UNIVERSITÄT KAISERSLAUTERN
Zentrum für Computeralgebra

REPORTS ON COMPUTER ALGEBRA
NO. 34

Libraries for Counting Real Roots

by

E.A. Tobis

July, 2005

The Zentrum für Computeralgebra (Centre for Computer Algebra) at the University of Kaiserslautern was founded in June 1993 by the Ministerium für Wissenschaft und Weiterbildung in Rheinland-Pfalz (Ministry of Science and Education of the state of Rheinland-Pfalz). The centre is a scientific institution of the departments of **Mathematics, Computer Science, and Electrical Engineering** at the University of Kaiserslautern.

The goals of the centre are to advance and to support the use of Computer Algebra in industry, research, and teaching. More concrete goals of the centre include

- the development, integration, and use of software for Computer Algebra
- the development of curricula in Computer Algebra under special consideration of interdisciplinary aspects
- the realisation of seminars about Computer Algebra
- the cooperation with other centres and institutions which have similar goals

The present coordinator of the Reports on Computer Algebra is:
Christoph Lossen (email: `lossen@mathematik.uni-kl.de`)

Zentrum für Computeralgebra
c/o Prof. Dr. G.-M. Greuel, FB Mathematik
Erwin-Schrödinger-Strasse
D-67663 Kaiserslautern; Germany
Phone: 49 - 631/205-2850 Fax: 49 - 631/205-5052
email: `greuel@mathematik.uni-kl.de`

Libraries for Counting Real Roots

Enrique A. Tobis*
Departamento de Matemática
FCEyN–UBA
etobis@dc.uba.ar

July 1, 2005

This report describes three libraries I have developed for SINGULAR [GPS05] that perform some calculations related to real solutions. Two of them count real solutions: one for univariate polynomials (`rootsur.lib`), and the other for systems of multivariate polynomials (`rootsmr.lib`). The remaining library (`signcond.lib`) determines which sign conditions can be realized by a set of multivariate polynomials on the variety of a certain ideal.

If you are using SINGULAR version 3.0.0, or higher, the libraries have already been installed in your system. If you have an older version, then in order to use them, these libraries have to be copied to your SINGULAR `LIB` directory (usually `/usr/local/Singular/<VERSION NUMBER>/LIB/`). Once they are installed, you can load them by typing[1]

```
> LIB "rootsur.lib";
// ** loaded /user/etobis/home/Singular/3-0-0/LIB/rootsur.lib
>
```

(the output may vary, depending on your SINGULAR installation)

The libraries contain information on what they do and how to use them. To obtain a list of the procedures available in a specific library, at the SINGULAR prompt, type, for instance,[1]

```
> help rootsur.lib;
```

This provides a summary of the behaviour of each function. Every library also includes information about each of its procedures. To access it, you must type, for instance,

```
> help varsigns;
```

*Supported by ANPCyT Grant PICT-99 03-06568 and Secyt Grant PAV 03-120, Subproject 03

[1]The actual ouput depends on your SINGULAR installation.

This provides a more detailed description of the procedure, along with the specific type of each parameter.

In order to work with polynomials in SINGULAR, one must define the ring of coefficients and the variables one wants to use. For example, to work with polynomials with rational coefficients in the variable `x`, one must define it by

```
> ring r = 0,x,lp;
```

This defines `r` as a the ring $\mathbb{Q}[\mathtt{x}]$. It also sets `r` as the basering. That is, any objects we define now (polynomials, ideals, etc.) will "live" inside `r`. Multiple rings may coexist during a single SINGULAR session. To switch between them, one must use the command `setring`

```
> ring r = 0,x,lp;
> poly p = 2x^2-5;
> p;
4x2-5
> ring s = 0,(x,y),dp;
> p;
   ? `p` is undefined
> poly p = 4xy-2y^2;
> p;
4xy-4y2
> setring(r);
> p;
4x2-5
```

We see in this example that a ring with several variables is declared similarly to a ring with one variable. The only difference is that we have to include a list of variables, surrounded by parentheses. We also need to choose a monomial order. Actually, we did specify a monomial order for `r`: `lp` is the lexicographic order. It did not matter then, since all orders are equivalent for polynomials in one variable. The order most frequently used is `dp`, which is the degree reverse lexicographic order. To find out more about monomial orders, check your SINGULAR documentation.

All the functions in the libraries can only work with non-parametric rings. I plan to upgrade this in a future version.

The following sections contain the mathematical background for these libraries.

1 Univariate polynomials

The library `rootsur.lib` contains routines for bounding, and counting, the number of real roots of a univariate polynomial.

`boundposDes`

INPUTS: A univariate polynomial P.
OUTPUT: An upper bound on the number of positive real roots of P, counted with multiplicities.

This routine computes an upper bound on the number of positive real roots of a polynomial. To obtain this bound, let V be the function that returns the number of sign changes in a list of numbers (e.g.: $V(-1,2,1,-4)=2$). If any of the numbers is 0, it is ignored for this calculation. Let our polynomial be $P=\sum_{i=0}^{n} a_n X^n$. Then the bound is $V(a_0,\ldots,a_n)$. That is, the number of sign changes in the list of coefficients of P. This bound is congruent to the actual number of positive roots modulo 2 (see [BR90] p. 14–15). When all the roots of a polynomial are real, this bound is the actual number of positive roots, counted with multiplicities. Since the negative roots of P are the positive roots of $P(-X)$, and it is trivial to know whether 0 is a root of a polynomial, we can also get a bound on all the roots of a polynomial. The corresponding command is `boundDes`.

```
> boundposDes((x+1)*(x-2)*(x-1000));
2
> boundposDes((x+1)*(x-2)*(x-1000)^2); // 1000 is a double root
3
> boundposDes((x+1)*(x-2)*(x-1000)^2*(x^2+1)); // Some complex roots
3
> boundposDes((x+1)*(x-2)*(x-1000)^2*(x^2+1)^2); // Some more complex roots
5
> boundDes((x+1)*(x-2)*(x-1000)^2); // Bound on all the real roots
4
```

---

boundBuFou

INPUTS: A univariate polynomial P and two numbers, $a<b$.
OUTPUT: An upper bound on the number of real roots of P lying in (a,b], counted with multiplicities.

The bound is computed as suggested by the Budan-Fourier Theorem. Let n be the degree of P, and $\mathrm{Der}(P)$ be $P',\ldots,P^{(n)}$, the list of the n nonzero derivatives of P. We shall write $\mathrm{Der}(P)(a)$ for the list $P'(a),\ldots,P^{(n)}(a)$, that is, the polynomials of $\mathrm{Der}(P)$ evaluated at a. Then this function returns $V(\mathrm{Der}(P)(a))-V(\mathrm{Der}(P)(b))$, where V is the sign changes function from the previous explanation. This bound is congruent to the number of real roots (counted with multiplicities) modulo 2 (see [BPR03] p. 38 for a proof).

```
> boundBuFou((x-1)*(x+1),-1,1);
1
> boundBuFou((x-1)*(x+1),-2,1); // The interval analyzed is (a,b]
2
> boundBuFou((x-1)*(x+1)^2,-2,1);
```

```
3
> boundBuFou((x-1)^3*(x+1)^2,-2,1); // The bound takes into account multiplicity
5
```

---

maxabs

INPUTS: A univariate polynomial P.
OUTPUT: An upper bound on the absolute value of the real roots of P.

The bound on the absolute value of the roots of P returned by this function is $1 + \max\{|a_n|, \dots, |a_0|\}$, where the a_i are the coefficients of the polynomial. (see [CLO98] p. 34).

```
> maxabs(x^2+10x-14);
15
```

---

sturmseq

INPUTS: A univariate polynomial P.
OUTPUT: A Sturm sequence for P.

The Sturm sequence we build is $\mathrm{St}_1 = P$, $\mathrm{St}_2 = P'$, $\mathrm{St}_i = -\mathrm{Rem}(\mathrm{St}_{i-2}, \mathrm{St}_{i-1})$, where $\mathrm{Rem}(P_1, P_2)$ is the remainder in the division of P_1 by P_2. The calculation is repeated until the remainder is 0. This procedure yields a Sturm sequence (see [BR90] p. 8–14). After this calculation, we divide every term of the sequence by the gcd of St_1 and St_2.

```
> sturmseq((x-3)*(x+6)^2*(x2+1));
[1]:
   x4+3x3-17x2+3x-18
[2]:
   x3+6/5x2-33/5x
[3]:
   x2-186/157x+225/157
[4]:
   x+2808/4279
[5]:
   -1
```

---

sturm

INPUTS: A univariate polynomial P and two numbers, $a < b$. $P(a)P(b) \neq 0$.
OUTPUT: The number of real roots of P lying in $[a, b]$, counted without multiplicities.

To calculate the number of roots, we first compute `sturmseq`(P). We evaluate the Sturm sequence at a and b, and the number we seek is $V($`sturmseq`$(P)(a))-$

$V(\texttt{sturmseq}(P)(b))$, where V is again the number of sign changes in a list, and $\texttt{sturmseq}(P)(x)$ is the list of numbers obtained by evaluating the polynomials of $\texttt{sturmseq}(P)$ at x (see [BR90] p. 8–14 for a proof).

```
> poly p = (x-5)*(x+28)*(x-15)*(x-6);
> sturm(p,-29,0);
1
> sturm(p,-29,16);
4
> p = p*(x-5);
> sturm(p,-29,16);
4
```

`sturmhaseq`

INPUTS: A univariate polynomial P.
OUTPUT: A Sturm-Habicht sequence for P.

The structure of the sequence is described in detail in [BPR03] (it is called signed subresultant sequence). It is somewhat similar to the Sturm sequence, but its coefficients have a smaller bitsize.

```
> sturmhaseq((x-3)*(x+6)^2*(x2+1));
[1]:
   -88711200x-532267200
[2]:
   102696x2+683568x+404352
[3]:
   314x3+1512x2-1782x+2700
[4]:
   5x4+36x3+3x2-198x
[5]:
   x5+9x4+x3-99x2-108
```

`sturmha`

INPUTS: A univariate polynomial P, and two numbers $a < b$. $P(a)P(b) \neq 0$.
OUTPUT: The number of real roots of P lying in $[a, b]$, counted without multiplicities.

This procedure returns the number of real roots of a polynomial (without multiplicities) using a Sturm-Habicht sequence for that polynomial and a modified function to count sign changes (see [BPR03] p. 279).

```
> poly p = (x-5)*(x+28)*(x-15)*(x-6);
> sturmha(p,-29,0);
1
```

```
> sturmha(p,-29,16);
4
> p = p*(x-5);
> sturmha(p,-29,16);
4
```

nrroots

INPUTS: A univariate polynomial P.
OUTPUT: The number of real roots of P, counted without multiplicities.

This functions returns `sturmha(P,-maxabs(P),maxabs(P))`.

```
> nrroots((x+1)*(x-2)*(x-1000)^2);
3
```

2 Systems of Multivariate Polynomial Equations

symsignature

INPUTS: A symmetric matrix M.
OUTPUT: The signature of M.

The signature of a matrix is equal to the number of positive eigenvalues minus the number of negative eigenvalues. The eigenvalues must be counted with multiplicities. If a matrix is symmetric and all its coefficientes are real, all its eigenvalues are real. In the description of `boundposDes`, we noted that when all the roots of a polynomial are real, the bound given by that function is equal to the actual number of positive roots. Taking advantage of this, we can compute the signature of a symmetric real matrix M: we simply apply `boundposDes` to $\chi_M(x)$ and $\chi_M(-x)$, where $\chi_M(x)$ is the characteristic polynomial of M.

```
> matrix m[3][3];
> m[1,1] = 3;m[1,2] = 1;m[2,1] = 1;m[2,2] = 3;m[3,3]= - 4;
> print(m);
3,1,0,
1,3,0,
0,0,-4
> symsignature(m);
1
```

In this case, it is easy to verify the result:

```
> eigenvals(m)[1];
_[1]=-4
_[2]=2
_[3]=4
```

sturmquery

INPUTS: A polynomial P, a Gröbner basis for a zero-dimensional ideal I, and a monomial basis B of the quotient R/I, where R is the basering.
OUTPUT: The Sturm Query of P on $V(I)$.

The Sturm Query of a polynomial P on a set Z of points is defined as

$$SQ(P, Z) = \#\{x \in Z | P(x) > 0\} - \#\{x \in Z | P(x) < 0\}.$$

We will see how this computation is performed. Let $\mathbb{R}$ be our field of coefficients. That is, we work with the ring of polynomials $R = \mathbb{R}[X_1, \ldots, X_n]$. Let I be our ideal and $P \in R$. We want to compute $SQ(P, V(I))$, assuming that I has a finite number of roots on $\mathbb{C}$, the algebraic closure of $\mathbb{R}$.

Let $A = \mathbb{R}[X_1, \ldots, X_n]/I$. This quotient is a ring and, assuming that I is zero-dimensional, it is also a finite-dimensional $\mathbb{R}$-vector space. We can consider the linear maps $m_f : A \to A, m_f(\overline{g}) = \overline{fg}$, called the multiplication maps by f on A. Furthermore, we can consider the maps $\varphi_h : A \times A \to A$, defined by

$$\varphi_h(\overline{f}, \overline{g}) = \mathrm{Trace}(m_{hfg})$$

This φ_h is a bilinear form. Hence, its associated matrix w.r.t a basis of A is symmetric. We can therefore compute its signature with `symsignature`. A theorem by Hermite states that

$$SQ(P, V(I)) = \mathrm{Sig}(\varphi_P)$$

where φ_P is constructed as above. This is how `sturmquery` computes a Sturm Query.

An important corollary of this theorem is that we can compute the number of real roots of I, without multiplicities, which is $SQ(1, V(I))$.

```
> ring r = 0,(x,y,z),dp;
> ideal i = (x-1)*(x+3)*x,y-2,(z-4)^2;
> i = groebner(i);
> ideal b = qbase(i); // qbase gives an ordered monomial basis
> sturmquery(1,b,i);
3
```

Let P be a polynomial in $\mathbb{R}[X_1, \ldots, X_n]$. Assuming that the roots of I are $V(I) = \{p_1, \ldots, p_r\}$, with multiplicities $\{m_1, \ldots, m_r\}$, we can construct the set $\{P(p_1), \ldots, P(p_r)\}$. Then the characteristic polynomial of the multiplication by P in the algebra $\mathbb{R}[X_1, \ldots, X_n]$ is

$$\chi_{m_P}(x) = \pm \prod_{i=1}^{r} (x - P(p_i))^{m_i}$$

If P separates the points of $V(I)$, that is, $P|_{V(I)}$ is injective, this characteristic polynomial has the same number of real roots as I. If we could somehow obtain χ_{m_P}, we could use one of the functions implemented in `rootsur.lib` to calculate the number of real roots of I. We can compute this polynomial by definition, as the determinant of the matrix $m_P - \lambda I$ in the basis `b`. But we can also obtain χ_{m_P} as follows. We observe that

$$\sum_{i=1}^{r} m_i P(p_1)^k = \mathrm{Trace}(m_{P^k}) \quad k = 0, \ldots, N-1$$

where N is the dimension of $\mathbb{R}[X_1, \ldots, X_n]/I$.

This is a simple way of calculating the power sums of the roots of χ_{m_P}. `powersums` does precisely this. We can then call `symmfunc` with the result, which uses Newton's formulae to obtain the symmetric functions of those same roots. Thus we obtain the coefficients of χ_{m_P}. All that remains to be done is to find a polynomial that separates the points of $V(I)$. But this is not difficult, since a polynomial linear on every variable of the ring of polynomials, with generic coefficients, splits the points. Such a polynomial can be obtained by calling the function `rndlnrpoly` (cf. also our command `verify` at the end of this section). For details and a proof of all the results mentioned, see [BPR03] p. 382-289.

`randlinpoly`

INPUTS: Optionally, n, an integer ≤ 10.
OUTPUT: A polynomial linear on each variable of the basering, with pseudo-random coefficients.

The pseudorandom numbers are generated using SINGULAR's `random` function. The random coefficients are taken from $[1, 10^n]$. A default value of 5 is assumed if no n is given.

```
> ring r = 0,(x,y),dp;
> randlinpoli();
34511x+4825y
> randlinpoly(9);
693366185x+159185867y
```

`powersums`

INPUTS: A polynomial P, a Gröbner basis for a zero-dimensional ideal I and a monomial basis B of R/I.
OUTPUT: The powersums of the results of evaluating P at the zeros of $V(I)$.

We calculate the power sums of the result of the evaluation of a polynomial on a variety as explained above.

```
> ring r = 0,(x,y,z),dp;
```

```
> ideal i = x^2+y^2+z^2-4,x^2+2*y^2-5,x*z-1;
> i = groebner(i);
> ideal b = qbase(i);
> poly p = randlinpoly();
> p;
26101x+27289y+59998z
> list l = powersums(p,b,i);
> l;
[1]:
   0
[2]:
   65255484498
[3]:
   0
[4]:
   825881217821537417837
[5]:
   0
[6]:
   23339670462634781593278088678179/2
[7]:
   0
[8]:
   676948636140153157059867280199242205996833/4
```

---

symmfunc

INPUTS: A list S of the power sums of a list of numbers L.
OUTPUT: The symmetric functions of the numbers of L.

Let $L = (l_1, \ldots, l_k)$ be a list of numbers. Then the symmetric functions of the elements of L are the coefficients of the polynomial

$$P(x) = \prod_{i=1}^{k}(x - l_i)$$

To calculate the symmetric functions from the power sums of the elements of L, we use Newton's formulae.

Continuing the previous example,

```
> l = symmfunc(l);
> poly f = univarpoly(l); // This function converts a list into a poly
> f;
x8-32627742249x6+130325791071282173816 5/4x4
-199479244176671480423676212 0073/2x2+
376236650106905991925055367673189462 2489/4
> nrroots(c);
8
```

randcharpoly

INPUTS: A Gröbner basis for an ideal I and a monomial basis B of R/I, where R is the basering. Optionally, n, an integer ≤ 10.
OUTPUT: The characteristic polynomial of m_a, where m_a is the multiplication map by a, a pseudorandom linear polynomial, on the quotient R/I.

A characteristic polynomial of the form previously explained is computed, passing n as a parameter to `randlinpoly`. With a debugging `printlevel`, the function outputs a warning about the result, namely, its probabilistic nature.

Continuing our previous example,

```
> printlevel = printlevel + 2;
> poly ch = randcharpoly(b,i);
*********************************************************************
* WARNING: This polynomial was obtained using  pseudorandom numbers.*
* If you want to verify the result, please use the command          *
*                                                                   *
* verify(p,b,i)                                                     *
*                                                                   *
* where p is the polynomial, b is the monomial basis used, and i    *
* the grobner basis of the ideal                                    *
*********************************************************************
> ch;
z8-7667299543723z6+85470839554754547471764093/4z4-
511326175447571954095566585395118O5867/2z2+
44240547528574552611005022131191711811988402365761/4
> nroots(ch);
8
```

verify

INPUTS: A polynomial P, a Gröbner basis for a zero-dimensional ideal I, and a monomial basis B of the quotient R/I, where R is the basering.
OUTPUT: 1 or 0 (representing true or false, resp.) indicating whether the result provided by `randcharpoly` was useful or not.

This function verifies the result given by `randcharpoly` by computing the rank of φ_1 and comparing it to the square-free form of P. If the rank of the matrix equals the degree of the computed square-free polynomial, then thpolynomial obtained by `randcharpoly` splits the points of $V(I)$.

We verify the previous result:

```
> verify(ch,b,i);
1
```

This library also provides two wrapper functions.

---

nrRootsProbab

INPUTS: A zero-dimensional ideal I.
OUTPUT: The number of real roots of points in $V_{\mathbb{R}}(I)$.

To compute the result, this function uses `randcharpoly`, together with `nrroots`. This result is probabilistic, but it is obtained much more quickly than the, certified, result of nrRootsProbab. The normal way of using these two wrapper functions is to try a few times with `nrRootsProbab`. If the results are the same, they are probably correct. If one needs complete assurance, one can run nrRootsDeterm, but it will generally be much slower.

---

nrRootsDeterm

INPUTS: A zero-dimensional ideal I.
OUTPUT: The number of real roots of points in $V_{\mathbb{R}}(I)$.

To compute the result, this functions uses `sturmquery`. The result is always correct, but it may take much longer than the one given by nrRootsProbab.

```
> ring r = 0,(x,y,z),dp;
> ideal i = x^2+y^2+z^2-4-3x^5z^2y^3,x^2+2*y^2-5,x*z-1+xy^2z+xy^2z;
> int t = timer; // timer holds the accumulated computational time
                 // for the session
> for (j = 0;j < 10;j++) {nrRootsProbab(i);};
6
6
6
6
6
6
6
6
6
6
> timer - t;
22
> t = timer;
> nrRootsDeterm(i);
6
> timer - t;
642
```

The times shown above are expressed in seconds, and correspond to one particular PC. However, they should illustrate the benefits that can be derived from the non-deterministic `nrRootsProbab`.

3 Sign Determination

The library `signcond.lib` contains three routines related to realizable sign conditions. That is, given a set of polynomials $\mathcal{P} = \{P_1, \ldots, P_k\}$ and a zero-dimensional ideal I, we want to know how many points x in $V(I)$ satisfy $P_1(x) = 0, P_2(x) = 0, \ldots, P_k(x) = 0$, how many satisfy $P_1(x) > 0, P_2(x) < 0, \ldots, P_k(x) = 0$, and all the possible sign conditions on the elements of $\mathcal{P}$. Notice that one particular application of this is finding out how many of the roots of an ideal are "positive", that is, have all their coordinates in $\mathbb{R}_{>0}$.

The case where $\mathcal{P} = \{P_1\}$ is easy to solve. If we let $c(P * 0)$ be the number of points x of $V(I)$ that satisfy $P(x) * 0$, where $*$ is one of $>$, $<$ or $=$, then if we solve the system

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & -1 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} c(P_1 = 0) \\ c(P_1 > 0) \\ c(P_1 < 0) \end{pmatrix} = \begin{pmatrix} SQ(1, \mathbf{V}(I)) \\ SQ(P_1, \mathbf{V}(I)) \\ SQ(P_1^2, \mathbf{V}(I)) \end{pmatrix},$$

we are done.

There is a simple way of generalizing this system for the case where $\#\mathcal{P} > 1$. We first solve the 3×3 system shown above for each $P \in \mathcal{P}$. If any of the $c(\ldots)$ is zero, we can reduce the original system to a smaller one, which contains the same information. These smaller systems are then combined together, and we just have to solve a bigger system (for details, see [BPR03] (p. 352–353) and [GVRR99] (p. 140–141)).

`signcnd`

INPUTS: A set $\mathcal{P}$ of polynomials and a Gröbner basis of an ideal I.
OUTPUT: A list, representing the sign conditions realized by the polynomials of $\mathcal{P}$ on the elements of $V(I)$, together with the number of points (counted without multiplicities) at which those conditions are realized.

This function performs the tasks described at the beginning of this section.

```
> ring r = 0,(x,y),dp;
> ideal i = (x^2-4)*(x^4+1),(y+12)*(y-5)*y;
> i = groebner(i);
> ideal P = x,x^2,y,y^2;
> list l = signcnd(P,i);
> psigncnd(P,l);
1 elements of V(I) satisfy {P[1] > 0,P[2] > 0,P[3] = 0,P[4] = 0}
1 elements of V(I) satisfy {P[1] < 0,P[2] > 0,P[3] = 0,P[4] = 0}
1 elements of V(I) satisfy {P[1] > 0,P[2] > 0,P[3] > 0,P[4] > 0}
1 elements of V(I) satisfy {P[1] < 0,P[2] > 0,P[3] > 0,P[4] > 0}
1 elements of V(I) satisfy {P[1] > 0,P[2] > 0,P[3] < 0,P[4] > 0}
1 elements of V(I) satisfy {P[1] < 0,P[2] > 0,P[3] < 0,P[4] > 0}
```

In the example shown above, we did not display the output of `signcnd` directly, since it is a bit cumbersome. Instead, we call another function, `psigncnd`, which performs some pretty-printing. A detailed explanation of the output of `signcnd` can be obtained by the command

```
> example psigncnd;
```

`fstoct`

INPUTS: A Gröbner basis for an ideal I.
OUTPUT: The number of real elements of $V(I)$ which have all their coordinates greater than zero, counted without multiplicities.

This function calls `signcnd` and does some postprocessing on the output.

```
> ideal i = (x^2-4)*(x^4+1),(y+12)*(y-5)*y;
> i = groebner(i);
> fstoct(i);
1
```

References

[BPR03] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry*, volume 10 of *Algorithms and Computation in Mathematics*. Springer-Verlag Berlin Heidelberg, 2003.

[BR90] Riccardo Benedetti and Jean-Jacques Risler. *Real algebraic and semi-algebraic sets*. Hermann, 1990.

[CLO98] David Cox, John Little, and Donal O'Shea. *Using Algebraic Geometry*, volume 185 of *Graduate Texts in Mathematics*. Springer-Verlag New York, 1998.

[GPS05] Gert-Martin Greuel, Gerhard Pfister, and Hans Schönemann. Singular 3.0.0. A Computer Algebra System for Polynomial Computations, Centre for Computer Algebra, University of Kaiserslautern, 2005. `http://www.singular.uni-kl.de`.

[GVRR99] Laureano González-Vega, Fabrice Rouillier, and Marie-Françoise Roy. Symbolic recipes for real solutions. In Arjeh M. Cohen, Hans Cuypers, and Hans Sterk, editors, *Some Tapas of Computer Algebra*, volume 4 of *Algorithms and Computation in Mathematics*, chapter 2, pages 34–65. Springer-Verlag Berlin Heidelberg, 1999.

Papers published in the Reports on Computer Algebra series

[RCA:33] A. Frühbis-Krüger and G. Pfister. Practical Aspects of Algorithmic Resolution of Singularities. October 2004.

[RCA:32] K. Frühbis-Krüger, A.; Krüger and H. Schönemann. Dynamic Modules in SINGULAR. December 2003.

[RCA:31] G.-M. Greuel, C. Lossen, and M. Schulze. Three Algorithms in Algebraic Geometry, Coding Theory, and Singularity Theory. March 2001.

[RCA:30] G.-M. Greuel. Applications of Computer Algebra to Algebraic Geometry, Singularity Theory and Symbolic-Numerical Solving. June 2000.

[RCA:29] G.-M. Greuel. Computer Algebra and Algebraic Geometry – Achievements and Perspectives. Feb 2000.

[RCA:28] T. Siebert. Recursive Computation of Free Resolutions and a Generalized Koszul Complex. Jan 2000.

[RCA:27] O. Bachmann and H.-G. Gräbe. The SymbolicData Project: Towards an Electronic Repository of Tools and Data for Benchmarks of Computer Algebra Software. Jan 2000.

[RCA:26] B. Reinert. Solving One-Sided Equations in Integer Monoid Rings. Jan 2000.

[RCA:25] B. Reinert and D. Zeckzer. Coset Enumeration using Prefix Gröbner Bases in Mrc – An Experimental Approach. Nov 1999.

[RCA:23] Birgit Reinert. Observations on coset enumeration. Nov 1998.

[RCA:22] K. Madlener and F. Otto. Some Applications Of Prefix-Rewriting In Monoids, Groups, And Rings. November 1998.

[RCA:21] G.-M. Greuel, G. Pfister, and H. Schönemann. Singular version 1.2 user manual. June 1998.

[RCA:20] B. Reinert and D. Zeckzer. MRC – A System for Computing Gröbner Bases in Monoid and Group Rings. July 1998.

[RCA:19] B. Reinert, K. Madlener, and T. Mora. A note on nielsen reduction and coset enumeration. February 1998.

[RCA:18] O. Bachmann and H. Schönemann. Monomial Representations for Gröbner Bases Computations. January 1998.

[RCA:17] Thomas Siebert. An algorithm for constructing isomorphisms of modules. January 1998.