first-steps-math

April 14, 2025

In this session we will use Macaulay2 to investigate some behavior of a class of ideals.

1 First steps: a homogeneous example ideal

Let's first take a homogeneous ideal and get the most common information about it.

Here we do this with the first key example in algebraic geometry: the ideal of the twised cubic curve!

First we define the ring the ideal will sit in.

[ ]:
```
R = ZZ/32003[a..d]
```

[ ]:
```
I = ideal(a*d-b*c, a*c-b^2, b*d-c^2)
```

Let's compute the dimension, codimension (these two should add to 4, the number of variables), and the degree
(this is geometrically the number of points of a general linear section of the zero set of complementary dimension.

[ ]:
```
dim I
```

[ ]:
```
codim I
```

[ ]:
```
degree I
```

The hilbert function `hilbertFunction(d, I)` is the dimension over the base field of the quotient vector space $(R/I)_3$.

[ ]:
```
hilbertFunction(3, I)
```

The Hilbert series of I (really, of the R-module R/I) is the formal sum

$$H_{R/I}(t) := \sum_{i=0}^{\infty} \dim(R/I)_d t^d.$$

Hilbert invented the notion of the finite free resolution of an idea;/module, to prove that this is a rational function in t.

[ ]:
```
hilbertSeries I
```

Hilbert also proved that if one divides by $1-t$ as much as possible, one obtains a power series $\frac{Q(t)}{(1-t)^m}$, where Q is a polynomial with integer coefficients with $Q(1) \neq 0$, and $m = \dim R/I$ (this is one more that the dimension of the zero set of I as a subvariety of projective space.

`reduceHilbert` does this division: it divides by $(1-t)$ as many times as possible.

[ ]:
```
reduceHilbert oo
```

Thus, $\dim R/I = 2$, and $V(I) \subset \mathbb{P}^3$ is a curve. Hilbert also proved from the above results that `hilbertFunction(d, I)`, for $d \gg 0$ is a polynomial of degree $d-1$, now called the **Hilbert polynomial**.

[ ]:
```
hilbertPolynomial(I, Projective => false)
```

In many ways, the minimal free resolution (unique up to isomorphism, i.e. change of bases) is the most important invariant of an ideal or module.

[ ]:
```
F = res I
```

[ ]:
```
F.dd
```

The degrees in the matrices in the free resolution are homogeneous (if I is). For instance, the first matrix has three generators of degree 2, and two first syzygies of degree 3.

exercise. Look up the documentation and make sure you understand what the `betti` function displays.

[ ]:
```
betti F
```

2 Ideals, matrices, lists, and complexes

You have seen lists, and above we have created ideals and complexes. It is nice to be able to switch from one type to another.

For example, we start with the ideal I above. How do we access the polynomials in it? By indexing, using the underscore operator. **Important Note!** All indexing for Macaulay2 (including indexing into lists, and matrices) is zero based!

[ ]:
```
R = ZZ/32003[a..d]
I = ideal(a*d-b*c, a*c-b^2, b*d-c^2)
```

[ ]:
```
I_0
```

[ ]:
```
numgens I
```

[ ]:
```
for i from 0 to numgens I - 1 list I_i
```

This is so common, here is a faster way to do it:

[ ]:
```
I_*
```

Going back and forth from lists (of polynomials, all in the same ring) to ideals is pretty easy:

[ ]:
```
L = I_*
```

[ ]:
```
ideal L == I
```

[ ]:
```
ideal{a,b,c,d^3-1}
```

The matrix of generators of an ideal: (one can also use `gens I`). Remember that if a function takes one argument it is not necessary to put parentheses around it.

[ ]:
```
generators I
```

A matrix has a bunch of information in it (the source and target modules, here they are free modules, degree information, and of course the entries in the matrix.

Here are some basic things you can do with a matrix.

[ ]:
```
R = QQ[a..i]
```

[ ]:
```
m = matrix{{a,b,c}, {d,e,f}, {g,h,i}}
```

[ ]:
```
numColumns m
```

[ ]:
```
numcols m
```

[ ]:
```
numRows m
```

[ ]:
```
numrows m
```

[ ]:
```
m_(0,0)
```

[ ]:
```
m_(1,3)
```

[ ]:
```
entries m
```

[ ]:
```
flatten entries m
```

[ ]:
```
ideal m
```

[ ]:
```
ideal m == ideal flatten entries m
```

We can make modules out of matrices too.

[ ]:
```
image m
```

[ ]:
```
coker m
```

[ ]:
```
kernel m
```

3 First exploration

One cool way to generate interesting ideals is to use Macaulay's inverse systems.
Let $F \in R = \Bbbk[x_1, \dots, x_n]$ be a homogeneous polynomial (one can define this in the local ring case as well, but let's just do the homogeneous case for now). Define an ideal, $F^\perp \subset R$ to be

$$F^\perp := \{g \in R \mid g(\partial_1, \dots, \partial_n)(F) = 0\},$$

where ∂_i is differentiation by x_i.

[ ]:
```
R = ZZ/32003[a..d]
```

[ ]:
```
F = a^3 + b^3 + c^3 + d^3
```

[ ]:
```
I = inverseSystem F
```

[ ]:
```
dim I, codim I, degree I
```

[ ]:
```
C = res I
```

[ ]:
```
betti C
```

Macaulay proved that such ideals $F^\perp$ are Artinian and Gorenstein. Here Artinian means that in high enough degrees d, $(R/I)_d = 0$. Gorenstein for Artinian ideals is equivalent to the condition that the last non-zero free module in the minimal free resolution of R/I is rank one.

We want to investigate: suppose we fix the number of variables n, and the degree d of F. What are the possible Betti tables that we can find?

Let's try: all F which are equal to a monomial.

[ ]:
```
B = flatten entries basis(3, R)
```

[ ]:
```
for b in B list inverseSystem ideal b
```

[ ]:
```
for b in B list betti res inverseSystem ideal b
```

[ ]:
```
tally oo
```

[ ]:
```
apply(100, i -> betti res inverseSystem ideal random(3, R))
```

[ ]:
```
tally oo
```

What else should we try? So far we have 4 different such ideals.

How about F is the sum of two monomials? three? four? What other suggestions to you have to try?

[ ]:
```
for x in subsets(B, 2) list betti res inverseSystem (sum x)
```

[ ]:
```
tally oo
```

[ ]:
```
tally for x in subsets(B, 2) list betti res inverseSystem (sum x)
```

[ ]:
```
netList pack(4, keys oo)
```

[ ]:
```
tally for x in subsets(B, 3) list betti res inverseSystem (sum x)
```

[ ]:
```
netList pack(keys oo, 4)
```

[ ]:
```
o46
```

[ ]:
```
#B
```

[ ]:
```
tally for x in subsets(B, 4) list betti res inverseSystem (sum x)
```

[ ]:
```
--%timeout=360000
```

[ ]:
```
tally for x in subsets(B, 4) list betti res inverseSystem (sum x)
```

[ ]:
```
pack(keys oo, 4)
```

[ ]:
```
randomPolySum = (R, d, m) -> (
    -- R is a polynomial ring
    -- d is an integer, the degree, should be >= 1.
    -- m is an integer: the number of d-th power random linear forms to take
    linears := for i from 0 to m-1 list random(1, R);
    linears
    )

netList randomPolySum(R, 3, 6)
```

[ ]:
```
randomPolySum = (R, d, m) -> (
    -- R is a polynomial ring
    -- d is an integer, the degree, should be >= 1.
    -- m is an integer: the number of d-th power random linear forms to take
    linears := for i from 0 to m-1 list random(1, R);
    sum for g in linears list g^d
    )

randomPolySum(R, 3, 6)
```

[ ]:
```
betti res inverseSystem oo
```

[ ]:
```
tally toList apply(100, i -> betti res inverseSystem randomPolySum(R, 3, 6))
```

[ ]:
```
tally toList apply(1000, i -> betti res inverseSystem randomPolySum(R, 3, 6))
```

[ ]:
```
tally toList apply(100, i -> betti res inverseSystem randomPolySum(R, 3, 5))
```

```
[ ]: tally toList apply(100, i -> betti res inverseSystem randomPolySum(R, 3, 4))
```

```
[ ]: tally toList apply(100, i -> betti res inverseSystem randomPolySum(R, 3, 3))
```

How can we make our life easier in collecting examples? One way: keep a mutable hash table of all of the examples we have found, only add to it if we have a new one.

Here is one way to do this.

```
[ ]: allOurBettis = new MutableHashTable
```

```
[ ]: handleExample = (F) -> (
         bt := betti res inverseSystem F;
         if not allOurBettis#?bt then (
             allOurBettis#bt = F;
             << "found new betti table: " << bt << endl;
             );
         )
```

```
[ ]: R = ZZ/32003[a..d]
```

```
[ ]: scan(1000, i -> handleExample randomPolySum(R, 3, 4))
```

```
[ ]: peek allOurBettis
```

```
[ ]: scan(1000, i -> handleExample randomPolySum(R, 3, 5))
```

```
[ ]: scan(1000, i -> handleExample randomPolySum(R, 3, 4))
```

```
[ ]: scan(1000, i -> handleExample randomPolySum(R, 3, 4))
```

```
[ ]: scan(1000, i -> handleExample randomPolySum(R, 3, 5))
```

```
[ ]: scan(1000, i -> handleExample randomPolySum(R, 3, 5))
```

```
[ ]: scan(1000, i -> handleExample randomPolySum(R, 3, 6))
```

```
[ ]: scan(1000, i -> handleExample randomPolySum(R, 3, 6))
```

```
[ ]: netList pack(keys allOurBettis, 4)
```

4 Exercises

- Write a function which takes a Betti table as input and a degree and homological degree (basically the row and column), and returns that Betti number.
- Consider ideals minimally generated by 4 quadratic homogeneous polynomials. How many Betti tables can you find for such ideals?
- Consider ideals of the form $F^{\perp}$, where F is a homogeneous polynomial in 4 variables, of degree 4. How many Betti tables can you find?

[ ]: