AN INTRODUCTION TO SINGULAR

LUIS GARCIA-PUENTE

Introduction

This document introduces Singular, which is a free computer algebra system for polynomial computations. You can download Singular at `http://www.singular.uni-kl.de/`. Singular features one of the fastest implementations of Buchberger's algorithm to compute a Gröbner basis. It also provides multivariate polynomial factorization, resultants, gcd computations, syzygy and free-resolution computations, symbolic-numeric root finding, visualization, primary decomposition, resolution of singularities, and many more related functionalities.

You cay run Singular either in shell mode by typing `Singular` in a terminal or by typing `ESingular`, which runs Singular within Xemacs adding some point-and-click functionality. Singular contains a C-like programming language that allows users to write their own libraries and procedures to extend Singular's capabilities.

1. First steps

Once Singular is started, it awaits an input after the prompt `>`. Every statement has to be terminated by ;

```
2+2;
// 4
```

All objects have a type, for example integer variables are of type `int`. An assignment is done by the symbol `=`, comparison is done by the symbols `==` and `!=` (or `<>`), returning a Boolean variable (0 or 1).

```
int i = 2;
i == 1;
// 0
i != 3;
// 1
```

The value of an object is displayed by typing its name

```
i;
// 2
```

The last displayed result is always available with the underscore symbol `_`. This is particularly useful if you are doing a long interactive computation and you forgot to store the result in a variable.

```
LIB "general.lib";
factorial(37);"";                //37! of type string (as long integer)
```

```
ring r1 = 0,x,dp;
factorial(37,0);        //37! of type number, computed in r1
number p = _;
p;
// 13763753091226345046315979581580902400000000
```

The previous example shows many Singular features which are worth describing. The first line calls for a library called `general.lib`. Many Singular commands are stored in libraries that you need to load in order to use them. The command `factorial` returns a `string` if called with one argument. But once a `ring` is defined, it returns a `number` if called with two arguments, a number is an element in the ground field (in this case $\mathbb{Q}$). This shows that a single procedure can have a different number of inputs and different outputs as well. Furthermore, it exemplifies that most Singular objects need to be defined within the context of a ring. Finally, text starting with `//` denotes a comment and is ignored in calculations.

The best way to learn Singular is to read the online documentation either through Singular's web site or through the local copy included in the installation tarball. You can access this local copy by typing `help;`. Explanation on a single topic, e.g., on `intmat` which defines a matrix of integers, is obtained by

```
help intmat;
```

Integers (`int`), integer matrices (`intmat`), integer vectors (`intvec`), and strings (`string`) can be defined without a ring being active. Integer matrices are defined as follows

```
intmat m[3][3] = 1,2,3,4,5,6,7,8,9;
```

This line defines a 3×3-matrix of integers and initializes it with some values. A single matrix entry may be selected and changed using square brackets [and].

```
m[1,2]=0;
m;
// 1,0,3,
// 4,5,6,
// 7,8,9
print(m); // notice that the command print makes the output look nicer.
//     1     0     3
//     4     5     6
//     7     8     9
```

To calculate the trace of this matrix, we use a `for` loop. The curly brackets `{` and `}` denote the beginning and end of a block. If you define a variable without giving an initial value, as the variable tr in the example below, Singular assigns a default value for the specific type. In this case, the default value for integers is 0.

```
int tr;
for ( int j=1; j <= 3; j++ ) { tr=tr + m[j,j]; }
tr;
// 15
```

Strings are delimited by `"` (double quotes). They may be used to comment the output of a computation or to give it a nice format. If a string contains valid Singular commands, it can be executed using the function `execute`. The result is the same as if the commands would have been written on the command line. This feature is especially useful to define new rings inside procedures.

```
"example for strings:";
// example for strings:
string s="The element of m ";
s = s + "at position [2,3] is:";  // concatenation of strings by +
s , m[2,3] , ".";
// The element of m at position [2,3] is: 6 .
s="m[2,1]=0; m;";
execute(s);
// 1,0,3,
// 0,5,6,
// 7,8,9
```

This example shows that expressions can be separated by a `,` (comma) giving a list of expressions. Singular evaluates each expression in this list and prints all results separated by spaces.

You can plot curves and surfaces in Singular with the `plot` command in the library `surf.lib`. So before we get to more technical stuff, lets create some pretty pictures.

```
LIB "surf.lib";
// plane curve
ring r0 = 0,(x1,x2),dp;
ideal I = x1^3 - x2^2;
plot(I);
// Singular Logo
ring r = 0,(x,y,z),dp;
poly logo = ((x+3)^3 + 2*(x+3)^2 - y^2)*(x^3 - y^2)*((x-3)^3-2*(x-3)^2-y^2);
plot(logo);
// Steiner surface
ideal J2 = x^2*y^2+x^2*z^2+y^2*z^2-17*x*y*z;
plot(J2);
// Whitney umbrella
plot(z^2-x^2*y);
```

2. Rings and Gröbner bases

To calculate with objects such as ideals, matrices, modules, and polynomial vectors, a ring has to be defined first.

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

The definition of a ring consists of three parts: the first part determines the ground field, the second part determines the names of the ring variables, and the third part determines the monomial ordering to be used. The example above declares a polynomial ring called `r` with a ground field of characteristic 0 (i.e., the rational numbers) and ring variables called

x, y, and z. The `dp` at the end means that the degree reverse lexicographical ordering should be used.

Other ring declarations:

```
ring r1=32003,(x,y,z),dp;
//    characteristic 32003, variables x, y, and z and ordering dp.
ring r2=32003,(a,b,c,d),lp;
//  characteristic 32003, variables a, b, c, d and lexicographical ordering
```

For the complete list of monomial orderings see `http://www.singular.uni-kl.de/Manual/2-0-5/sing_364.htm`

Typing the name of a ring prints its definition. The example below shows, that the default ring in Singular is Z/32003[x,y,z] with degree reverse lexicographical ordering:

```
ring r5;
r5;
 //   characteristic : 32003
 //   number of vars : 3
 //        block   1 : ordering dp
 //                  : names    x y z
 //        block   2 : ordering C
```

Defining a ring makes it the current active ring (Singular assigns it the variable `basering`), so each ring definition above switches to a new basering. The concept of rings in Singular is discussed in detail in the Chapter "Rings and orderings" of the Singular manual.

The basering now is r5, to switch back to a different ring we need to use the function setring:

```
setring r;
```

Once a ring is active, we can define polynomials. A monomial, say x^3, may be entered in two ways: either as `x^3`, or in short-hand notation as `x3`. Note, that the short-hand notation is forbidden if the name of the ring variable consists of more than one character. Note also that Singular always expands brackets and automatically sorts the terms with respect to the monomial ordering of the basering.

```
poly f =  x3+y3+(x-y)*x2y2+z2;
f;
// x3y2-x2y3+x3+y3+z2
```

The command `size` determines in general the number of "single entries" in an object. In particular, for polynomials, `size` determines the number of monomials.

```
size(f);
// 5
```

A natural question is to ask if a point e.g. $(x, y, z) = (1, 2, 0)$ lies on the variety defined by the polynomials f and g. For this we define an ideal generated by both polynomials, substitute the coordinates of the point for the ring variables, and check if the result is zero:

```
poly g =  f^2 *(2x-y);
ideal I = f,g;
ideal J= subst(I,var(1),1);
```

```
J = subst(J,var(2),2);
J = subst(J,var(3),0);
J;
// J[1]=5
// J[2]=0
```

Since the result is not zero, the point $(1, 2, 0)$ does not lie on the variety $V(f, g)$.

Another question is to decide whether some function vanishes on a variety, or in algebraic terms if a polynomial is contained in a given ideal. For this we calculate a Gröbner basis using the command `groebner` and afterwards reduce the polynomial with respect to this Gröbner basis.

```
ideal sI = groebner(f);
reduce(g,sI);
// 0
```

As the result is 0 the polynomial g belongs to the ideal defined by f.

The function `groebner`, like many other functions in Singular, prints a protocol during calculation, if desired. The command `option(prot);` enables protocoling whereas `option(noprot);` turns it off. Another useful option for `groebner` is `option(redSB)` which forces Singular to return a *reduced* Gröbner basis. In the Singular documentation, Gröbner bases are referred to as *standard bases*. In fact, Singular provides also the command `std` to compute a standard basis. Below we will explain the difference between both commands.

The command `kbase` calculates a basis of the polynomial ring modulo an ideal, if the quotient ring is finite dimensional. As an example we calculate the Milnor number of a hypersurface singularity in the global and local case. This is the vector space dimension of the polynomial ring modulo the Jacobian ideal in the global case respectively of the power series ring modulo the Jacobian ideal in the local case.

The Jacobian ideal is obtained with the command `jacob`.

```
ideal J = jacob(f);
// // ** redefining J **
J;
// J[1]=3x2y2-2xy3+3x2
// J[2]=2x3y-3x2y2+3y2
// J[3]=2z
```

Singular prints the line `// ** redefining J **`. This indicates that we have previously defined a variable with name `J` of type ideal within the ring `r` (see above).

To obtain a representing set of the quotient vector space we first calculate a Gröbner basis, then we apply the function `kbase` to this standard basis.

```
J = groebner(J);
ideal K = kbase(J);
K;
// K[1]=y4
// K[2]=xy3
// K[3]=y3
// K[4]=xy2
// K[5]=y2
// K[6]=x2y
// K[7]=xy
// K[8]=y
// K[9]=x3
// K[10]=x2
// K[11]=x
// K[12]=1
//
size(K);
// 12
```

The command size gives the desired vector space dimension. As in Singular the functions may take the input directly from earlier calculations, the whole sequence of commands may be written in one single statement.

```
size(kbase(groebner(jacob(f))));
// 12
```

When we are not interested in a basis of the quotient vector space, but only in the resulting dimension we may even use the command vdim and write:

```
vdim(groebner(jacob(f)));
// 12
```

2.1. **Gröbner bases for local orderings and multiplicities.** The following is an example of a ring with the so-called "negative degree reverse lexicographical" local monomial ordering:

```
ring r=0,(x,y,z),ds; // ds is a local ordering
```

Define an ideal in this ring:

```
 poly s1=x3-y*z;
 poly s2=y3-x*z;
 poly s3=z3-x*y;
 ideal I=(s1,s2,s3);
```

A standard basis with respect to a local ordering is an analogue of a Gröbner basis for global monomial orderings. One can use a standard basis to compute the multiplicity at the origin:

```
 ideal J=std(I);
 mult(J);
// 11
```

This multiplicity is the number of elements in the basis of standard monomials for the local quotient ring, which can be created as before:

```
 ideal K = kbase(J);
 K;
// K[1]=z4
// K[2]=z3
// K[3]=z2
// K[4]=z
// K[5]=y3
// K[6]=y2
// K[7]=y
// K[8]=x3
// K[9]=x2
// K[10]=x
// K[11]=1
```

3. Solving systems of polynomial equations

Singular applies symbolic-numeric methods to find the solutions of a system of polynomial equations. Even though the complexity of these methods is higher than the complexity of purely numerical methods it is desirable in many cases, for example, to avoid troubles near singularities or to solve systems involving parameters providing a simultaneous solution for all parameter values. As a first example, we will solve the following system of linear equations in x, y, z, and u with free parameters a, b, c, and d.

$$\begin{aligned} 3x + y + z - u &= a \\ 13x + 8y + 6z - 7u &= b \\ 14x + 10y + 6z - 7u &= c \\ 7x + 4y + 3z - 3u &= d \end{aligned}$$

```
// define a ring with 4 parameters and 4 indeterminates
ring R = (0,a,b,c,d), (x,y,z,u), dp;
//
//the equations
//
ideal I = 3x  + y   + z  - u  - a,
          13x + 8y  + 6z - 7u - b,
          14x + 10y + 6z - 7u - c,
          7x  + 4y  + 3z - 3u - d;
//
// compute a reduced Groebner basis
//
option(redSB);
ideal G = groebner(I);
//
// return the generators of G as monic polynomials
//
simplify(G,1);
// _[1]=u+(6/5a+4/5b+1/5c-12/5d)
// _[2]=z+(16/5a-1/5b+6/5c-17/5d)
// _[3]=y+(3/5a+2/5b-2/5c-1/5d)
// _[4]=x+(-6/5a+1/5b-1/5c+2/5d)
```

So the unique solution to our system is

$$\begin{aligned} u &= -(6/5a + 4/5b + 1/5c - 12/5d),\\ z &= -(16/5a - 1/5b + 6/5c - 17/5d),\\ y &= -(3/5a + 2/5b - 2/5c - 1/5d),\\ x &= -(-6/5a + 1/5b - 1/5c + 2/5d). \end{aligned}$$

The performance of Buchberger's algorithm for computing Gröbner bases depends heavily in the chosen term order. In general, lexicographic Gröbner bases are the most expensive to compute. But if the ideal is zero-dimensional, that is, the system has a finite number of solutions, we can use the *FGLM* algorithm to move from any Gröbner basis to a lexicographic one.

```
LIB "ring.lib";
timer = 0;
//
ring R = 0, (x,y,z), dp;
ideal I = x7+y7, y7+z7, x7+z7+2, x6y+y6z+z6+x;
//
// compute  reduced GB w.r.t. dp
//
option(redSB);
ideal J=std(I);
timer;
// 4
timer=0;
//
// define a new ring S by changing the term order of R
def S=changeord("lp");
setring S;
ideal J = fglm(R,J);
timer;
// 0
timer=0;
//
// the command groebner decides on the best method to compute a GB
// first we need to copy the original ideal I into the ring S
ideal I =  imap(R,I);
ideal K = groebner(I);
timer;
// 4
timer=0;
```

```
// finally the direct computation of a reduced lex Groebner basis

K = std(I);
// ......................................................
```

The last computation takes a very long time so it is a good moment to point out that one forces Singular to quit with `^C-^C`.

3.1. **Elimination of variables.** Elimination of variables is a common way to solve systems of polynomials. In Singular, one may either choose an appropriate ordering (for example, a product ordering) or apply the command `eliminate`, which computes in the present ordering modified by an extra weight vector.

Next, we will determine all the z-values of the zeros of the following system of equations

$$\begin{aligned} 4z^8y - 5z^3x - 3x^2y + xy^2 &= 8 \\ z^9 - 3z^5xy + z^3x - (7x^2+1)y - 2xy^2 &= 1 \\ z^9x(5+2y) + 5z^8y + 5z - x^2y - 4xy^2 &= -1 \end{aligned}$$

```
// First we define a ring with a product ordering
// to eliminate the first 2 variables
ring R = 0, (x,y,z), (dp(2),lp);
ideal I = 4z8y-5z3x-3x2y+xy2-8,
          z9-3z5xy+z3x-(7x2+1)*y-2xy2-1,
          z9x*(5+2y)+5z8y+5z-x2y-4xy2+1;
option(redSB);
ideal J = groebner(I);
// find the univariate polynomial in z
simplify(lead(J),1);
// _[1]=z85
// _[2]=y
// _[3]=x
poly g = J[1];
```

We will next compute all the zeros of the univariate polynomial g. We will use Laguerre's algorithm in Singular implemented in the library `solve.lib`.

```
LIB "solve.lib";
// define a ring with complex coefficients to store the zeros
ring Rfloat = (complex,10,I), (x,y,z), lp;
poly g = imap(R,g);
list L = laguerre_solve(g,100,100);
size(L);
// 85
//
// to pick the real zeros we can loop thru L
for (int i=1; i<=size(L); i++)
```

```
{
   if (impart(L[i])==0) {L[i];}
}
// -1.8135284651
// -1.559997662
// -0.2679041558
// 0.9657875564
// 1.1035140081
// 1.2770264747
// 2.7768083
```

As an exercise, implement a method to test that these points are indeed real zeros. We could have found the roots of g without defining a new ring using the command `solve` in the library `solve.lib`.

3.2. **Primary decomposition.** When solving a system of equations, it is important to understand the geometry of the zero-set. An important step towards this goal is finding the *primary decomposition* of the defining ideal. Singular provides different methods to find the primary decomposition of an ideal. One can used several canned-algorithms implemented in the library `primdec.lib`, or, in case those procedures fail to terminate, perform a manually-guided primary decomposition computing ideal saturations with the command `sat` included in the library `elim.lib`.

Singular also provides an efficient implementation (`facstd`) of the factorizing Gröbner bases algorithm. This procedure returns a list of ideals whose intersection equals the radical of the input ideal. It is important to note that the decomposition returned by `facstd` is not a complete (irreducible) decomposition. It only splits the decomposition problem into smaller problems.

The library `primdec.lib` contains two different algorithms to compute primary decomposition: the Gianni-Trager-Zacharias algorithm and the Shimoyama-Yokoyama algorithm. Below is a simple example on how to use this library.

```
LIB "primdec.lib";
ring R = 0, (x,y,z),dp;
ideal I = xy2-y3-xy+y2,
          x2y-x3,
          x3-y3-x2+y2;
primdecGTZ(I);
//[1]:
//   [1]:
//      _[1]=x-y
//   [2]:
//      _[1]=x-y
//[2]:
//   [1]:
//      _[1]=y-1
//      _[2]=x2
//    [2]:
//       _[1]=y-1
//       _[2]=x
//[3]:
//    [1]:
//       _[1]=y
//       _[2]=x2
//    [2]:
//       _[1]=y
//       _[2]=x
```

Acknowledgments

This tutorial relies heavily on the introductory Singular tutorial available online, some introductory notes written by Anton Leykin and the paper “Singular: A computer algebra system” by Christoph Lossen.

Department of Mathematics, Texas A&M University, College Station, TX 77843-3368, USA

E-mail address: lgp@math.tamu.edu

URL: http://www.math.tamu.edu/~lgp