Computing with Polynomials in Singular

1. Polynomials, Resultants, and Factorization
   - computer algebra for polynomial computations
   - resultants to eliminate variables

2. Gröbner Bases and Multiplication Matrices
   - ideals of polynomials and Gröbner bases
   - normal forms and multiplication matrices
   - multiplicity as the dimension of the local quotient ring

3. Formulas for the 4-bar Coupler Point
   - formulation of the problem
   - processing a Gröbner basis

MCS 507 Lecture 35
Mathematical, Statistical and Scientific Software
Jan Verschelde, 16 November 2012
Computing with Polynomials in Singular

1. Polynomials, Resultants, and Factorization
   - computer algebra for polynomial computations
   - resultants to eliminate variables

2. Gröbner Bases and Multiplication Matrices
   - ideals of polynomials and Gröbner bases
   - normal forms and multiplication matrices
   - multiplicity as the dimension of the local quotient ring

3. Formulas for the 4-bar Coupler Point
   - formulation of the problem
   - processing a Gröbner basis
Singular is a computer algebra system for polynomial computations, with special emphasis on commutative and non-commutative algebra, algebraic geometry, and singularity theory, under the GNU General Public License.

Singular’s core algorithms handle

- polynomial factorization and resultants
- characteristic sets and numerical root finding
- Gröbner, standard bases, and free resolutions.

Its development is directed by Wolfram Decker, Gert-Martin Greuel, Gerhard Pfister, and Hans Schönemann, within the Dept. of Mathematics at the University of Kaiserslautern.
Advanced algorithms are contained in more than 90 libraries, written in a C-like programming language.

For visualization of plane curves and surfaces, the software `surf` and `surfex` need to be installed.

We run Singular explicitly in Sage via

1. the class Singular, do `help(singular);` or
2. opening a Terminal Session with Singular, type `singular_console();` in a Sage terminal.
Computing with Polynomials in Singular

1. Polynomials, Resultants, and Factorization
   - computer algebra for polynomial computations
   - resultants to eliminate variables

2. Gröbner Bases and Multiplication Matrices
   - ideals of polynomials and Gröbner bases
   - normal forms and multiplication matrices
   - multiplicity as the dimension of the local quotient ring

3. Formulas for the 4-bar Coupler Point
   - formulation of the problem
   - processing a Gröbner basis
Before we enter a polynomial in Singular, we must *first* decide on

1. the type of the coefficients,
2. names and order of the variables,
3. which order will be used for the monomials.

In a Singular console, at the prompt `>`, we type

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

We declared a ring `r` over the rational numbers (characteristic 0), for polynomials in `x` and `y`, in the degree lexicographic ordering (as in a dictionary).
entering polynomials

> poly p = 4*y^2 - 4 + 4*x*y + x^2;
> p;
\texttt{x2+4xy+4y2-4}

We see \( p \) shown in the degree lexicographic order:
- first the highest degree monomials (\( x^2 = x^2 \)),
- with those with highest powers of \( x \) appearing first.

Then we define a second polynomial \( q \):

> poly q = 4*x^2 - 4*x*y + y^2 - 4;

The polynomials \( p \) and \( q \) define a system.
elimination with a resultant

```plaintext
> p;
x2+4xy+4y2-4
> q;
4x2-4xy+y2-4

To eliminate \( x \) from the system \( p(x, y) = 0 = q(x, y) \):

```plaintext
> poly rx = resultant(p, q, x);
> rx;
625y4-1000y2+144
``` 

All values for \( y \) for which \( p(x, y) = 0 = q(x, y) \) also make \( \text{resultant}(p, q, x) \) zero.
loading a library

> LIB "solve.lib";
// ** loaded /usr/local/sage-4.7.1/local/share/singular/solve.lib (12231,2009-11-02)
// ** ../singular/triang.lib (12231,2009-11-02)
// ** ../singular/elim.lib (12231,2009-11-02)
// ** ../singular/ring.lib (12231,2009-11-02)
// ** ../singular/primdec.lib (12962,2010-07-09)
// ** ../singular/absfact.lib (12231,2009-11-02)
// ** ../singular/matrix.lib (12898,2010-06-23)
// ** ../singular/nctools.lib (12790,2010-05-14)
// ** ../singular/random.lib (12827,2010-05-28)
// ** ../singular/poly.lib (12443,2010-01-19)
// ** ../singular/inout.lib (12541,2010-02-09)
// ** ../singular/general.lib (12904,2010-06-24)
computing all roots

We ask to compute all roots with 10 decimal places of precision:

```plaintext
> list L = laguerre_solve(rx,10);
> L;
[1]:
   -1.2
[2]:
   -0.4
[3]:
   0.4
[4]:
   1.2
```
Root finding is related to factorization:

```plaintext
> LIB "factor.lib";
// ** loaded /usr/local/sage-4.7.1/local/share/
   singular/factor.lib (12231,2009-11-02)
> factorize(rx);
[1]:
   [_1]=1
   [_2]=5y+2
   [_3]=5y-2
   [_4]=5y-6
   [_5]=5y+6
[2]:
   1,1,1,1,1
```

The factors give the exact roots: $\pm \frac{2}{5}, \pm \frac{6}{5}$. 
solving a system

\[ \text{solve}([p,q]); \]

\[
\begin{align*}
[1]: & \quad [1]: \\
& \quad 0.4 \\
& \quad [2]: \\
& \quad -1.2 \\
[2]: & \quad [1]: \\
& \quad -1.2 \\
& \quad [2]: \\
& \quad -0.4 \\
[3]: & \quad [1]: \\
& \quad 1.2 \\
& \quad [2]: \\
& \quad 0.4 \\
[4]: & \quad [1]: \\
& \quad -0.4 \\
& \quad [2]: \\
& \quad 1.2
\end{align*}
\]

The four solutions are

\((+0.4, -1.2), (-1.2, -0.4),
\((-0.4, +1.2), (+1.2, +0.4)).\)
// 'solve' created a ring, in which a list SOL
// of numbers (the complex solutions) is stored.
// To access the list of complex solutions,
// type (if the name R was assigned
// to the return value):
// setring R; SOL;
// characteristic : 0 (complex:8 digits,
// additional 8 digits)
// 1 parameter : i
// minpoly : (i^2+1)
// number of vars : 2
// block 1 : ordering lp
// : names x y
// block 2 : ordering C
Computing with Polynomials in Singular

1. Polynomials, Resultants, and Factorization
   - computer algebra for polynomial computations
   - resultants to eliminate variables

2. Gröbner Bases and Multiplication Matrices
   - ideals of polynomials and Gröbner bases
   - normal forms and multiplication matrices
   - multiplicity as the dimension of the local quotient ring

3. Formulas for the 4-bar Coupler Point
   - formulation of the problem
   - processing a Gröbner basis
Gröbner bases for ideals

Given a set $S$ of polynomials, we call the set of all polynomial combinations of the polynomials in $S$ an *ideal*.

For $p$ and $q$ (as before), we define the ideal $I$ as

```plaintext
> ideal I = p, q;
```

A Gröbner basis solves the membership problem: Given any polynomial $f$, does $f$ belong to $I$?

```plaintext
> ideal g = groebner(I);
> g;
```

```
[1] 20xy + 15y^2 - 12
[2] 4x^2 - 4xy + y^2 - 4
[3] 125y^3 + 48x - 164y
```
changing rings

A lexicographic monomial order eliminates variables and the resulting Gröbner basis has a triangular form.

A lexicographic monomial order in Singular is \( lp \):

\[
\text{> ring s = 0, (x,y), lp;}
\]

After this ring definition, the current ring becomes \( s \) and the ideal \( I \) does not exist in this ring.

To return to the original ring \( r \), we type

\[
\text{> setring r;}
\]
reduced standard basis

After returning to the ring $r$, we have

\[
\begin{align*}
> & I; \\
I[1] &= x^2 + 4xy + 4y^2 - 4 \\
I[2] &= 4x^2 - 4xy + y^2 - 4
\end{align*}
\]

Before converting to a lexicographic order, we must force the computation of a reduced standard basis:

\[
\begin{align*}
> & \text{option(redSB);} \\
> & I = \text{std}(I); \\
> & I; \\
I[1] &= 20xy + 15y^2 - 12 \\
I[2] &= 5x^2 + 5y^2 - 8 \\
I[3] &= 125y^3 + 48x - 164y
\end{align*}
\]
converting to lex order

Instead of computing a Gröbner basis in the ring \( s \) from scratch, we convert with the FGLM Algorithm:

\>
setring s;
ideall J = fglm(r,I);
J;
\>
\[ J[1] = 625y^4 - 1000y^2 + 144 \]
\[ J[2] = 48x + 125y^3 - 164y \]

We see that after conversion we get a triangular form.
Computing with Polynomials in Singular

1. Polynomials, Resultants, and Factorization
   - computer algebra for polynomial computations
   - resultants to eliminate variables

2. Gröbner Bases and Multiplication Matrices
   - ideals of polynomials and Gröbner bases
   - normal forms and multiplication matrices
   - multiplicity as the dimension of the local quotient ring

3. Formulas for the 4-bar Coupler Point
   - formulation of the problem
   - processing a Gröbner basis
normal forms

> setring r;
> g;
\[
\begin{align*}
g[1] &= 20xy + 15y^2 - 12 \\
g[2] &= 4x^2 - 4xy + y^2 - 4 \\
g[3] &= 125y^3 + 48x - 164y
\end{align*}
\]

With a Gröbner basis \( g \) we can rewrite any polynomial modulo the ideal into a unique normal form:

> reduce(x^2, g);
-\( y^2 + 8/5 \)
> reduce(x*y, g);
-\( 3/4y^2 + 3/5 \)
> reduce(x*y^2, g);
\( 36/125x - 48/125y \)

Monomials returned by \texttt{reduce} are 1, \( x \), \( y \), and \( y^2 \).
For a polynomial system with finitely many solutions, say $N$, the number of different monomials that can appear in the outcome of `reduce` equals $N$.

> LIB "rootsmr.lib";
// ** loaded /usr/local/sage-4.7.1/local/share/\
  singular/rootsmr.lib
// ** ../singular/rootsur.lib
// ** ../singular/linalg.lib (12258,2009-11-09)

With the `rootsmr.lib` written by Enrique A. Tobis, we can compute the generalization of a companion matrix for several variables.
the quotient ring

A basis for the quotient ring is computed as

> ideal B = qbase(I);
> B;
B[1]=y2
B[2]=x
B[3]=y
B[4]=1

Because the system defined by \( p \) and \( q \) has four solutions, the computation modulo \( p \) and \( q \) defines a four dimensional quotient ring.
a multiplication matrix

> matrix m = matmult(x,B,I);
> print(m);
0, -1, -3/4, 0,
36/125, 0, 0, 1,
-48/125, 0, 0, 0,
0, 8/5, 3/5, 0

The columns of the matrix $m$ contain the normal forms of the multiples of the basis elements in $B$ with $x$. Recall

> reduce(x*y^2,g);
36/125x-48/125y

We constructed the eigenvalue problem $x \ B = m \ B$. 
The eigenvalues of the multiplication matrix give the values for the \( x \) coordinate of the solutions.

> eigenvals(m);
[1]:
  _[1]=-6/5
  _[2]=-2/5
  _[3]=2/5
  _[4]=6/5
[2]:
  1,1,1,1

To avoid introducing multiplicities, one should take a random linear form instead of multiplying with \( x \).
Computing with Polynomials in Singular

1. Polynomials, Resultants, and Factorization
   - computer algebra for polynomial computations
   - resultants to eliminate variables

2. Gröbner Bases and Multiplication Matrices
   - ideals of polynomials and Gröbner bases
   - normal forms and multiplication matrices
   - multiplicity as the dimension of the local quotient ring

3. Formulas for the 4-bar Coupler Point
   - formulation of the problem
   - processing a Gröbner basis
intersection multiplicity

Let $z \in \mathbb{C}^n$ be an isolated solution of $f(x) = 0$. The intersection multiplicity $\mu(z)$ is defined algebraically as the dimension of the local quotient ring:

$$\mu(z) = \dim_{\mathbb{C}} \left( \mathbb{C}[x]/\langle x_1 - z_1, x_2 - z_2, \ldots, x_n - z_n \rangle \right).$$

In analogy with global quotient rings, consider $\langle x^2, xy, y^2 \rangle$:

The big black dots are the generators $x^2$, $xy$, and $y^2$. The small black dots are the generated monomials.

The monomials 1, $x$, and $y$ represented by the empty circles are a basis for the quotient ring. The multiplicity of $(0, 0)$ equals three, the number of monomials under the staircase.
local ordering, standard basis, and multiplicity

\[ l = \langle x^3 - yz, y^3 - xz, z^3 - xy \rangle, \]

> ring R = 32003, (x, y, z), ds;
> poly f1 = x**3-y*z;
> poly f2 = y**3-x*z;
> poly f3 = z**3-x*y;
> ideal i = f1, f2, f3;
> ideal j = std(i);
> j;

j[1] = xy - z3
j[3] = yz - x3
j[4] = x4 - z4
j[5] = y4 - z4
j[6] = z5
> mult(j);
11
Computing with Polynomials in Singular

1. Polynomials, Resultants, and Factorization
   - computer algebra for polynomial computations
   - resultants to eliminate variables

2. Gröbner Bases and Multiplication Matrices
   - ideals of polynomials and Gröbner bases
   - normal forms and multiplication matrices
   - multiplicity as the dimension of the local quotient ring

3. Formulas for the 4-bar Coupler Point
   - formulation of the problem
   - processing a Gröbner basis
a 4-bar mechanism (recall lecture 15)

We have 4 bars and 4 joints, labeled A, B, C, and D:

\[ A \quad B(a, 0) \quad C \quad D \]

\[ R \quad D \quad b \quad L \]

\[ C \quad L \quad B(a, 0) \]

\[ t \text{ is the angle between } AB \text{ and crank } AC, \text{ above } t = \frac{\pi}{2} \text{ and } t = \pi. \]

As the crank turns, the point E traces a curve.
the coordinates of the coupler point

For an angle \( t \), the coordinates of the crank \( C \) are \((L \cos(t), L \sin(t))\).

Abbreviating \( \cos(t) \) by \( c \) and \( \sin(t) \) by \( s \), we have to solve the system

\[
\begin{align*}
(x - a)^2 + y^2 - r^2 &= 0 \\
(x - Lc)^2 + (y - Ls)^2 - R^2 &= 0 \\
c^2 + s^2 - 1 &= 0
\end{align*}
\]

The first equation expresses that the point \( D \) with coordinates \((x, y)\) is at distance \( r \) from the point \( B \) with coordinates \((a, 0)\).

The second equation expresses that the point \( D \) with coordinates \((x, y)\) is at distance \( R \) from the crank \( C \).

The third equation originates from the substitution \( c = \cos(t) \) and \( s = \sin(t) \), needed to have an equivalent algebraic system.
a session in a Sage notebook

# The crank C of length L and angle t is (L*cos(t),L*sin(t)).
# The point D(x,y) is at distance r from B(a,0) => the equation f,
# f = (x-a)^2 + y^2 - r^2 = 0.
# The point D(x,y) is at distance R from C(L*cos(t),L*sin(t)) =>
# the equation g, g = (x - L*cos(t))^2 + (y - L*sin(t))^2 - R^2 = 0.
# Replacing cos(t) and sin(t) by c and s respectively, adding
# h = c^2 + s^2 - 1 = 0 we obtain a system of algebraic equations
# in (x,y) and with parameters (a,r,R,L).

x,y,a,r,t,L,R = var('x,y,a,r,t,L,R')
f = (x - a)^2 + y^2 - r^2
g = (x - L*cos(t))^2 + (y - L*sin(t))^2 - R^2
c,s = var('c,s')
h = c^2 + s^2 - 1
gcs = g.subs(cos(t) == c).subs(sin(t) == s)
print [f,gcs,h]

[(a - x)^2 - r^2 + y^2, (L*s - y)^2 + (L*c - x)^2 - R^2, \c^2 + s^2 - 1]
print type(f)
# We will now go to singular, declaring a polynomial ring.
P = PolynomialRing(QQ,8,names='xyarLRcs', order='lex')
P
# We cast the original polynomials into the ring:
F = P(f); G = P(gcs); H = P(h); print [F,G,H]

<type 'sage.symbolic.expression.Expression'>
[x^2 - 2*x*a + y^2 + a^2 - r^2, x^2 - 2*x*L*c + y^2
   - 2*y*L*s + L^2*c^2 + L^2*s^2 - R^2, c^2 + s^2 - 1]

# We compute a lexicographic Groebner basis with Singular
I = P.ideal(F,G,H)
gb = I.groebner_basis('libsingular:slimgb'); print gb
Computing with Polynomials in Singular

1. Polynomials, Resultants, and Factorization
   - computer algebra for polynomial computations
   - resultants to eliminate variables

2. Gröbner Bases and Multiplication Matrices
   - ideals of polynomials and Gröbner bases
   - normal forms and multiplication matrices
   - multiplicity as the dimension of the local quotient ring

3. Formulas for the 4-bar Coupler Point
   - formulation of the problem
   - processing a Gröbner basis
a lexicographic Gröbner basis

\[ \begin{align*}
x^2 - 2xLc + y^2 - 2yLs + L^2 - R^2, \\
x y L s - 1/2 x r^2 - 1/2 x L^2 s^2 + 1/2 x R^2 + y^2 a \\
- y^2 L c - 3/2 y a L s + 1/2 y L^2 c s + 1/4 a^3 \\
- 3/4 a^2 L c - 1/4 a r^2 + 3/4 a L^2 - 3/4 a R^2 \\
+ 3/4 r^2 L c - 1/4 L^3 c + 1/4 L R^2 c, \\
x a - x L c - y L s - 1/2 a^2 + 1/2 r^2 + 1/2 L^2 - 1/2 R^2, \\
y^2 a^2 - 2 y^2 a L c + y^2 L^2 - y a^2 L s + 2 y a L^2 c s \\
- y r^2 L s - y L^3 s + y L R^2 s + 1/4 a^4 - a^3 L c \\
- 1/2 a^2 r^2 - a^2 L^2 s^2 + 3/2 a^2 L^2 - 1/2 a^2 R^2 \\
+ a r^2 L c - a L^3 c + a L R^2 c + 1/4 r^4 + r^2 L^2 s^2 \\
- 1/2 r^2 L^2 - 1/2 r^2 R^2 + 1/4 L^4 - 1/2 L^2 R^2 + 1/4 R^4, \\
c^2 + s^2 - 1 \end{align*} \]
processing the Gröbner basis

# The second element in the lexicographic Groebner basis is
# linear in x and we can express the x coordinate in terms of y
# and the other parameters of the problem.
# To solve the expression for x, we have to convert the singular
# polynomial back to a symbolic Sage expression.
# The conversion has two difficulties, because of Python:
# (1) instead of ^ we must use ** for powers;
# (2) the 1/2 gets evaluated to 0 in Python.
s = str(2*gb[2])
L = s.split('^'); print L
pL = reduce(lambda x,y: x+' ** '+y,L); print 'pL =', pL
var('x,y,a,c,s,L,r,R')
px = eval(pL); print 'px =', px; print type(px)
xsol = solve([px==0],x); print xsol
print 'x =', xsol[0].rhs()
equations for x

gb[2] = x*a - x*L*c - y*L*s - 1/2*a^2 + 1/2*r^2 + 1/2*L^2
    - 1/2*R^2
[2*x*a - 2*x*L*c - 2*y*L*s - a', '2 + r', '2 + L', '2 - R', '2']
pL = 2*x*a - 2*x*L*c - 2*y*L*s - a**2 + r**2 + L**2 - R**2
px = -2*L*c*x - 2*L*s*y + L^2 - R^2 - a^2 + 2*a*x + r^2
<type 'sage.symbolic.expression.Expression'>
[
  x == -1/2*(2*L*s*y - L^2 + R^2 + a^2 - r^2)/(L*c - a)
]
x = -1/2*(2*L*s*y - L^2 + R^2 + a^2 - r^2)/(L*c - a)
equations for y

# The third element in the lexicographic Groebner basis
# depends only on y and the parameters.
print gb[3]
# We convert again to a symbolic Sage expression:
s = str(4*gb[3])
L = s.split('^'); pLy = reduce(lambda x,y: x+'**'+y,L);
var('y,a,c,s,L,r,R')
py = eval(pLy); print py
ysol = solve([py==0],y); print 'y =', ysol[0].rhs()
output of the Sage commands

\[
y^2 a^2 - 2 y^2 a L c + y^2 L^2 - y a^2 L s + 2 y a L^2 c s - y r^2 L s - y L^3 s + y L R^2 s + 1/4 a^4 - a^3 L c - 1/2 a^2 r^2 - a^2 L^2 s^2 + 3/2 a^2 L^2 - 1/2 a^2 R^2 + a r^2 L c - a L^3 c + a L R^2 c + 1/4 r^4 + r^2 L^2 s^2 - 1/2 r^2 L R^2 + 1/4 L^4 - 1/2 L^2 R^2 + 1/4 R^4 - 4 L^2 a^2 s^2 + 8 L^2 a c s y + 4 L^2 r^2 s^2 - 4 L^3 a c - 4 L^3 s y + 4 L R^2 a c + 4 L R^2 s y - 4 L a^3 c - 4 L a^2 s y + 4 L a c r^2 - 8 L a c y^2 - 4 L r^2 s y + L^4 - 2 L^2 R^2 + 6 L^2 a^2 - 2 L^2 r^2 + 4 L^2 y^2 + R^4 - 2 R^2 a^2 - 2 R^2 r^2 + a^4 - 2 a^2 r^2 + 4 a^2 y^2 + r^4
\]

\[
y = 1/2 * (2 L^2 a c s - L^3 s + L R^2 s - (a^2 + r^2) L s + sqrt((s^2 - 1) L^6 - 2 (2 a c s^2 - 3 a c) L^5 - a^6 + 2 a^4 r^2 - a^2 r^4 + ((s^2 - 1) L^2 + 2 L a c - a^2) R^4 - (8 a^2 c^2 - 2 (2 a^2 c^2 + 3 a^2 - r^2) s^2 + 7 a^2 - 2 r^2) L^2 - 2 ((s^2 - 1) L^4 - 2 (a c s^2 - 2 a c) L^3 - a^4 - a^2 r^2 - (4 a^2 c^2 - (a^2 + r^2) s^2 + 2 a^2 + r^2) L^2 + 2 (2 a^3 c + a c r^2) L) R^2 + 2 (3 a^5 c - 4 a^3 c r^2 + a c r^4) L)) / (2 L a c - L^2 - a^2)
\]
extracting the discriminant

```python
# We extract the discriminant of the solution for y:
y0 = ysol[0].rhs(); print y0
print y0.operator(); print y0.operands()
s = str(y0.operands()[1]); print 's =', s
L = s.split('sqrt'); print 'L =', L
d = L[1]; print 'discriminant (as string) =', d
```

produces as final output:

discriminant (as string) = 

```plaintext
((s^2 - 1)*L^6 - 2*(2*a*c*s^2 - 3*a*c)*L^5 - a^6 + 2*a^4*r^2 - a^2*r^4 + ((s^2 - 1)*L^2 + 2*L*a*c - a^2)*R^4 - (8*a^2*c^2 - 2*(2*a^2*c^2 + 3*a^2 - r^2))*L^4 + 4*(5*a^3*c - 2*a*c*r^2 - (3*a^3*c - a*c*r^2)*s^2)*L^3 - (8*a^4*c^2 + 7*a^4 + r^4 - 4*(2*a^2*c^2 + a^2)*r^2 - (5*a^4 - 2*a^2*r^2 + r^4)*s^2)*L^2 - 2*a*c)*L^3 - a^4 - a^2*r^2 - (4*a^2*c^2 - (a^2 + r^2)*s^2 + 2*a^2 + r^2)*L^2 + 2*(2*a^3*c + a*c*r^2)*L)*R^2 + 2*(3*a^5*c - 4*a^3*c*r^2 + a*c*r^4)*L)
```
In addition to the online manual, there is *A Singular Introduction to Commutative Algebra* by Gert-Martin Greuel and Gerhard Pfister, 2nd edition, Springer-Verlag, 2008.

1. Derive the formulas for the 4-bar coupler point in a Singular Terminal session, that is: do all the processing of the Gröbner basis inside a proper Singular session.

2. Instead of a lexicographic Gröbner basis for the system of the coupler point, could you apply resultants to obtain the formulas for the coordinates?

Third Project due Wednesday 28 November at 10:00AM.