

# Manual of SARAG (Some Algorithms in Real Algebraic Geometry)

SARAG was developed by Fabrizio Caruso under the scientific guidance of Marie-Françoise Roy at the University of Rennes 1, France, with support from the RAAG network, further developed at the University of Pisa, Italy, modified by Alexandre Le Meur and Marie-Françoise Roy at the University of Rennes 1. This manual has been written by Alexandre Le Meur.

Please report bugs to: [fabrizio.caruso@posso.dm.unipi.it](mailto:fabrizio.caruso@posso.dm.unipi.it) or [marie-francoise.roy@univ-rennes1.fr](mailto:marie-francoise.roy@univ-rennes1.fr)

Also part of the free interactive book "Algorithms in Real Algebraic Geometry" by Saugata Basu, Richard Pollack, Marie-Françoise Roy Springer-Verlag, Berlin, 2003 (second edition 2006) which together with SARAG is available for download at <http://name.math.univ-rennes1.fr/marie-francoise.roy/bpr-posted2.html>

## 1 Introduction

### 1.1 The SARAG Library

Free open source Maxima library for some algorithms of real algebraic geometry.

SARAG provides functions for linear algebra, theory of subresultants, Cauchy Index and its application to real root counting, isolation of real roots, sign determination, Thom encodings as well as certificates of positivity in the univariate case.

### 1.2 Requirements

- MAXIMA (5.9.2 or above) The library has been initially developed and tested with Maxima version 5.9.2 and 5.9.3. SARAG with the only exception of plotting (e.g., "drawTopology") also works on Maxima 5.9.1. The latest version of Maxima tested so far is Maxima 5.20.1. The latest version of Maxima is available on line at <http://maxima.sourceforge.net/>.

- GNUPLOT (3.7.x, 4.0.x or above) The library uses Gnuplot for plotting graphs and it has been tested successfully on Gnuplot 3.7.x and 4.0.x.

It is also useful to install  $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$  which has a Maxima plugin, if you want to use this manual interactively for example (using saragmanual.tm rather than saragmanual.pdf).

### 1.3 Loading the files

You can work directly in Maxima, or use the Maxima plugin inside  $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$  (which is done in the examples given in this manual).

file : sarag.mac

The library is contained in the following files:

sarag.mac (it loads all the files, settings.mac (general settings), constants.mac, sarag\_initialization.mac ,aliases.mac (name conventions), lowLevel.mac (low level routines), sarag\_linear\_algebra.mac (linear algebra and matrix manipulation) ,root-Counting.mac (real root counting), rootIsolation.mac (root isolation by De Casteljaun method), signDetermination.mac (sign determination), intervalArithmetic.mac (interval arithmetic), topology.mac (topology of curves), certificatesOfPositivity (certificates of positivity in the univariate case), multiCertificatesOfPositivity (certificates of positivity in the multivariate case), arag\_test.mac (test file for the book "Algorithms in Real Algebraic Geometry") hard\_test.mac (test files with mostly topology computations), saragmanuel.tm and saragmanual.pdf (this manual).

ATTENTION: The variables of the different functions are explicated in the file named « variables ».

In order to load the library either load the file "sarag.mac" that will load all the files if they are in the same directory indicated by **path** : `load("path/sarag.mac")` or load each single file with the "load" Maxima command.

Example: if the library is contained in a folder named SARAG in the directory where you are working type

`load("./SARAG/sarag.mac");`

- Load the library

**(%i1)** `load("./SARAG/sarag.mac");`

```
(%o1) ./SARAG/sarag.mac
```

In order to test the library you can use:

```
batch("path/arag_test.mac",test);
```

and

```
batch("path/hard_test.mac",test);
```

**path** must be replaced by the path of the folder containing the library :

exemple :

```
batch("./SARAG/arag_test.mac",test)
```

```
batch("./SARAG/hard_test.mac",test);
```

If there are errors they are reported in arag\_test.ERR and hard\_test.ERR. Don't pay attention to the output you get in Maxima, just open these files.

- Test the library

```
(%i2) load("./SARAG/sarag.mac");
```

```
(%o2) ./SARAG/sarag.mac
```

```
(%i3) batch("./SARAG/arag_test.mac",test);
```

```
(%i3) batch("./SARAG/hard_test.mac",test);
```

```
(%i2)
```

## 1.4 Manual

file : manual.tm and manual.pdf

This manual describes the high level functions ("main functions") of the SARAG library and the most important auxiliary functions.

The file `manual.tm` is a  $\text{\TeX}_{\text{MACS}}$  file in which you can work interactively using Maxima plugin.

For more details and for the theory behind the algorithms we refer to "Algorithms in Real Algebraic Geometry" by S. Basu, R. Pollack, M.-F. Roy which together with SARAG is available for download at <http://name.math.univ-rennes1.fr/marie-francoise.roy/bpr-posted1.html> and, for certificate of positivity, to F. Boudaoud, F. Caruso M.-F. Roy Certificates of positivity in the Bernstein basis, Discrete and Computational Geometry 39 4 639-655 (2008)

## 1.5 Naming conventions

file : `aliases.mac`

The names of main functions are formed by adding prefixes to specify the method/algorithm which is used and suffixes to specify an alternative version of the function or output.

When no prefix or no suffix is used the function with the default method/algorithm will be called as set in the file: `"aliases.mac"`.

Auxiliary functions may not follow such conventions.

## 1.6 Form of the output

The output of all functions of the library is a Maxima expression. Maxima uses brackets "[", "]" to describe couples and lists (e.g. an open interval (a,b) is described by a couple containing the ends, which in Maxima is "[a,b]")

## 2 Functions and syntax

In this Section, all functions are described, with syntax and arguments. For each function we use an example to show how the function works and what it returns. The code of each function can be found in the corresponding file. Moreover, a maxima window untitled « use it yourself » is provided to test functions with your own examples.

For several functions the library provides different methods for computing the result. This is indicated in the syntax part of a description in the paragraph Method. The default method, `sert` is `aliases.mac` is indicated with a `*`. Methods not implemented yet are indicated in grey.

## 2.1 Linear algebra

The corresponding results are explained in "Algorithms in Real Algebraic Geometry" by S. Basu, R. Pollack, M.-F. Roy, mainly in Chapter 8.

*file : sarag\_linear\_algebra.mac*

This file contains functions related to Gaussian elimination and matrix manipulations (products, determinant, etc.)

### 2.1.1 Matrix product

**Matrix Product:..** Computes the product of two matrices

- Syntax
  - Name : `matrixProd(A,B)`
  - Input : A and B are lists of lists representing the rows of two matrices such that A has the same number of columns as B as rows
  - Output : the row-column product matrix of A, B
- Example

```
(%i3) load("./SARAG/sarag.mac");
(%o6) ./SARAG/sarag.mac
(%i7) matrixProd([[1,0,1],[0,1,0],[1,1,0]], [[1,0,0],[0,1,0],[0,0,1]]);
(%o7) [[1,0,1],[0,1,0],[1,1,0]]
(%i8)
```

- Code

The code can be found in `./SARAG/sarag_linearAlgebra.mac`

- Use it yourself



### 2.1.2 Determinant, elimination

#### Main Functions :

**Determinant:.** Computes the determinant of a square matrix

- Syntax
  - Name : `det(M)`
  - Input : `M` is a list of lists representing the rows of a matrix
  - Method :
    - Gauss [Gaussian elimination with no pivot optimization]
    - Bareiss [Bareiss method]\*

Output : the determinant of `M`

- Example

```
(%i8) load("./SARAG/sarag.mac");
(%o8) ./SARAG/sarag.mac
(%i9) det([[1,0,0],[0,1,0],[0,0,1]]);
(%o9) 1
(%i10) gaussDet([[1,0,0],[0,1,0],[0,0,1]]);
(%o10) 1
(%i11) bareissDet([[1,0,0],[0,1,0],[0,0,1]]);
(%o11) 1
(%i12)
```

- Code
  - The code can be found in `./SARAG/sarag_linearAlgebra.mac`
  - Use it yourself

□

**Elimination:.** Computes the upper triangular equivalent form of a matrix

- Syntax
  - Name : `elim(M)`
  - Input : M is a list of lists representing the rows of a matrix
  - Method :
    - Gauss [Gaussian elimination with no pivot optimization]
    - Bareiss [Bareiss method]

Output : [t,c,z] where :

- t is an upper triangular equivalent form of M with possibly some zero rows computed by Gaussian elimination with divisions and columns exchanges
- c is the list of couples describing the columns exchanges
- z is the list of zero rows

- Example

```
(%i2) load("./SARAG/sarag.mac");
(%o2) ./SARAG/sarag.mac
(%i7) elim([[0,1,0],[0,0,1],[1,0,0]]);
(%o7) [[[1,0,0],[0,1,0],[0,0,1]],[[0,1],[1,2]],[]]
(%i8)
```

- Code

The code can be found in `./SARAG/sarag_linearAlgebra.mac`

- Use it yourself

□

### 2.1.3 Characteristic polynomial

#### Auxiliary Functions :

**Polynomial From Newton ::** Computes the polynomial corresponding to given Newton Sums

- Syntax

Name : `polyFromNewton(ns,var)`

Input : `ns` is a list containing the Newton sums of a given polynomial in variable `var`

Output : the corresponding polynomial

- Example

```
(%i5) load("./SARAG/sarag.mac");
```

```
(%o5) ./SARAG/sarag.mac
```

```
(%i13) polyFromNewton([3,0,2,0],x);
```

```
(%o13)  $x^3 - x$ 
```

```
(%i14)
```

- Code

The code can be found in `./SARAG/sarag_linearAlgebra.mac`

- Use it yourself

□

**Newton From Polynomial ::** Computes the Newton Sums corresponding to a given Polynomial.

- Syntax

Name : `newtonFromPoly(P,var,len)`

Input : `P` is a polynomial in variable `var`, `len` is a non negative integer

Output : array containing the `n+1` newton sums of `P`



- Example

```
(%i16) load("./SARAG/sarag.mac");
(%o16) ./SARAG/sarag.mac
(%i17) newtonFromPoly(x^3-x,x,3);
(%o17) [3, 0, 2, 0]
(%i18)
```

- Code

The code can be found in `./SARAG/sarag_linearAlgebra.mac`

- Use it yourself

### Main Functions :



**Characteristic Polynomial:.** Computes the characteristic polynomial of a square matrix

- Syntax

Name : `charPol(A,var)`

Input : A is a list of lists representing the rows of a matrix

var is the variable of the polynomial (for example x)

Method :

- Gauss [Gaussian elimination with no pivot optimization]
- Bareiss [Bareiss method]
- Baby Giant [baby step, giant step trace-based method]\*

Output : the characteristic polynomial of A in the indeterminate var

- Example

```
(%i12) load("./SARAG/sarag.mac");
```

```
(%o12) ./SARAG/sarag.mac
(%i13) charPol([[1,2,3,4], [2,3,4,5], [3,4,5,6], [5,6,7,8]],x);
(%o13)  $x^4 - 17x^3 - 26x^2$ 
(%i14) babyGiantCharPol([[1,2,3,4], [2,3,4,5], [3,4,5,6], [5,6,7,8]],x);
(%o14)  $x^4 - 17x^3 - 26x^2$ 
(%i15) gaussCharPol([[1,2,3,4], [2,3,4,5], [3,4,5,6], [5,6,7,8]],x);
(%o15)  $x^4 - 17x^3 - 26x^2$ 
(%i16)
```

- Code
  - The code can be found in `./SARAG/sarag_linearAlgebra.mac`
  - Use it yourself

**Descartes Signature:.** Computes the signature of a symmetric matrix by the Descartes' Rule applied to the characteristic polynomial □

- Syntax
  - Name : `descartesSignature(M)`
  - Input : M is a list of lists representing a square symmetric matrix
  - Output : the signature of M
- Example

```
(%i33) load("./SARAG/sarag.mac");
(%o33) ./SARAG/sarag.mac
(%i34) descartesSignature([[1,0,0], [0,1,0], [0,0,1]]);
(%o34) 3
```

(%i35)

- Code
  - The code can be found in `./SARAG/sarag_linearAlgebra.mac`
  - Use it yourself

□

## 2.2 Root counting

*file : rootCounting.mac*

### 2.2.1 Signes changes

The corresponding results are explained in "Algorithms in Real Algebraic Geometry" by S. Basu, R. Pollack, M.-F. Roy, mainly in Chapter 2 and 9.

**Sign Changes:.** Counts the number of sign changes in a sequence of real numbers

- Syntax
  - Name : `signChanges(seq)`
  - Input : `seq` is a sequence of number
  - Output : number of sign changes in the sequence
- Example

```
(%i35) load("./SARAG/sarag.mac");
(%o35) ./SARAG/sarag.mac
(%i36) signChanges([1,0,0,1,2,3,-8,0,1]);
(%o36) 2
(%i37)
```

- Code

The code can be found in `./SARAG/rootCounting.mac`

- Use it yourself

**Modified Sign Changes:.** Slight modification of the number of sign changes in a sequence of real numbers taking into account existing zeroes in the sequence. □

- Syntax

Name : `modifiedSignChanges(seq)`

Input : `seq` is a sequence of number

Output : number of sign changes in the sequence. Warning! Annulations count for a sign change!

- Example

```
(%i41) load("./SARAG/sarag.mac");
```

```
(%o41) ./SARAG/sarag.mac
```

```
(%i42) modifiedSignChanges([1,0,0,1,2,3,-8,0,1]);
```

```
(%o42) 4
```

```
(%i43)
```

- Code

The code can be found in `./SARAG/rootCounting.mac`

- Use it yourself

**Generalized Permanences Minus Variations:.** Computes a generalization of the difference between Permanences and Variations of a sequence □

- Syntax

Name: `genPermMVar(seq)`

Input: `seq` a sequence of elements in an ordered integral domain

Output: `PmV(seq)`

- Example

```
(%i6) load("./SARAG/sarag.mac");
(%o6) ./SARAG/sarag.mac
(%i23) genPermMVar([1,0,0,1,2,3,-8,0,1]);
(%o23) 0
(%i24)
```

- Code
  - The code can be found in `./SARAG/rootCounting.mac`
  - Use it yourself

□

### 2.2.2 Signed remainder sequence

The corresponding results are explained in "Algorithms in Real Algebraic Geometry" by S. Basu, R. Pollack, M.-F. Roy, mainly in Chapter 2 and 8.

**Signed Remainder Sequence:.** Computes the signed remainder sequence of two polynomials

- Syntax
  - Name : `sRem(P,Q,var)`
  - Input : `P` and `Q` are polynomials
  - `var` is the variable of the polynomials (for example `x`)

Output : list containing signed remainder sequence

- Example

```
(%i37) load("./SARAG/sarag.mac");
(%o37) ./SARAG/sarag.mac
(%i38) sRem(x^11-x^10+1,11*x^10-10*x^9,x);
```

$$(\%o38) \left[ x^{11} - x^{10} + 1, 11x^{10} - 10x^9, \frac{10x^9 - 121}{121}, -\frac{1331x - 1210}{10}, \frac{275311670611}{285311670611} \right]$$

(%i39)

- Code
  - The code can be found in `./SARAG/rootCounting.mac`
  - Use it yourself

**Extended Signed Remainder Sequence:.** Same as previous function but include also the corresponding sequence of cofactors □

- Syntax
  - Name : `sRemExt(P,Q,var)`
  - Input : P and Q are polynomials
  - var is the variable of the polynomials (for example x)

Output :  $[r,u,v]$  forming the extended signed remainder sequence where r is the signed remainder sequence and u and v are the corresponding sequences of cofactors

- Example

(%i39) `load("./SARAG/sarag.mac");`

(%o39) `./SARAG/sarag.mac`

(%i40) `sRemExt(x^3+2*x^2+3*x+4,3*x^2+4*x+3,x);`

$$(\%o40) \left[ \left[ x^3 + 2x^2 + 3x + 4, 3x^2 + 4x + 3, -\frac{10x + 30}{9}, -18 \right], \left[ 1, 0, -1, \frac{27x}{10} - \frac{9}{2} \right], \left[ 0, 1, \frac{x}{3} + \frac{2}{9}, \frac{9x}{10} - \frac{9x^2}{10} \right] \right]$$

(%i41)

- Code

The code can be found in `./SARAG/rootCounting.mac`

- Use it yourself

□

### 2.2.3 Subresultants

The corresponding results are explained in "Algorithms in Real Algebraic Geometry" by S. Basu, R. Pollack, M.-F. Roy, mainly in Chapter 4 and 8.

**Signed Subresultant Polynomials:.** Computes the list of the signed subresultant polynomials of two polynomials

- Syntax

Name : `sSubResPol(P,Q,var)`

Input : P and Q are polynomials

var is the variable of the polynomials (for example x)

Output : `[sSubRes]` where `sSubRes` is a list containing the signed subresultant sequence

- Example

```
(%i43) load("./SARAG/sarag.mac");
```

```
(%o43) ./SARAG/sarag.mac
```

```
(%i44) sSubResPol(x^4+a*x^2+b*x+c, diff(x^4+a*x^2+b*x+c,x),x);
```

```
(%o44) [x^4 + a x^2 + b x + c, 4 x^3 + 2 a x + b, - 8 a x^2 - 12 b x - 16 c, 32 a c x - 36 b^2 x - 8 a^3 x - 48 b c - 4 a^2 b, 256 c^3 - 128 a^2 c^2 + 144 a b^2 c + 16 a^4 c - 27 b^4 - 4 a^3 b^2]
```

```
(%i45)
```

- Code

The code can be found in `./SARAG/rootCounting.mac`

- Use it yourself



**Sturm Sequence:.** Computes the Sturm sequence of a polynomial

- Syntax
  - Name : sturmSequence(P,var)
  - Input : P polynomial
    - var is the variable of the polynomial (for example x)
  - Output : the Sturm sequence of P (signed remainder sequence of P and its derivative)
- Example

```
(%i73) load("./SARAG/sarag.mac");
```

```
(%o73) ./SARAG/sarag.mac
```

```
(%i74) sturmSequence(x^4+x+2,x);
```

```
(%o74)  $\left[ x^4 + x + 2, 4x^3 + 1, -\frac{3x + 8}{4}, \frac{2021}{27} \right]$ 
```

```
(%i75)
```

- Code
  - The code can be found in ./SARAG/rootCounting.mac
  - Use it yourself



**Signed Subresultant Coefficients.** Computes the signed subresultant coefficients of two polynomials

- Syntax
  - Name : sSubResCoeff(P,Q,var)
  - Input : P is a polynomial in variable var
  - Output : the coeffs of subdiscriminant of the polynomials P and Q
- Example

```
(%i49) load("./SARAG/sarag.mac");
```



```
(%o49) ./SARAG/sarag.mac
(%i52) sSubResCoeff(x^3+a*x+b,diff(x^3+a*x+b,x),x);
(%o52) [1,3,-6 a,-27 b^2-4 a^3]
(%i53)
```

- Code

The code can be found in ./SARAG/rootCounting.mac

- Use it yourself

**Extended Signed Subresultant Sequence:.** Same as previous but includes also the corresponding sequence of cofactors □

- Syntax

Name : sSubResExt(P,Q,var)

Input : P and Q are polynomials

var is the variable of the polynomials (for example x)

Output : [sSubRes, s, u, v] where

1. sSubRes is a list containing the signed subresultant sequence
2. u, v are the corresponding cofactors

- Example

```
(%i53) load("./SARAG/sarag.mac");
(%o53) ./SARAG/sarag.mac
(%i54) sSubResExt(x^4+b*x+c,4*x^3+b,x);
(%o54) [[x^4+b x+c,4 x^3+b,-12 b x-16 c,-36 b^2 x-48 b c,256 c^3-27 b^4],[1,0,-16,-48 b,144 b^2 x^2-192 b c x+256 c^2,-1024 c^3 x^3+108 b^4 x^3-256 b c^3+27 b^5],[0,1,4 x,12 b x,-36 b^2 x^3+48 b c x^2-64 c^2 x-27 b^3,256 c^3 x^4-27 b^4 x^4+256 b c^3 x-27 b^5 x+256 c^4-27 b^4 c]]
```

(%i55)

- Code

The code can be found in `./SARAG/rootCounting.mac`

- Use it yourself

**GCD free part.** Computes the GCD of two polynomials  $P$  and  $Q$  as well as  $P$  divided by this GCD

□

- Syntax

Name : `gcdFreePart(P,Q,var)`

Input :  $P$  and  $Q$  are polynomials

var is the variable of the polynomials (for example  $x$ )

Output :  $[g,f]$  where

1.  $g$  is a  $\gcd(P,Q)$
2.  $f$  is a gcd-free part of  $P$  with respect to  $Q$

Name : `gcdFreePartWithZ(P,Q,var)`

Input :  $P$  and  $Q$  are polynomials with integer coefficients

var is the variable of the polynomials (for example  $x$ )

Output :  $[g,f]$  where

1.  $g$  is a  $\gcd(P,Q)$
2.  $f$  is a gcd-free part of  $P$  with respect to  $Q$

- Example

(%i55) `load("./SARAG/sarag.mac");`

(%o55) `./SARAG/sarag.mac`

(%i58) `gcdFreePart((x^2-1)*(x+2),(x+1)*x,x);`

(%o58)  $[2x + 2, 2x^2 + 2x - 4]$

```
(%i57) gcdFreePartWithZ((x^2-1)*(x+2),(x+1)*x,x);
```

```
(%o57)  $[x + 1, x^2 + x - 2]$ 
```

```
(%i58)
```

- Code
  - The code can be found in `./SARAG/rootCounting.mac`
  - Use it yourself

**Sylvester Resultant.** Computes the Sylvester Resultant of two polynomials

□

- Syntax
  - Name : `sylvesterResultant(P,Q,var)`
  - Input : P and Q are polynomials
    - var is the variable of the polynomials (for example x)
  - Output : the resultant of P and Q (i.e. the determinant of the Sylvester Matrix of P and Q)
- Example

```
(%i59) load("./SARAG/sarag.mac");
```

```
(%o59) ./SARAG/sarag.mac
```

```
(%i61) sylvesterResultant(a*x^2+b*x+c,2*a*x+b,x);
```

```
(%o61)  $4a^2c - ab^2$ 
```

```
(%i62)
```

- Code
  - The code can be found in `./SARAG/rootCounting.mac`
  - Use it yourself

**Discriminant.** Computes the discriminant of a polynomial

□

- Syntax

Name : `discriminant(P,var)`

Input : P is a polynomial in variable var

Output : the discriminant of the polynomial P (product of the square of the differences of the roots)

- Example

```
(%i62) load("./SARAG/sarag.mac");
```

```
(%o62) ./SARAG/sarag.mac
```

```
(%i63) discriminant(x^3+p*x+q,x);
```

```
(%o63)  $-27 q^2 - 4 p^3$ 
```

```
(%i64)
```

- Code

The code can be found in `./SARAG/rootCounting.mac`

- Use it yourself

□

**Subdiscriminant.** Computes the subdiscriminant coefficients of a polynomial

- Syntax

Name : `subDiscrCoeff(P,var)`

Input : P is a polynomial in variable var

Output : the subdiscriminants of the polynomial P

- Example

```
(%i68) load("./SARAG/sarag.mac");
```

```
(%o68) ./SARAG/sarag.mac
```

```
(%i69) subDiscrCoeff(x^3+p*x+q,x);
```

```
(%o69)  $[1, 3, -6 p, -27 q^2 - 4 p^3]$ 
```

(%i70)

- Code
  - The code can be found in `./SARAG/rootCounting.mac`
  - Use it yourself

□

### 2.2.4 Root counting

The corresponding results are explained in "Algorithms in Real Algebraic Geometry" by S. Basu, R. Pollack, M.-F. Roy, mainly in Chapter 9.

*file : rootcounting.mac*  
*rootIsolation.mac* (for De Casteljaun-based's root counting method)

#### Main Functions :

**Cauchy Index.** Computes the Cauchy Index of a rational function

- Syntax
  - Name : `cauchyIndex(Q,P,var)`
  - Input : Q, P polynomials  
         var is the variable of the polynomials (for example x)
  - Method :
    - i. sRem [signed remainder sequence]
    - ii. sSubRes [signed subresultants]\*

Output : Cauchy index of Q/P in the whole real line

- Example

```
(%i75) load("./SARAG/sarag.mac");
(%o75) ./SARAG/sarag.mac
(%i76) cauchyIndex( (x-5)*(x-4)*(x-2)*(x+1)*(x+2)*(x+4), (x-3)^2*(x-1)*(x+3),x);
(%o76) 0
(%i78) sRemCauchyIndex( (x-5)*(x-4)*(x-2)*(x+1)*(x+2)*(x+4), (x-3)^2*(x-1)*(x+3),x);
(%o78) 0
(%i79) sSubResCauchyIndex( (x-5)*(x-4)*(x-2)*(x+1)*(x+2)*(x+4), (x-3)^2*(x-1)*(x+3),x);
(%o79) 0
(%i80)
```

- Code
  - The code can be found in ./SARAG/rootCounting.mac
  - Use it yourself

**Tarski Query.** Computes the Tarski Query of two polynomials  $Q, P$  which is the difference between the number of roots of  $P$  where  $Q$  is positive and the number of roots of  $P$  where  $Q$  is negative □

- Syntax
  - Name : tarskiQuery(Q,P,var)
  - Input : Q, P polynomials
    - var is the variable of the polynomials (for example x)
  - Method :
    - i. sRem [signed remainder sequence]
    - ii. sSubRes [signed subresultants]\*

Output : Tarski query of  $Q, P$

- Example

```
(%i82) load("./SARAG/sarag.mac");
(%o82) ./SARAG/sarag.mac
(%i83) tarskiQuery(1,x^11-x^10+1,x);
(%o83) 1
(%i84) sSubResTarskiQuery(1,x^11-x^10+1,x);
(%o84) 1
(%i85) sRemTarskiQuery(1,x^11-x^10+1,x);
(%o85) 1
(%i86)
```

- Code

The code can be found in `./SARAG/rootCounting.mac`

- Use it yourself

**Number of Roots.** Computes the number of real roots of a polynomial

□

- Syntax

Name : `numberOfRoots(P,var)`

Input : P polynomial

var is the variable of the polynomial (for example x)

Method :

- i. sRem [signed remainder sequence]
- ii. sSubRes [signed subresultants]
- iii. deCasteljau [De Casteljau method for isolation]\*

iv. monomial [monomial method for isolation]

Output : number of roots of P in the whole real line

- Example

```
(%i86) load("./SARAG/sarag.mac");
(%o86) ./SARAG/sarag.mac
(%i89) numberOfRoots(x^2-2*x+1,x);
(%o89) 1
(%i91) deCasteljauNumberOfRoots(x^2-2*x+1,x);
(%o91) 1
(%i92) sRemNumberOfRoots(x^2-2*x+1,x);
(%o92) 1
(%i93) sSubResNumberOfRoots(x^2-2*x+1,x);
(%o93) 1
(%i94)
```

- Code

The code can be found in `./SARAG/rootCounting.mac` for `sSubRes` and `sRem` methods and in `./SARAG/rootIsolation.mac` for Decasteljau's method

- Use it yourself

**Cauchy Index Between.** Computes the Cauchy Index of a rational function between two real numbers

□

- Syntax

Name : `cauchyIndexBetween(Q,P,var,a,b)`



Input : Q, P polynomials  
         var is the variable of the polynomials (for example x)  
         a and b are reals ( $a < b$ ) or -INFINITY or +INFINITY

Method :

- i. sRem [signed remainder sequence]
- ii. sSubRes [signed subresultants]\*

Output : Cauchy index of Q/P in the interval ]a,b[

- Example

```
(%i94) load("./SARAG/sarag.mac");
(%o94) ./SARAG/sarag.mac

(%i95) cauchyIndexBetween( (x-5)*(x-4)*(x-2)*(x+1)*(x+2)*(x+4), (x-3)^2*(x-1)*(x+3), x, 1, -1);
(%o95) 0

(%i98) sSubResCauchyIndexBetween( (x-5)*(x-4)*(x-2)*(x+1)*(x+2)*(x+4), (x-3)^2*(x-1)*(x+3), x, 1, -1);
(%o98) 0

(%i99) sRemCauchyIndexBetween( (x-5)*(x-4)*(x-2)*(x+1)*(x+2)*(x+4), (x-3)^2*(x-1)*(x+3), x, 1, -1);
(%o99) 0

(%i100)
```

- Code

The code can be found in ./SARAG/rootCounting.mac

- Use it yourself

**Tarski Query Between.** Computes the Tarski Query of two polynomials between two real numbers



- Syntax

Name : `tarskiQueryBetween(Q,P,var,a,b)`

Input : Q, P polynomials

var is the variable of the polynomials (for example x)

a and b are reals ( $a < b$ ) or `-INFINITY` or `+INFINITY`

Method :

- i. `sRem` [signed remainder sequence]
- ii. `sSubRes` [signed subresultants]\*

Output : Tarski query of Q,P in  $]a,b[$

- Example

```
(%i100) load("./SARAG/sarag.mac");
(%o100) ./SARAG/sarag.mac
(%i103) tarskiQueryBetween(1,x^11-x^10+1,x,0,1);
(%o103) 0
(%i104) sSubResTarskiQueryBetween(1,x^11-x^10+1,x,0,1);
(%o104) 0
(%i105) sRemTarskiQueryBetween(1,x^11-x^10+1,x,0,1);
(%o105) 0
(%i106)
```

- Code

The code can be found in `./SARAG/rootCounting.mac`

- Use it yourself



**Number of Roots Between.** Computes the number of real roots of a polynomial between two real numbers

- Syntax

Name : `numberOfRootsBetween(P,var,a,b)`

Input : P polynomial

var is the variable of the polynomial (for example x)

a and b are reals ( $a < b$ ) or `-INFINITY` or `+INFINITY`

Method :

- i. `sRem` [signed remainder sequence]
- ii. `sSubRes` [signed subresultants]
- iii. `deCasteljau` [De Casteljau method for isolation]\*
- iv. `monomial` [monomial method for isolation]

Output : number of roots of P in the interval  $]a,b[$

- Example

```
(%i8) numberOfRootsBetween(x^2-2,x,0,10);
(%o8) 1
(%i9) deCasteljauNumberOfRootsBetween(x^2-2,x,0,10);
(%o9) 1
(%i10) sSubResNumberOfRootsBetween(x^2-2,x,0,10);
(%o10) 1
(%i11) sRemNumberOfRootsBetween(x^2-2,x,0,10);
(%o11) 1
(%i12)
```

- Code

The code can be found in `./SARAG/rootIsolation.mac`

- Use it yourself

**Hankel Matrix.** Computes the Hankel quadratic form associated to a sequence of numbers □

- Syntax

Name : `hankelMatrix(seq)`

Input : sequence `seq` of odd length of elements of an integral domain

Output : Hankel quadratic form for `seq` (represented by a square symmetric matrix)

- Example

```
(%i119) load("./SARAG/sarag.mac");
```

```
(%o119) ./SARAG/sarag.mac
```

```
(%i120) hankelMatrix([0,1,2,3,4]);
```

```
(%o120) [[0, 1, 2], [1, 2, 3], [2, 3, 4]]
```

```
(%i121)
```

- Code

The code can be found in `./SARAG/rootCounting.mac`

- Use it yourself

**Hankel Signature.** Computes the signature of the Hankel quadratic form associated to a sequence □

- Syntax

Name : `hankelSignature(seq)`

Input : sequence `seq` of odd length of elements of an integral domain

Output : signature of the Hankel quadratic form for `seq`

- Example

```
(%i117) load("./SARAG/sarag.mac");
```

```
(%o117) ./SARAG/sarag.mac
(%i118) hankelSignature([0,1,2,3,4]);
(%o118) 0
(%i119)
```

- Code
  - The code can be found in ./SARAG/rootCounting.mac
  - Use it yourself

**Difference of positive and negative roots.** Computes the difference between the number of roots a polynomial with positive real part and those with negative real part □

- Syntax
  - Name : posNegDiff(P,var)
  - Input : P is a polynomial in variable var
  - Output : difference between the number of roots of P with positive real part and those with negative real part
- Example

```
(%i114) load("./SARAG/sarag.mac");
(%o114) ./SARAG/sarag.mac
(%i115) posNegDiff((x^2-1)*(x^3-1),x);
(%o115) -1
(%i116)
```

- Code
  - The code can be found in ./SARAG/rootCounting.mac
  - Use it yourself

□

### 2.2.5 Isolation of roots

The corresponding results are explained in Chapter 10 of "Algorithms in Real Algebraic Geometry" by S. Basu, R. Pollack, M.-F. Roy

*file : rootIsolation.mac*

The routines contained in this files deal with the problem of isolation of real roots by using the conversion to the Bernstein basis and De Casteljaeu's method.

#### Auxiliary Functions :

**Cauchy Upper Bound.** Computes the difference between the number of roots a polynomail with positive real part and those with negative real part

- Syntax
  - Name : `cauchyRootUpperBound(P,var)`
  - Input : P is a polynomial in variable var
  - Output : the upper bound for the absolute values of all real roots of P
- Example

```
(%i7) load("./SARAG/sarag.mac");
(%o7) ./SARAG/sarag.mac
(%i8) cauchyRootUpperBound(x+2,x);
(%o8) 3
(%i9)
```

- Code
  - The code can be found in `./SARAG/rootIsolation.mac`
  - Use it yourself

□

**Cauchy Lower Bound.** Computes the lower bound for the absolute values of all real roots of a polynomial

- Syntax
  - Name : `cauchyRootLowerBound(P,var)`
  - Input : P is a polynomial in variable var
  - Output : the lower bound for the absolute values of all non-zero real roots of P
- Example

```
(%i9) load("./SARAG/sarag.mac");
(%o9) ./SARAG/sarag.mac
(%i10) cauchyRootLowerBound(x+2,x);
(%o10)  $\frac{2}{3}$ 
(%i11)
```

- Code
  - The code can be found in `./SARAG/rootIsolation.mac`
- Use it yourself

**Prime Cauchy Upper Bound.** Computes an (alternative) upper bound for the absolute values of all real roots of a polynomial □

- Syntax
  - Name : `primeCauchyRootUpperBound(P,var)`
  - Input : P is a polynomial in variable var
  - Output : (alternative) upper bound for the absolute values of all real roots of P
- Example

```
(%i3) load("./SARAG/sarag.mac");
(%o3) ./SARAG/sarag.mac
```

```
(%i45) primeCauchyRootUpperBound(x+2,x);
(%o45) 10
(%i46)
```

- Code
  - The code can be found in `./SARAG/rootIsolation.mac`
  - Use it yourself

**Prime Cauchy Lower Bound.** Computes an (alternative) lower bound for the absolute values of all real roots of a polynomial P □

- Syntax
  - Name : `primeCauchyRootLowerBound(P,var)`
  - Input : P is a polynomial in variable var
  - Output : (alternative) lower bound for the absolute values of all non-zero real roots of P
- Example

```
(%i11) load("./SARAG/sarag.mac");
(%o11) ./SARAG/sarag.mac
(%i12) primeCauchyRootLowerBound(x+2,x);
(%o12)  $\frac{2}{5}$ 
(%i13)
```

- Code
  - The code can be found in `./SARAG/rootIsolation.mac`
  - Use it yourself

**Convert to Bernstein.** Computes the coefficients of a polynomial in the Bernstein's Basis □



- Syntax

Name : `convert2Bernstein(P,d,var,l,r)`

Input : `P` is a polynomial in variable `var` of degree at most `d`.

`l` and `r` are the parameters of the Basis. ( $l < r$ )

Output : list containing the coefficients of `P` in the Bernstein basis of degree `d` for `l,r`

- Example

```
(%i13) load("./SARAG/sarag.mac");
```

```
(%o13) ./SARAG/sarag.mac
```

```
(%i14) convert2Bernstein(x^2,2,x,0,1);
```

```
(%o14) [0, 0, 1]
```

```
(%i15)
```

- Code

The code can be found in `./SARAG/rootIsolation.mac`

- Use it yourself

**Berntein Split.** Computes the coefficients of `P` in the Bernstein's Basis for `l,m` and `mr`, knowing those for `l,r`

□

- Syntax

Name : `bernsteinSplit(coeffList, l,r,m)`

Input : `coeffList` is the list of coefficients of a polynomial in Bernstein basis for `l,r`, `m` is another number

Output : `[bern_lm, bern_mr]` where

1. `bern_lm` is a list containing the coefficients in the Bernstein basis of degree `d` for `l,m`

2. `bern_mr` is a list containing the coefficients in the Bernstein basis of degree `d` for `m,r`

- Example

```
(%i19) load("./SARAG/sarag.mac");
```

```
(%o19) ./SARAG/sarag.mac
(%i20) bernsteinSplit([4,-6,7,10],0,1/2,1);
(%o20) [[4,-16,56,-44],[-44,6,13,10]]
(%i21)
```

- Code
  - The code can be found in `./SARAG/rootIsolation.mac`
  - Use it yourself

**Special Berntein Split.** Computes integer multiples of the coefficients of  $P$  in the Berstein's Basis for  $l, (l+r)/2$  and  $(l+r)/2, r$  knowing those for  $l, r$  □

- Syntax
  - Name : `specialBernsteinSplit(coeffList, l, r)`
  - Input : `coeffList` is the list of coefficients of a polynomial in berstein basis for parameters  $l, r$
  - Output : `[bern_first, bern_second]` where :
    1. `bern_first` is a list containing the coefficients in the Bernstein basis for  $l, (l+r)/2$  of  $2^{\deg(P)} P$
    2. `bern_second` is a list containing the coefficients in the Bernstein basis for  $(l+r)/2, r$  of  $2^{\deg(P)} P$
- Example

```
(%i7) load("./SARAG/sarag.mac");
(%o7) ./SARAG/sarag.mac
(%i51) specialBernsteinSplit([4,-6,7,10],0,1);
(%o51) [[32,-8,-2,17],[17,36,68,80]]
(%i52)
```

- Code
  - The code can be found in `./SARAG/rootIsolation.mac`
- Use it yourself

### Main Functions :



**Isolate Roots.** Computes a list containing points or open intervals which contain the roots of a given polynomial

- Syntax
  - Name : `isolateRoots(P,var)`
  - Input :  $P$  is a polynomial in variable  $var$ .
  - Method :
    - deCasteljau [De Casteljau method for root isolation]
    - `monomial` [root isolation in the monomial basis]

Output : characterization of the roots of  $P$  by a list of elements of the form either

1.  $[pt]$  describing a real root  $pt$  of  $P$
2.  $[a,b]$  describing the open interval  $]a,b[$  containing exactly one root of  $P$

- Example

```
(%i4) load("./SARAG/sarag.mac");
(%o4) ./SARAG/sarag.mac
(%i5) isolateRoots(-33*x^3+69*x^2-30*x+4,x);
(%o5) [[1, 2]]
(%i6)
```

- Code
  - The code can be found in `./SARAG/rootIsolation.mac`

- Use it yourself

**Isolate Roots Between.** Computes a list containing points or open intervals which contain the roots of a given polynomial between two real numbers □

- Syntax

Name : `isolateRootsBetween(P,var,a,b)`

Input : P is a polynomial in variable var.

a,b two real numbers

Method :

- i. deCasteljau [De Casteljau method for root isolation]
- ii. `monomial` [root isolation in the monomial basis]

Output : list of elements of the form either

1. [pt] describing the real root pt which is a root of P
2. [c,d] describing the open interval "[c,d]" containing exactly a root of P

- Example

```
(%i16) load("./SARAG/sarag.mac");
(%o16) ./SARAG/sarag.mac
(%i17) isolateRootsBetween((x-2)^2-1,x,-4,4);
(%o17) [[0, 2], [2, 4]]
(%i18)
(%i18)
```

- Code

The code can be found in `./SARAG/rootIsolation.mac`

- Use it yourself

**Find Roots.** Computes a list containing points of open intervals smaller than threshold which contain exactly one root of a given polynomial □

- Syntax

Name : `findRoots(P,var,threshold)`

Input : `P` is a polynomial in variable `var`.

`threshold` is a number

Method :

- i. deCasteljau [De Casteljau method for root isolation]
- ii. `monomial` [root isolation in the monomial basis]

Output : list of elements of the form either

1. `[pt]` describing the real root `pt` of `P`
2. `[a,b]` describing the open interval "`a,b`" with `b-a` smaller than threshold containing exactly one root of `P`

- Example

```
(%i25) load("./SARAG/sarag.mac");
```

```
(%o25) ./SARAG/sarag.mac
```

```
(%i26) findRoots(x+2,x,1);
```

```
(%o26) [[ -2]]
```

```
(%i27)
```

- Code

The code can be found in `./SARAG/rootIsolation.mac`

- Use it yourself

**Find Roots Between.** Computes a list containing points of open intervals smaller than `treshold` which contain exactly one root of a given polynomial and are between two real numbers □

- Syntax

Name : `findRoots(P,var,treshold)`

Input : `P` is a polynomial in variable `var`.

`treshold` is the interval in which the function search the roots.

Method :

i. `deCasteljau` [De Casteljau method for root isolation]

ii. `monomial` [root isolation in the monomial basis]

Output : list of elements of the form either

1. `[pt]` describing the real root `pt` of `P`

2. `[a,b]` describing the open interval "`]a,b[`" smaller than `treshold` and containing exactly a root of `P`

- Example

```
(%i27) load("./SARAG/sarag.mac");
```

```
(%o27) ./SARAG/sarag.mac
```

```
(%i28) findRootsBetween(x^3-1,x,1,-2,2);
```

```
(%o28) [[1]]
```

```
(%i29)
```

- Code

The code can be found in `./SARAG/rootIsolation.mac`

- Use it yourself

□

**Roots Sign.** Computes a list containing couples describing the real roots of P and the sign of Q at these real roots

- Syntax

Name : `signsAtRoots(isInt,P,Q,var)`

Input : P,Q are polynomials in variable var.

isolating list isInt for the real roots of P in the same form as in the output of "isolateRealRoots"

Method :

- deCasteljau [De Casteljau method for root isolation]
- `monomial` [root isolation in the monomial basis]

Output : lists of couples describing a real root and the sign of Q at this real root

- Example

```
(%i29) load("./SARAG/sarag.mac");
(%o29) ./SARAG/sarag.mac
(%i30) polR:(x-2)*(x^2-1);
(%o30) (x - 2) (x^2 - 1)
(%i31) isInt : deCasteljauIsolateRoots(polR,x);
(%o31) [[ - 8, 0], [0, 2], [2]]
(%i32) polQ:x^2-4;
(%o32) x^2 - 4
(%i37) signsAtRoots(isInt,polR,polQ,x);
(%o37) [[[ - 2, 0], - 1], [[0, 2], - 1], [[2], 0]]
(%i36) deCasteljauSignsAtRoots(isInt,polR,polQ,x);
(%o36) [[[ - 2, 0], - 1], [[0, 2], - 1], [[2], 0]]
```

(%i37)

- Code
  - The code can be found in `./SARAG/rootIsolation.mac`
  - Use it yourself

**Evaluate signs.** Evaluates the signs of two polynomials at their roots

□

- Syntax
  - Name : `evaluateSignsAtRoots(P,Q,var)`
  - Input : P,Q are polynomials in variable var.

Method :

- i. `deCasteljau` [De Casteljau method for root isolation]
- ii. `monomial` [root isolation in the monomial basis]

Output : `[com,signNComP,signNComQ]` where

1. `com` is an isolating list for the common real roots of P and Q
2. `signNComP` is an isolating list with signs of Q for the real roots of P that are not roots of Q
3. `signNComQ` is an isolating list with signs of P for the real roots of Q that are not roots of P

- Example

```
(%i48) load("./SARAG/sarag.mac");
(%o48) ./SARAG/sarag.mac
(%i49) evaluateSignsAtRoots(x+1,x,x);
(%o49) [[], [[[ - 2, 0], - 1]], [[[ - 1, 1], 1]]]
(%i50) deCasteljauEvaluateSignsAtRoots(x+1,x,x);
```



```
(%o50) [[], [[[ - 2, 0], - 1]], [[[ - 1, 1], 1]]]
```

```
(%i51)
```

- Code

The code can be found in `./SARAG/rootIsolation.mac`

- Use it yourself

### 2.2.6 Sign Determination □

The corresponding results are explained in Chapter 10 of "Algorithms in Real Algebraic Geometry" by S. Basu, R. Pollack, M.-F. Roy

*file : signDetermination.mac*

NOTE: When we refer to an algorithm that computes the Tarski query of polynomials we assume that it has the same input/output format as "tarskiQuery" (see Section Root Counting)

**Sign Determination.** Computes the signs of list of polynomials in a given finite set (typically defined as the zero set of a polynomial)

- Syntax

Name : `signDetermination(polList,ptSet,sqAlg,var)`

Input : list `polList` of polynomials in `var`, a description of a finite set of points, an algorithm `sqAlg` to compute the Tarski query of polynomials in `polList` with respect to `ptSet`

Method :

- naive [brute force method on a huge matrix of signs]
- smart [a method that uses much smaller matrices of signs]\*

Output : a list representing a subset of the all elements of  $\{0,-1,1\}^{\text{polList}}$  describing the possible signs of the polynomials in `polList` at `ptSet`

- Example

```

(%i134) load("./SARAG/sarag.mac");
(%o134) ./SARAG/sarag.mac

(%i51) signDetermination( [x,x+1,x-2], x^5-9*x^3-x^2+9, tarskiQuery,x);
(%o51) [[1, 1, 1], [1, 1, - 1], [- 1, - 1, - 1]]

(%i52) smartSignDetermination( [x,x+1,x-2], x^5-9*x^3-x^2+9, tarskiQuery,x);
(%o52) [[1, 1, 1], [1, 1, - 1], [- 1, - 1, - 1]]

(%i53) naiveSignDetermination( [x,x+1,x-2], x^5-9*x^3-x^2+9, tarskiQuery,x);
(%o53) [[- 1, - 1, - 1], [1, 1, - 1], [1, 1, 1]]

(%i54)

```

- Code
  - The code can be found in `./SARAG/signDetermination.mac`
  - Use it yourself

□

### 2.2.7 Thom encoding

The corresponding results are explained in Chapter 10 of "Algorithms in Real Algebraic Geometry" by S. Basu, R. Pollack, M.-F. Roy

*file : signDetermination.mac*

**Thom Encoding.** Computes the Thom encodings of the real roots of a polynomial, which is the list of signs taken by the derivatives

- Syntax
  - Name : `thomEncoding(P,var)`
  - Input : polynomial  $P$  in variable  $var$
  - Output : a list containing the Thom encoding of the real roots of  $P$

- Example

```
(%i54) load("./SARAG/sarag.mac");
(%o54) ./SARAG/sarag.mac
(%i56) thomEncoding(x^2-2,x);
(%o56) [[-1,1],[1,1]]
(%i57)
```

- Code

The code can be found in `./SARAG/signDetermination.mac`

- Use it yourself

**Thom Compare.** Compares the Thom encodings of the real roots of two polynomials

□

- Syntax

Name : `thomCompare(P,Q,var)`

Input : polynomials `P,Q` in variable `var`

Output : a list containing elements of the form `[owner,thomInf]` with the following possibilities :

1. `[0,thomP,thomQ]` for a common root of `P` with Thom encoding `thomP` and of `Q` with Thom encoding `thomQ`
2. `[1,thomP]` for a root of `P` not of `Q` with Thom encoding `thomP`
3. `[2,thomQ]` for a root of `Q` not of `P` with Thom encoding `thomQ`

Moreover the corresponding real numbers are ordered.

- Example

```
(%i55) load("./SARAG/sarag.mac");
(%o55) ./SARAG/sarag.mac
(%i133) thomCompare( x^2-4, expand((x-2)*(x^3-1)),x);
```

```
(%o133) [[1, [-1, 1]], [2, [-1, 0, 1, 1]], [0, [1, 1], [1, 1, 1, 1]]]
```

```
(%i134)
```

- Code
  - The code can be found in `./SARAG/signDetermination.mac`
  - Use it yourself

**Thom Encoding with signs.** Computes the Thom encoding of  $P$  at each root and the sign of  $Q$  for each root □

- Syntax
  - Name : `thomEncodingWithSigns(P,Q,var)`
  - Input : polynomials  $P, Q$  in variable  $var$
  - Output : a list of couples containing for each root of  $P$  its Thom encodings and the corresponding sign of  $Q$
- Example

```
(%i57) load("./SARAG/sarag.mac");
```

```
(%o57) ./SARAG/sarag.mac
```

```
(%i58) thomEncodingWithSigns(x^2-2,x-1,x);
```

```
(%o58) [[[1, 1], 1], [[-1, 1], -1]]
```

```
(%i59)
```

- Code
  - The code can be found in `./SARAG/signDetermination.mac`
  - Use it yourself

## 2.3 Topology □

The corresponding results are explained in Chapter 11 of "Algorithms in Real Algebraic Geometry" by S. Basu, R. Pollack, M.-F. Roy

**Topology:.** Computes the topology of a plane curve

- Syntax

Name: `topology(curve,x,y)`.

Input: the equation of a curve in two variables with rational coefficients and the name of the two variables.

Output: a couple whose first element gives the value of  $a$  in the initial change of coordinates, the second describes the topology with respect to the new coordinates. It gives for each interval between the roots of the discriminant the number of branches above it and for each root  $x$  of the discriminant, a couple containing the number of points of the curve above  $x$  and the position of the unique critical point over  $x$ .

More precisely, the output is  $[a, [m_0, [n_1, c_1], \dots, m_{r-1}, [n_r, c_r], m_r]]$ .

Name: `drawTopology(T[2],x,y)`.

Input: the output `T` to `topology(curve,x,y)`.

Output: a piecewise linear curve homeomorphic to the zero set of the polynomial curve.

- Example

```
(%i1) load("./SARAG/sarag.mac");
(%o1) ./SARAG/sarag.mac
(%i2) examplecurve:2*y^3+(3*x-3)*y^2+(3*x^2-3*x)*y+x^3;
(%o2) 2 y^3 + (3 x - 3) y^2 + (3 x^2 - 3 x) y + x^3
(%i3) T:topology(examplecurve,x,y);
(%o3) [0, [1, [2, 2], 3, [2, 1], 3, [2, 2], 1]]
(%i4) drawTopology(T[2]);
(%o4) true
(%i5) vide:topology(x^2+y^2+1,x,y);
(%o5) [0, [0]]
```

```

(%i6) drawTopology(vide[2]);
plot2d: nothing to plot.
(%o6) true
(%i7) notgeneric:(y^2+x)*(y^2-x+1);
(%o7)  $(y^2 - x + 1)(y^2 + x)$ 
(%i8) B:topology(notgeneric,x,y);
(%o8) [1, [2, [1, 1], 0, [1, 1], 2]]
(%i9) drawTopology(B[2]);
(%o9) true
(%i10)

```

- Code
  - The code can be found in `./SARAG/topology.mac`
  - Use it yourself

## 2.4 Certificate of positivity

□

a) In the univariate case, the corresponding reference is F. Boudaoud, F. Caruso M.-F. Roy Certificates of positivity in the Bernstein basis, Discrete and Computational Geometry 39 4 639-655 (2008)

*file : certificateOfPositivity.mac*

**Certificate of positivity.** Decides whether a polynomial is positive or negative on an interval and provides a certificate

- Syntax
  - Name : `certificate(P,var,a,b)`
  - Input : P is a polynomial in variable var. a and b are real numbers

Output :

1. [positive, certificate] if P is positive. Certificate
2. [negative, certificate] if P is negative
3. [false, [a], sqPol] if P(a) equal 0 (where sqPol is the separable part of P)
4. [false, [c,d], sqPol] if sqPol(c)\*sqPol(d)<0 (where sqPol is the separable part of P)

- Example

```
(%i10) load("./SARAG/sarag.mac");
```

```
(%o10) ./SARAG/sarag.mac
```

```
(%i11) certificate(x^4+(8*x-1)^2,x,-1,1);
```

```
(%o11) [positive, [[[-1, 0], 3, [3936, 2160, 944, 240, 48]], [[0, 1/8], 3, [196608, 98304, 32768, 0, 48]], [[1/8, 1], 3, [48, 384, 1608704, 4841472, 9830400]]]]]
```

```
(%i29) certificate((x-2)^2-1,x,-1,1);
```

```
(%o29) [false, [[1], x^2 - 4x + 3]]
```

```
(%i30) certificate((2*x^2-1)^2,x,-1,1);
```

```
(%o30) [false, [-1, 0], 4x^2 - 2]
```

```
(%i31) certificate(-x^4-(8*x-1)^2,x,-1,1);
```

```
(%o31) [negative, [[[-1, 0], 3, [-3936, -2160, -944, -240, -48]], [[0, 1/8], 3, [-196608, -98304, -32768, 0, -48]], [[1/8, 1], 3, [-48, -384, -1608704, -4841472, -9830400]]]]]
```

(%i32)

- Code
  - The code can be found in `./SARAG/certificateOfPositivity.mac`
  - Use it yourself

b) In the multivariate case, the corresponding reference is R. Leroy, Certificats de positivité et minimisation polynomiale dans la base de Bernstein multivariée [http://rileroy.perso.math.cnrs.fr/Page\\_personnelle/Recherche.html](http://rileroy.perso.math.cnrs.fr/Page_personnelle/Recherche.html) □  
*file : multiCertificateOfPositivity.mac*

**Bernstein coefficients on a simplex.** Various annex functions expressing a polynomial in the Bernstein basis on a simplex.

- Syntax
  - Name : `compositions(d,k+1)`  
 Input : `d` and `k` are natural numbers  
 Output : all the multiindices of sum `d` and length `k+1`
  
  - Name : `simplex2bar(V,vars)`  
 Input : a simplex `V` given by `k+1` points, each of them described by its `k` coordinates  
 Output : the `k+1` barycentric coordinates defining the simplex
  
  - Name : `monom2bern(P,V,vars,d)`  
 Input : `P` is a multivariate polynomial in a list of `k` variable `vars`, `d` is at least the degree of `P`, `V` is a simplex described by `k+1` points with `k` coordinate  
 Output : the list of coefficients of `P` in the Bernstein basis of degree `d` on `V` (in the order corresponding to compositions `d,k` for the barycentric coordinates given by `simplex2bar`)
  
  - Name : `deCasteljau(simpl,ber,deg,M)`  
 Input : a simplex defined by `k+1` points with `k` coordinates, Bernstein coefficients on this simplex of a polynomial of degree at most `deg` and the `k` coordinates of a point



Output : the list of coefficients of P in the Bernstein basis of degree d on the k+1 subsimplices defined by the initial simplex and the point

- Example

```
(%i30) load("./SARAG/sarag.mac");
(%o30) ./SARAG/sarag.mac
(%i31) Comp:compositions(2,3);
(%o31) [[2,0,0],[1,1,0],[1,0,1],[0,2,0],[0,1,1],[0,0,2]]
(%i32) L:simplex2bar([[0,0],[1,0],[0,1]],[x,y]);
(%o32) [-y-x+1,x,y]
(%i33) expand(sum(apply(multinomial_coeff,Comp[i])*prod(L[j]^Comp[i][j],j,1,length(L)),i,1,length(Comp))):monom2bern
(%o33) 1
(%i35) B:monom2bern(9*y^2-24*x*y+12*y+16*x^2+16*x+5,[[0,0],[1,0],[0,1]],[x,y],2);
(%o35) [5,13,11,37,7,26]
(%i36) expand(sum(B[i]*apply(multinomial_coeff,Comp[i])*prod(L[j]^Comp[i][j],j,1,length(L)),i,1,length(Comp))):monom2bern
;
(%o36) 9 y^2 - 24 x y + 12 y + 16 x^2 + 16 x + 5
(%i38) deCasteljau([[0,0],[1,0],[0,1]],B,2,[1/2,1/2]);
(%o38) [[[[[1/2, 1/2], [1,0],[0,1]], [77/4, 22, 33/2, 37, 7, 26]], [[0,0], [1/2, 1/2], [0,1]], [5, 12, 11, 77/4, 33/2, 26]], [[0,0], [1,0], [1/2, 1/2]],
[5, 13, 12, 37, 22, 77/4]]]]
```

(%i39)

- Code

The code can be found in `./SARAG/multiCertificateOfPositivity.mac`

- Use it yourself

**Certificate of positivity in the multivariate case.** Decides whether a polynomial is positive on a simplex and provides a certificate □

- Syntax

Name : `multiCertificate(P,V,vars,d,sub,cert)`

Input : P is a multivariate polynomial in a list of k variable vars, d is at least the degree of P, V is a simplex described by k+1 points with k coordinates, sub is a subdivision method (bisection, bisection\_castel, standard\_triang2, bisection\_max\_pos)

Output :

1. [certificate] if P is positive i.e. a sequence of subsimplexes and Bernstein coefficients on the corresponding subsimplex
2. [a, sign(P(a))] if  $P(a) \leq 0$
3. in other cases the output is not satisfactory yet

Name : `drawMultiCertificate(list)`

Input : the certificate output by `multiCertificate`, when the polynomial is positive, in the bivariate case

Output : a picture of the corresponding subdivision

- Example

(%i39) `load("./SARAG/sarag.mac");`

(%o39) `./SARAG/sarag.mac`

(%i40) `f:9*y^2-24*x*y+12*y+16*x^2-16*x+5;`

(%o40)  $9y^2 - 24xy + 12y + 16x^2 - 16x + 5$

```
(%i41) C1:multiCertificate(f,[[0,0],[1,0],[0,1]],[x,y],2,bisection,pos);
```

```
(%o41) [[[[[0,0],[1/2,1/2],[0,1]],[5,4,11,13/4,17/2,26]],[[[[0,0],[1/2,0],[1/2,1/2]],[5,1,4,1,1,13/4]],[[[[1/2,0],[3/4,1/4],[1/2,1/2]],[1,1,1,17/16,5/8,13/4]],[[[[1/2,0],[1,0],[3/4,1/4]],[1,1,1,5,3/2,17/16]]]]]]
```

```
(%i42) drawMultiCertificate(C1,500);
```

```
(%o42) [gr2d(polygon,polygon,polygon,polygon)]
```

```
(%i43) C2:multiCertificate(f,[[0,0],[1,0],[0,1]],[x,y],2,bisection_max_pos,pos);
```

```
(%o43) [[[[[0,0],[7/10,3/10],[0,1]],[5,6/5,11,101/100,3/2,26]],[[[[2/5,0],[1,0],[7/10,3/10]],[29/25,1/5,26/25,5,4/5,101/100]],[[[[0,0],[2/5,0],[7/10,3/10]],[5,9/5,6/5,29/25,26/25,101/100]]]]]]
```

```
(%i44) drawMultiCertificate(C2,500);
```

```
(%o44) [gr2d(polygon,polygon,polygon)]
```

```
(%i45) C3:multiCertificate(f,[[0,0],[1,0],[0,1]],[x,y],2,standard_triang2,pos);
```

```
(%o45) [[[[[0,1/2],[1/2,1/2],[0,1]],[53/4,25/4,37/2,13/4,17/2,26]],[[[[1/2,0],[0,1/2],[1/2,1/2]],[1,1,1,53/4,25/4,13/4]],[[[[0,0],[1/2,0],[0,1/2]],[5,1,8,1,1,53/4]],[[[[1/2,1/4],[3/4,1/4],[1/2,1/2]],[25/16,13/16,17/8,17/16,5/8,13/4]],[[[[3/4,0],[1,0],[3/4,1/4]],[2,3,5/4,5,3/2,17/16]],[[[[3/4,0],[1/2,1/4],[3/4,1/4]],[2,1/4,5/4,25/16,13/16,17/16]],[[[[1/2,0],[3/4,0],[1/2,1/4]],[1,1,1,2,1/4,25/16]]]]]]]]
```

```
(%i46) drawMultiCertificate(C3,500);
```

```
(%o46) [gr2d(polygon,polygon,polygon,polygon,polygon,polygon,polygon)]
```

```
(%i47) multiCertificate(x^2-y^2-1,[[0,0],[1,0],[0,1]],[x,y],2,bisection,pos);
```

```
(%o47) [[0,0], -1]
```

```
(%i48) g:9*y^2-24*x*y+12*y+16*x^2-16*x+5+z^2;
```

```
(%o48)  $z^2 + 9y^2 - 24xy + 12y + 16x^2 - 16x + 5$ 
```

```
(%i49) M1:multiCertificate(g,[[0,0,0],[1,0,0],[0,1,0],[0,0,1]],[x,y,z],2,standard_triang2,pos);
```

```
(%o49) [[[[[0,0,1/2],[1/2,0,1/2],[0,1/2,1/2],[0,0,1],[21/4,5/4,33/4,11/2,5/4,5/4,3/2,27/2,17/2,6]],[[[0,1/2,0],[1/2,1/2,0],[0,1,0],[0,1/2,1/2],[53/4,25/4,37/2,53/4,13/4,17/2,25/4,26,37/2,27/2]],[[[0,1/2,0],[1/2,1/2,0],[1/2,0,1/2],[0,1/2,1/2],[29/4,9/4,6,29/4,5/4,2,9/4,21/4,25/4,15/2]],[[[0,1/2,0],[0,0,1/2],[1/2,0,1/2],[0,1/2,1/2],[29/4,6,6,29/4,21/4,21/4,25/4,21/4,25/4,15/2]],[[[1/2,0,0],[0,1/2,0],[1/2,1/2,0],[1/2,0,1/2]],[[[0,1/2,0],[0,0,1/2],[1/2,0,1/2],[0,1/2,1/2]],[[[1,1,1,1,53/4,25/4,1,13/4,1,5/4]],[[[1/2,0,0],[0,1/2,0],[0,0,1/2],[1/2,0,1/2]],[[[1,1,1,1,85/4,1,1,5/4,5/4,5/4]],[[[0,0,0],[1/2,0,0],[0,1/2,0],[0,0,1/2]],[[[5,1,8,5,1,1,1,53/4,8,21/4]],[[[1/2,0,1/4],[3/4,0,1/4],[1/2,1/4,1/4],[1/2,0,1/2]],[[[17/16,17/16,17/16,9/8,33/16,5/16,9/8,13/8,9/8,5/4]],[[[1/2,1/4,0],[3/4,1/4,0],[1/2,1/2,0],[1/2,1/4,1/4]],[[[25/16,13/16,17/8,25/16,17/16,5/8,13/16,13/4,17/8,13/8]],[[[1/2,1/4,0],[3/4,1/4,0],[3/4,0,1/4],[1/2,1/4,1/4]],[[[5/4,3/4,1,5/4,5/4,1,3/4,17/16,17/16,21/16]],[[[3/4,0,0],[1,0,0],[3/4,1/4,0],[3/4,0,1/4]],[[[2,3,5/4,2,5,3/2,3,17/16,5/4,33/16]],[[[1/2,1/4,0],[1/2,0,1/4],[3/4,0,1/4],[1/2,1/4,1/4]],[[[5/4,1,1,5/4,17/16,17/16,17/16,17/16,21/16]],[[[3/4,0,0],[1/2,1/4,0],[3/4,1/4,0],[3/4,0,1/4]],[[[2,1/4,5/4,2,25/16,13/16,1/4,17/16,5/4,33/16]],[[[3/4,0,0],[1/2,1/4,0],[1/2,0,1/4],[3/4,0,1/4]],[[[2,0,2,2,2,0,0,33/16,33/16,33/16]],[[[1/2,0,0],[3/4,0,0],[1/2,1/4,0],[1/2,0,1/4]],[[[1,1,1,1,2,1/4,1,25/16,1,17/16]]]]]]]
```

```
(%i50) M2:multiCertificate(g,[[0,0,0],[1,0,0],[0,1,0],[0,0,1]], [x,y,z],2,bisection_max_pos,pos);
```

```
(%o50) [[[[[0,0,0],[7/10,3/10,0],[0,1,0],[0,0,1]], [5,6/5,11,5,101/100,3/2,6/5,26,11,6]], [[ [0,0,0],[3/5,0,2/5],[7/10,3/10,0],[0,0,1],
[5,1/5,6/5,5,33/25,24/25,3/5,101/100,6/5,6]], [[ [2/5,0,0],[1,0,0],[7/10,3/10,0],[3/5,0,2/5]], [29/25,1/5,26/25,21/25,5,4/5,9/5,101/100,24/25,33/25]], [[ [0,0,0],
[2/5,0,0],[7/10,3/10,0],[3/5,0,2/5]], [5,9/5,6/5,1/5,29/25,26/25,21/100,24/25,33/25]]]]]
```

```
(%i51)
```

- Code

The code can be found in `./SARAG/multiCertificateOfPositivity.mac`

- Use it yourself

□