CoCoALib - Bug #1280

Determinant & Inverse of matrix over non integral domain

03 May 2019 13:51 - John Abbott

Status:	In Progress	**Start date:**	03 May 2019
Priority:	High	**Due date:**	
Assignee:	John Abbott	**% Done:**	40%
Category:	Improving	**Estimated time:**	0.00 hour
Target version:	CoCoALib-0.99880	**Spent time:**	3.05 hours

Description

CoCoA gives unnecessary errors when computing det or inverse of a matrix over a non-integral domain.

Rectify!

Related issues:

Related to CoCoALib - Bug #15: Adjoint of a non-invertible matrix	**Closed**	**28 Oct 2011**
Related to CoCoALib - Design #1279: Tidy up code for matrix determinant	**Closed**	**03 May 2019**

History

#1 - 03 May 2019 13:57 - John Abbott

Here is a simple failing case:

```
ZZ6 ::= ZZ/(6);
M := IdentityMat(ZZ6,6);
det(M);  --> ERROR (not sparse poly ring !??!)
M^(-1);  --> ERROR (not int.dom.)
```

Inverse always fails, but det succeeds up to size 5x5 because small matrices are handled specially.

#2 - 03 May 2019 13:57 - John Abbott

- Related to Bug #15: Adjoint of a non-invertible matrix added

#3 - 03 May 2019 13:58 - John Abbott

- Related to Design #1279: Tidy up code for matrix determinant added

#4 - 03 May 2019 15:59 - John Abbott

- Status changed from New to In Progress

- % Done changed from 0 to 10

Some thoughts about computing the determinant.

There is special case code for:

- small matrices (up to 5x5 incl.)

- certain special matrices (*e.g.* ZeroMat, IdentityMat)
- matrix over a field (*i.e.* gaussian reduction)
- matrix over ZZ

The code is confusing because some special cases are handled "automatically" by C++ inheritance mechanism (*e.g.* ZeroMat and IdentityMat, perhaps also DiagMat),
while other special cases have to be handled separately.

It is also not entirely clear to me what the "general case" should be. The most general is DetDirect (expansion by minors), but it is likely to be very slow for matrices larger than 6x6, say. Fraction-free Bareiss is probably faster whenever it can be used, but there could be problems with zero-divisors. Perhaps the general case should try Bareiss, and if that fails then resort to expansion by minors...

#5 - 03 May 2019 16:35 - John Abbott

- Assignee set to John Abbott

I'd like to rewrite ConstMatrixViewBase::myDet so that it calls the "fast" impls for small matrices. How to do this cleanly?

Presumably the final code should look something like this:

```
if (IsSmallMatrix(M)) return DetOfSmallMatrix(M);
```

The complications are:

- **(A)** the matrix is actually a *ConstMatrixViewBase (via this)
- **(B)** there is no IsSmallMatrix test
- **(C)** there is no function DetOfSmallMatrix

Part **(C)** can easily be resolved, but somehow **(B)** and **(C)** should be solved together. Part **(A)** can be resolved by wrapping the pointer into a ConstMatrixView object (this is a bit "clunky" but should be safe and efficient).

The fictitious IsSmallMatrix function actually depends on the operation to be performed (in this case det), so this should somehow be apparent in the call.

#6 - 03 May 2019 20:36 - John Abbott

I'm now wondering what is the difference between ConstMatrixViewBase::myDet and DenseMatImpl::myDet...

#7 - 08 Jan 2020 22:45 - John Abbott

- Target version changed from CoCoALib-0.99700 to CoCoALib-0.99800

#8 - 06 Oct 2020 12:20 - John Abbott

Now det of a matrix over ZZ/(6) work also for larger matrices, but it is horribly slow :-(

```
ZZ6 ::= ZZ/(6);
M := IdentityMat(ZZ6,11);
det(M); --> takes 22s
```

Why does it not recognise that M is an identity matrix?
In this case we could "speculatively" try gauss (it will work), or alternatively compute det over ZZ and then reduce mod 6...

#9 - 06 Oct 2020 12:21 - John Abbott

- Target version changed from CoCoALib-0.99800 to CoCoALib-0.99850

- % Done changed from 10 to 40

This question clearly needs more work -- postponing.

#10 - 29 Oct 2021 19:13 - John Abbott

Here are some more examples which should work, but which do not:

```
/**/ ZZ15 ::= ZZ/(15);
/**/ M := mat(ZZ15, [[5,3],[3,5]]);
/**/ inverse(M);
--> ERROR: Ring is not an integral domain
--> [CoCoALib] InverseByGauss(Mat)  over non-integral domain
--> inverse(M);
--> ^^^^^^^^^^
/**/ det(M);
1
/**/ M := mat(ZZ15, [[5,3,3],[3,5,3],[3,3,5]]);
/**/ det(M);
-1
/**/ inverse(M);
--> ERROR: Ring is not an integral domain
```

Note that det works here but inverse does not. As observed earlier, it could be tricky deciding how to compute the inverse since all entries are zero-divisors.

#11 - 30 Oct 2021 22:09 - John Abbott

Here is a way of making test cases for computing determinant:
take a unimodular matrix, and multiply it by a nzzd (which is not nilpotent).
So every entry is a zero-divisor, and all linear combinations of rows/cols contain only zero-divisors.

One possible approach is to represent the determinant as a sum of two other determinants,
by representing the [1,1]-element as a sum of two non-zero-divisors; compute recursively
the dets of the two matrices obtained by substituting the two summands in posn [1,1].
A problem is that we would have to repeat this process for each row/col; there would
have to be about 2^n low-level det computations.

I wonder if there is some clever "fraction-free" method.

Another (daft?) idea: if all entries are zero-divisors then try multiplying the matrix
by a random unimodular one. Maybe we will be lucky and get a matrix some of whose entries
are "nice" (*i.e.* not zero-divisors).

Two concrete examples:

```
UniMod := matrix(ZZ, [[1, 1, 3],  [2, 1, 4],  [1, 1, 2]]);
M := 3*matrix(ZZ15, UniMod); // all entries are nzzds
```

Here is a "lucky" multiplication by a unimodular mat:

```
N := mat(ZZ15, [[5,3,3],[3,5,3],[3,3,5]]);
N*mat(ZZ15, UniMod); // UniMod from example above
matrix( ZZ15,
 [[-1, -4, 3],
  [1, -4, 5],
  [-1, -4, 1]])
```

I wonder how often we are lucky?

#12 - 21 Jan 2024 19:43 - John Abbott

- Target version changed from CoCoALib-0.99850 to CoCoALib-0.99880