CoCoALib - Slug #725

Example database: Slow ideal equality test

06 Jun 2015 13:14 - John Abbott

Status:	New	**Start date:**	06 Jun 2015
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:	Various	**Estimated time:**	0.00 hour
Target version:	CoCoALib-1.0	**Spent time:**	1.05 hour

Description

This is probably not important, but I wanted to record this example which seems to be unexpectedly slow -- it might be useful for future testing/tuning.

The example comes from a study of symplectic matrix groups (JAA+Eick). I just wanted to verify that the set of generators I have is minimal (*i.e.* non of the generators is redundant). There are 12 generators to test for redundancy; the 10th leads to a much slower computation than the rest (more time than all the rest put together, I believe).

Ideally I want to do the computation over QQ, but I actually did it over ZZ/(3) hoping it would be faster.

```
n := 3; // almost instant with parameter n=2

use P ::= QQ[x[1..2*n, 1..2*n]];
use P ::= ZZ/(3)[x[1..2*n, 1..2*n]];

M := mat([[x[i,j] | i in 1..2*n ] | j in 1..2*n]);

Id := IdentityMat(P,n);
Z := mat(P,[[0 | i in 1..n] | j in 1..n]);
omega := BlockMat([[Z,Id],[-Id,Z]]);
CondMat := transposed(M)*omega*M - omega;

G := [];
for i:=1 to 2*n do
  for j:=i+1 to 2*n do
    append(ref G, CondMat[i,j]);
  endfor;
endfor;
I := ideal(G);

-- This loop is very slow when j=10
for j:=1 to len(G) do
  t0 := CpuTime();
  test := ideal(WithoutNth(G,j)) = I;
  println "test for j=",j,"  result=",test,"  time=",TimeFrom(t0);
endfor;
```

Related issues:

Related to CoCoALib - Feature #1267: Ideal equality	**In Progress**	**28 Mar 2019**

History

#1 - 06 Jun 2015 13:19 - John Abbott

Re-running the program (over QQ) I get the following output (still incomplete):

```
test for j=1  result=false  time=10.481
test for j=2  result=false  time=48.799
test for j=3  result=false  time=23.532
test for j=4  result=false  time=6.957
test for j=5  result=false  time=1.681
test for j=6  result=false  time=45.491
```

```
...
```

So it is clear that there is considerable variability in time even among the "fast" cases. The case j=7 is quite slow, but j=10 seemed to be much slower...

I suppose the main cost is computing a G-basis for the ideal with one generator removed.

#2 - 06 Jun 2015 14:25 - John Abbott

Progress...

```
test for j=7  result=false  time=1805.911
test for j=8  result=false  time=18.879
test for j=9  result=false  time=5.747
```

Note: j=10 took 8400s over ZZ/(3) -- now my computer's rather warm :-/

#3 - 06 Jun 2015 20:32 - John Abbott

It has finally finished:

```
test for j=10  result=false  time=19217.141
test for j=11  result=false  time=4650.939
test for j=12  result=false  time=31.482
test for j=13  result=false  time=30.232
test for j=14  result=false  time=14.689
test for j=15  result=false  time=59.276
```

These times are for computation over QQ.

#4 - 06 Jun 2015 21:17 - Anna Maria Bigatti

Instead of checking equality you should just check whether g_j is in the ideal generated by the others. However, this will not make a big difference in speed. The input is not homogeneous, is it?

#5 - 06 Jun 2015 22:46 - John Abbott

Anna, you are right (in principle). In practice, I think almost the whole time is spent computing a G-basis (rather than using the G-basis for computing NFs).

The input is "not quite homogeneous"; several generators are homogeneous (deg=2), but some are 1+homog(deg=2).

I think my main point is to note this example as a test case for future improvements to the GBMill.

#6 - 06 Jun 2015 22:49 - John Abbott

For completeness here is the list of generators in QQ[x[1..6,1..6]]

```
[
  -x[1,4]*x[2,1] -x[1,5]*x[2,2] -x[1,6]*x[2,3] +x[1,1]*x[2,4] +x[1,2]*x[2,5] +x[1,3]*x[2,6],
  -x[1,4]*x[3,1] -x[1,5]*x[3,2] -x[1,6]*x[3,3] +x[1,1]*x[3,4] +x[1,2]*x[3,5] +x[1,3]*x[3,6],
  -x[1,4]*x[4,1] -x[1,5]*x[4,2] -x[1,6]*x[4,3] +x[1,1]*x[4,4] +x[1,2]*x[4,5] +x[1,3]*x[4,6] -1,
  -x[1,4]*x[5,1] -x[1,5]*x[5,2] -x[1,6]*x[5,3] +x[1,1]*x[5,4] +x[1,2]*x[5,5] +x[1,3]*x[5,6],
  -x[1,4]*x[6,1] -x[1,5]*x[6,2] -x[1,6]*x[6,3] +x[1,1]*x[6,4] +x[1,2]*x[6,5] +x[1,3]*x[6,6],
  -x[2,4]*x[3,1] -x[2,5]*x[3,2] -x[2,6]*x[3,3] +x[2,1]*x[3,4] +x[2,2]*x[3,5] +x[2,3]*x[3,6],
  -x[2,4]*x[4,1] -x[2,5]*x[4,2] -x[2,6]*x[4,3] +x[2,1]*x[4,4] +x[2,2]*x[4,5] +x[2,3]*x[4,6],
  -x[2,4]*x[5,1] -x[2,5]*x[5,2] -x[2,6]*x[5,3] +x[2,1]*x[5,4] +x[2,2]*x[5,5] +x[2,3]*x[5,6] -1,
  -x[2,4]*x[6,1] -x[2,5]*x[6,2] -x[2,6]*x[6,3] +x[2,1]*x[6,4] +x[2,2]*x[6,5] +x[2,3]*x[6,6],
  -x[3,4]*x[4,1] -x[3,5]*x[4,2] -x[3,6]*x[4,3] +x[3,1]*x[4,4] +x[3,2]*x[4,5] +x[3,3]*x[4,6],
  -x[3,4]*x[5,1] -x[3,5]*x[5,2] -x[3,6]*x[5,3] +x[3,1]*x[5,4] +x[3,2]*x[5,5] +x[3,3]*x[5,6],
  -x[3,4]*x[6,1] -x[3,5]*x[6,2] -x[3,6]*x[6,3] +x[3,1]*x[6,4] +x[3,2]*x[6,5] +x[3,3]*x[6,6] -1,
  -x[4,4]*x[5,1] -x[4,5]*x[5,2] -x[4,6]*x[5,3] +x[4,1]*x[5,4] +x[4,2]*x[5,5] +x[4,3]*x[5,6],
  -x[4,4]*x[6,1] -x[4,5]*x[6,2] -x[4,6]*x[6,3] +x[4,1]*x[6,4] +x[4,2]*x[6,5] +x[4,3]*x[6,6],
  -x[5,4]*x[6,1] -x[5,5]*x[6,2] -x[5,6]*x[6,3] +x[5,1]*x[6,4] +x[5,2]*x[6,5] +x[5,3]*x[6,6]
]
```

#7 - 07 Jun 2015 12:46 - John Abbott

In the specific example of this issue there are 36 indets (all used), 15 polynomials, and dim(P/I) gives 21. Isn't this enough to prove that there are no redundant generators?

If so, perhaps this could be added to MinimalSubsetOfGens?

Note that it is important to know how many indets are actualy being used (see issue #658).

#8 - 10 Jun 2015 14:24 - Anna Maria Bigatti

John Abbott wrote:

> In the specific example of this issue there are 36 indets (all used), 15 polynomials, and dim(P/I) gives 21. Isn't this enough to prove that there are no redundant generators?

yes

> If so, perhaps this could be added to MinimalSubsetOfGens?

hmm, true

#9 - 10 Dec 2023 20:46 - John Abbott

- Related to Feature #1267: Ideal equality added