < 0, P^{(3)} < 0]\end{aligned}$$

Example 2: Multivariate case. Cassou-Noguès (from the PoSSo test suit data base) :

$$\begin{aligned}&15b^4cd^2 + 6b^4c^3 + 21b^4c^2d - 144b^2c - 8b^2c^2e - 28b^2cde - 648b^2d \\ &\quad +36b^2d^2e + 9b^4d^3 - 120\end{aligned}$$

$$\begin{aligned}&30c^3b^4d - 32de^2c - 720db^2c - 24c^3b^2e - 432c^2b^2 + 576ec - 576de \\ &\quad +16cb^2d^2e + 16d^2e^2 + 16e^2c^2 + 9c^4b^4 + 5184 + 39d^2b^4c^2 \\ &\quad +18d^3b^4c - 432d^2b^2 + 24d^3b^2e - 16c^2b^2de - 240c\end{aligned}$$

$$\begin{aligned}&216db^2c - 162d^2b^2 - 81c^2b^2 + 5184 + 1008ec - 1008de + 15c^2b^2de \\ &\quad -15c^3b^2e - 80de^2c + 40d^2e^2 + 40e^2c^2\end{aligned}$$

$$261 + 4db^2c - 3d^2b^2 - 4c^2b^2 + 22ec - 22de$$

Variables: $\{b, c, d, e\}$
List of constraints : $\{b, c, d, e\}$

Output of SI :

```
 SI Algorithm
 ------------

 Number of distinct real roots : 4

 For the list of constraints :
{{b},{c},{d},{e}}

 { {+,+,+,+} , 1 }
 { {-,+,+,+} , 1 }
 { {+,+,+,-} , 1 }
 { {-,+,+,-} , 1 }

        (SI) User Time : 6,720000 sec

        (RealSolving) User Time : 7,230000 sec

        (Global) User Time : 49,500000 sec
```

This means:

- There is 1 real solution where $b > 0$ $c > 0$ $d > 0$ $e > 0$
- There is 1 real solution where $b < 0$ $c > 0$ $d > 0$ $e > 0$
- There is 1 real solution where $b > 0$ $c > 0$ $d > 0$ $e < 0$
- There is 1 real solution where $b < 0$ $c > 0$ $d > 0$ $e < 0$

One table of timings

In this table we show timings for RAN algorithm over a collection of univariate polynomials. Times are given in seconds. The computations has been performed on a SPARCStation 10, 80 MB.

	PoSSo	Axiom
Feng	33,50	794.83
Mignotte_51_974	15,48	5967.53
Mignotte_67_3245	41,57	5859.37
Mignotte_89_3523	119,74	39706.67
Wilkinson_354_16	48,79	307.32
Wilkinson_87_21	253,08	1819.85

- *Feng* represents the discriminant in respect of y of the polynomial (from the PoSSo test suit data base):

$$\begin{aligned} & 2x^8 + 4x^6y^2 + 3x^4y^4 + 2x^2y^6 + y^8 - 83x^6y - 88x^4y^3 - 47x^2y^5 - 42y^7 \\ & +398x^6 + 1323x^4y^2 + 401x^2y^4 + 682y^6 - 8637x^4y - 2241x^2y^3 \\ & -5262y^5 + 20200x^4 + 12259x^2y^2 + 18741y^4 - 30300x^2y \\ & -24220y^3 + 10100y^2 \end{aligned}$$

- *Mignotte_d_a* represents the polynomial:

$$P = x^d - 2(ax - 1)^2$$

- *Wilkinson_M_n* represents the polynomial:

$$x^{n-1} + M \cdot \prod_{i=0}^{n}(x - i)$$

References

[BAR68] Bareiss E. H: *Sylvester's identity and multistep integer preserving Gaussian elimination.* Math. Comp. 22, 565-578 (1968).

[BCR87] J. Bochnak, M. Coste and M. F. Roy: *Géometrie algébrique réelle.* Ergebnisse der Mathematik. Berlin: Springer-Verlag (1987).

[BKR86] M. Ben-Or, D. Kozen and J. Reif: *The complexity of elementary algebra and geometry.* Journal of Computation and Systems Sciences 32, 251–264 (1986).

[CR88] M. Coste and M.F. Roy: *Thom's lemma, the coding of real algebraic numbers and the topology of semialgebraic sets.* Journal of Symbolic Computation 5, 121-129 (1988).

[G89] L. González Vega: *La sucesión de Sturm-Habicht y sus aplicaciones al Algebra Computacional.* Doctoral Thesis. Universidad de Cantabria. (1989).

[RS90] M.F. Roy and A. Szpirglas: *Complexity of Computation on Real Algebraic Numbers.* Journal of Symbolic Computation 10, 39-51 (1990).