Example: SCSCP client in Python3 calculates Groebner basis with Singular

Python users needing an implementation of the Groebner basis algorithm may use SymPy (http://www.sympy.org/ (http://www.sympy.org/)) - a symbolic computation library that, among other features, contains a polynomial manupulation module.

In this example we demonstrate an alternative and much faster approach, which first uses SymPy to create multivariate polynomials, and calls GAP SCSCP server to pass them to Singular.

Because SymPy is presently unable to encode/decode polynomias in OpenMath, this requires desigining remote procedures and their calls to pass external representations of these polynomials in the form of lists of integers, which both systems support, demonstrating the flexibilty of our approach.

In [1]:
```
import sympy
```

In [2]:
```
from sympy.polys import ring, ZZ, QQ
```

In [3]:
```
from scscp import SCSCPCLI
```

- Create a multivariate polynomial

In [4]:
```
R, x, y, z = ring("x, y, z", ZZ)
```

In [5]:
```
f = x*y*z+y**2*z+x**2*z+1
```

In [6]:
```
f
```

Out[6]:
```
x**2*z + x*y*z + y**2*z + 1
```

- Presently SymPy does not implement OpenMath support for polynomials, so we will be passing their external representation instead. The following lists describe monomials and corresponding coefficients.

In [7]:
```
coeffs = f.coeffs()
```

In [8]:
```
coeffs
```

Out[8]:
```
[1, 1, 1, 1]
```

In [9]:
```
mons = [ list(x) for x in f.monoms() ]
```

In [10]:
```
mons
```

Out[10]:
```
[[2, 0, 1], [1, 1, 1], [0, 2, 1], [0, 0, 0]]
```

- We will need the following two functions for conversion between SymPy polynomials and their external representations

In [11]:
```
def ext_rep_poly(f):
    return [ f.coeffs(), [ list(x) for x in f.monoms() ] ]
```

In [12]:
```
from numpy import prod
def construct_poly(R,extrep):
    g = R.gens
    coeffs = extrep[0]
    mons = extrep[1]
    return sum ( [ coeffs[m]*prod([g[i]**mons[m][i] for i in
range(len(g))]) for m in range(len(mons))] )
```

- Obviously, the following condition should always hold

In [13]:
```
g = construct_poly(R,ext_rep_poly(f))
```

In [14]:
```
f == g
```

Out[14]:
```
True
```

- Similar functions for conversion between GAP polynomials and their external representation (as produced by SymPy) have been defined on the GAP SCSCP server. Let's test that we can send polynomials back and forth using the "Ping-Pong" test which encodes and decodes each polynomial twice - on the SymPy's side and on the GAP's side.

In [15]:
```
c = SCSCPCLI('localhost')
```

In [16]:
```
f == construct_poly(R, c.heads.scscp_transient_1.PingPongPoly([ ext_rep_
poly(f)]))
```

Out[16]:
```
True
```

In [17]:
```
c.quit()
```

- Now we show a small example of a Groebner basis computation with SymPy

In [18]:
```
R, x0, x1, x2, x3 = ring("x0, x1, x2, x3", ZZ)
```

In [19]:
```
f1=x0+x1+x2+x3
f2=x0*x1+x1*x2+x0*x3+x2*x3
f3=x0*x1*x2+x0*x1*x3+x0*x2*x3+x1*x2*x3
f4=x0*x1*x2*x3-1
```

In [20]:
```
time(sympy.polys.groebnertools.groebner([f1,f2,f3,f4],R))
```

```
CPU times: user 9.42 ms, sys: 473 µs, total: 9.89 ms
Wall time: 9.64 ms
```

Out[20]:
```
[x0 + x1 + x2 + x3,
 x1**2 + 2*x1*x3 + x3**2,
 x1*x2 - x1*x3 + x2**2*x3**4 + x2*x3 - 2*x3**2,
 x1*x3**4 - x1 + x3**5 - x3,
 x2**3*x3**2 + x2**2*x3**3 - x2 - x3,
 x2**2*x3**6 - x2**2*x3**2 - x3**4 + 1]
```

- To calculate it remotely, first we start new SCSCP session

In [21]:
```
c = SCSCPCLI('localhost')
```

- Just another check for passing polynomials around

In [22]:
```
all( t == construct_poly(R, c.heads.scscp_transient_1.PingPongPoly([ ext
_rep_poly(t)])) for t in [f1,f2,f3,f4] )
```

Out[22]:
```
True
```

- Now call the remote procedure `GroebnerBasisWithSingular` with polynomials from the example above

In [23]:
```
bas = c.heads.scscp_transient_1.GroebnerBasisWithSingular( [ [ ext_rep_p
oly(x) for x in [f1,f2,f3,f4] ] ] )
```

- The result came in external representation, so we have to convert it to SymPy polynomials

In [24]:
```
[ construct_poly(R,t) for t in bas ]
```

Out[24]:
```
[x0 + x1 + x2 + x3,
 x1**2 + 2*x1*x3 + x3**2,
 x1*x2**2 - x1*x3**2 + x2**2*x3 - x3**3,
 x1*x2*x3**2 - x1*x3**3 + x2**2*x3**2 + x2*x3**3 - x3**4 - 1,
 x1*x3**4 - x1 + x3**5 - x3,
 x2**3*x3**2 + x2**2*x3**3 - x2 - x3,
 x1*x2 - x1*x3 + x2**2*x3**4 + x2*x3 - 2*x3**2]
```

- Finally, close SCSCP session

In [25]:
```
c.quit()
```

- Now we present an example when remote calculation with Singular is much faster than local calculation with SimPy

In [26]:
```
R, x0, x1, x2, x3, x4 = ring("x0, x1, x2, x3, x4", ZZ)
```

In [27]:
```
f1=x0+x1+x2+x3+x4
f2=x0*x1+x1*x2+x2*x3+x0*x4+x3*x4
f3=x0*x1*x2+x1*x2*x3+x0*x1*x4+x0*x3*x4+x2*x3*x4
f4=x0*x1*x2*x3+x0*x1*x2*x4+x0*x1*x3*x4+x0*x2*x3*x4+x1*x2*x3*x4
f5=x0*x1*x2*x3*x4-1
```

- Local calculation with SymPy takes about 2 minutes

In [28]:
```
time(sympy.polys.groebnertools.groebner([f1,f2,f3,f4,f5],R))
```

```
CPU times: user 1min 56s, sys: 915 ms, total: 1min 57s
Wall time: 1min 59s
```

Out[28]:
```
[x0 + x1 + x2 + x3 + x4,
 275*x1**2 + 825*x1*x4 + 550*x3**6*x4 + 1650*x3**5*x4**2 + 275*x3**4*x
4**3 - 550*x3**3*x4**4 + 275*x3**2 - 566*x3*x4**11 - 69003*x3*x4**6 +
 69019*x3*x4 - 1467*x4**12 - 178981*x4**7 + 179073*x4**2,
 275*x1*x2 - 275*x1*x4 + 275*x2**2 + 550*x2*x4 - 330*x3**6*x4 - 1045*x
3**5*x4**2 - 275*x3**4*x4**3 + 275*x3**3*x4**4 - 550*x3**2 + 334*x3*x4
**11 + 40722*x3*x4**6 - 40726*x3*x4 + 867*x4**12 + 105776*x4**7 - 1058
73*x4**2,
 275*x1*x3 - 275*x1*x4 - 110*x3**6*x4 - 440*x3**5*x4**2 - 275*x3**4*x4
**3 + 275*x3**3*x4**4 + 124*x3*x4**11 + 15092*x3*x4**6 - 15106*x3*x4 +
346*x4**12 + 42218*x4**7 - 42124*x4**2,
 55*x1*x4**5 - 55*x1 + x4**11 + 143*x4**6 - 144*x4,
 275*x2**3 + 550*x2**2*x4 - 550*x2*x4**2 + 275*x3**6*x4**2 + 550*x3**5
*x4**3 - 550*x3**4*x4**4 + 550*x3**2*x4 - 232*x3*x4**12 - 28336*x3*x4*
*7 + 28018*x3*x4**2 - 568*x4**13 - 69289*x4**8 + 69307*x4**3,
 275*x2*x3 - 275*x2*x4 + 440*x3**6*x4 + 1210*x3**5*x4**2 - 275*x3**3*x
4**4 + 275*x3**2 - 442*x3*x4**11 - 53911*x3*x4**6 + 53913*x3*x4 - 1121
*x4**12 - 136763*x4**7 + 136674*x4**2,
 55*x2*x4**5 - 55*x2 + x4**11 + 143*x4**6 - 144*x4,
 55*x3**7 + 165*x3**6*x4 + 55*x3**5*x4**2 - 55*x3**2 - 398*x3*x4**11 -
48554*x3*x4**6 + 48787*x3*x4 - 1042*x4**12 - 127116*x4**7 + 128103*x4*
*2,
 55*x3**2*x4**5 - 55*x3**2 - 2*x3*x4**11 - 231*x3*x4**6 + 233*x3*x4 -
 8*x4**12 - 979*x4**7 + 987*x4**2,
 x4**15 + 122*x4**10 - 122*x4**5 - 1]
```

- But remote calculation with Singular takes about 6 seconds

In [29]:
```
c = SCSCPCLI('localhost')
```

In [30]:
```
all( t == construct_poly(R, c.heads.scscp_transient_1.PingPongPoly([ ext
_rep_poly(t)])) for t in [f1,f2,f3,f4,f5] )
```

Out[30]:
```
True
```

In [31]:
```
time( [ construct_poly(R,t) for t in \
       c.heads.scscp_transient_1.GroebnerBasisWithSingular( [ [ ext_rep_
poly(x) for x in [f1,f2,f3,f4,f5] ] ] ) ] )
```

```
CPU times: user 5.85 s, sys: 21.9 ms, total: 5.87 s
Wall time: 6.01 s
```

Out[31]:

```
[x0 + x1 + x2 + x3 + x4,
 x1**2 + x1*x3 + 2*x1*x4 - x2*x3 + x2*x4 + x4**2,
 x1*x2*x3 - 2*x1*x3**2 - 2*x1*x3*x4 + 3*x1*x4**2 + x2**3 + 3*x2**2*x4
 - x2*x3**2 - 2*x2*x3*x4 + 3*x2*x4**2 - x3**3 - 3*x3**2*x4 - 2*x3*x4**
2 + 2*x4**3,
 x1*x2**2 - x1*x2*x3 + x1*x3*x4 - x1*x4**2 + x2**2*x3 - x2**2*x4 + x2*
x3*x4 - 2*x2*x4**2 + x3**2*x4 + x3*x4**2 - x4**3,
 14*x1*x2*x3*x4 - x1*x2*x4**2 - 27*x1*x3**2*x4 - 10*x1*x3*x4**2 + 24*x
1*x4**3 + 6*x2**2*x3*x4 + 7*x2**2*x4**2 + 2*x2*x3**2*x4 - 9*x2*x3*x4**
2 + 33*x2*x4**3 + x3**4 - 15*x3**3*x4 - 33*x3**2*x4**2 - 14*x3*x4**3 +
22*x4**4,
 32*x1*x2*x3*x4 - 24*x1*x2*x4**2 - 40*x1*x3**2*x4 - 12*x1*x3*x4**2 + 4
4*x1*x4**3 + 11*x2**2*x3*x4 - 3*x2**2*x4**2 + 19*x2*x3**3 + 10*x2*x3**
2*x4 - 45*x2*x3*x4**2 + 32*x2*x4**3 + 5*x3**4 + x3**3*x4 - 32*x3**2*x4
**2 - 32*x3*x4**3 + 34*x4**4,
 3*x1*x2*x3*x4 - 4*x1*x2*x4**2 + x1*x3**3 - 2*x1*x3**2*x4 - 2*x1*x3*x4
**2 + 4*x1*x4**3 + x2**2*x3*x4 - 2*x2**2*x4**2 + 3*x2*x3**3 + 3*x2*x3*
*2*x4 - 7*x2*x3*x4**2 - x2*x4**3 + x3**4 + 3*x3**3*x4 + x3**2*x4**2 -
 4*x3*x4**3 + 2*x4**4,
 -x1*x2*x3*x4 + 2*x1*x2*x4**2 - 2*x1*x3**3 + 2*x1*x3*x4**2 - x1*x4**3
 + x2**2*x3**2 - x2**2*x3*x4 + x2**2*x4**2 - x2*x3**3 - 2*x2*x3**2*x4
 + 3*x2*x3*x4**2 + 2*x2*x4**3 - x3**4 - 2*x3**3*x4 - 2*x3**2*x4**2 + 2
*x3*x4**3,
 x1*x2*x3**2 + x1*x2*x3*x4 - x1*x2*x4**2 - x1*x3**2*x4 - x1*x3*x4**2 +
x1*x4**3 + x2**2*x3*x4 + x2*x3**2*x4 + x2*x4**3 - x3**3*x4 - 2*x3**2*x
4**2 - x3*x4**3 + x4**4,
 2*x1*x2*x3*x4**2 - x1*x2*x4**3 - 2*x1*x3*x4**3 + x1*x4**4 + x2**2*x3*
x4**2 + 2*x2*x3**2*x4**2 - x2*x3*x4**3 + x2*x4**4 - x3**2*x4**3 - 2*x3
*x4**4 + x4**5 - 1,
 x1*x4**5 - x1 - x2*x4**5 + x2,
 -20*x1*x2*x4**4 + 5*x1*x3*x4**4 + 15*x1 - 20*x2**2*x4**4 + 15*x2*x3**
2*x4**3 - 25*x2*x3*x4**4 - 23*x2*x4**5 - 7*x2 + 10*x3**3*x4**3 + 30*x3
**2*x4**4 - 3*x3*x4**5 + 3*x3 - 4*x4**6 + 24*x4,
 -3*x1*x2*x4**4 + 11*x1*x3**2*x4**3 - 2*x1*x3*x4**4 - 6*x1 - 3*x2**2*x
4**4 + 5*x2*x3**2*x4**3 - x2*x3*x4**4 - 15*x2*x4**5 + 5*x2 + 7*x3**3*x
4**3 + 10*x3**2*x4**4 - x3*x4**5 + x3 - 5*x4**6 - 3*x4,
 2*x2*x3*x4**5 - 2*x2*x3 + 8*x2*x4**6 - 8*x2*x4 + x3**2*x4**5 - x3**2
 + x3*x4**6 - x3*x4 + 3*x4**7 - 3*x4**2,
 3*x2**2*x4**5 - 3*x2**2 - 2*x2*x3*x4**5 + 2*x2*x3 + x2*x4**6 - x2*x4
 - x3**2*x4**5 + x3**2 - x3*x4**6 + x3*x4,
 -9*x1*x2*x4**5 - x1*x2 + 11*x1*x3*x4**5 - x1*x3 - 3*x1*x4**6 + 3*x1*x
4 - 6*x2**2*x4**5 - 4*x2**2 - 5*x2*x3*x4**5 - 9*x2*x4**6 - 6*x2*x4 + 5
*x3**3*x4**4 + 14*x3**2*x4**5 + x3**2 + 5*x3*x4**6 - 5*x3*x4 - 3*x4**7
+ 13*x4**2,
 42*x1*x2*x3 - 76*x1*x2*x4 - 165*x1*x3**2 + 13*x1*x3*x4 + 186*x1*x4**2
+ 21*x2**2*x3 - 55*x2**2*x4 + 42*x2*x3**2 - 131*x2*x3*x4 + 21*x2*x4**2
- 55*x3**3 - 21*x3**2*x4 - 42*x3*x4**2 + x4**8 + 219*x4**3,
 -110*x1*x2*x3 + 29*x1*x2*x4 + 52*x1*x3**2 - 34*x1*x3*x4 + 63*x1*x4**2
- 55*x2**2*x3 - 26*x2**2*x4 + 60*x2*x3**2 - 102*x2*x3*x4 - 120*x2*x4**
2 + 39*x3**3 + 120*x3**2*x4 + x3*x4**7 + 109*x3*x4**2 - 26*x4**3,
 -112*x1*x2*x3 + 33*x1*x2*x4 + 61*x1*x3**2 - 35*x1*x3*x4 + 53*x1*x4**2
- 56*x2**2*x3 - 23*x2**2*x4 + 58*x2*x3**2 - 95*x2*x3*x4 + 8*x2*x4**7 -
129*x2*x4**2 + 42*x3**3 + 121*x3**2*x4 + x3*x4**7 + 111*x3*x4**2 + 3*x
4**8 - 41*x4**3,
 36*x1*x2*x3 - 11*x1*x2*x4 - 37*x1*x3**2 - 7*x1*x3*x4 + 19*x1*x4**2 +
 8*x2**3 + 14*x2**2*x3 + 27*x2**2*x4 - 20*x2*x3**2 + x2*x3*x4 + 53*x2*
x4**2 - 20*x3**3 + x3**2*x4**6 - 54*x3**2*x4 - 44*x3*x4**2 + 34*x4**3]
```

In [32]:

```
c.quit()
```