# C. Using Macaulay 2

by **David Eisenbud,**
**Daniel R. Grayson**
**and Michael E. Stillman**

*Macaulay 2* is a computer software package devoted to supporting research in algebraic geometry and commutative algebra. It was written in the years 1993-1997 by Daniel R. Grayson and Michael E. Stillman with funding from the National Science Foundation and includes subroutine packages for factoring contributed by Gert-Martin Greuel, Ruediger Stobbe and Michael Messollen. In this appendix we will illustrate some of the concepts presented in this book through computations accomplished with *Macaulay 2*.

Copies of *Macaulay 2* for use in educational institutions may be obtained from the web site at `http://www.math.uiuc.edu/Macaulay2`. The source code is available there, as well as versions already compiled for various types of computers and operating systems, including: Linux, Windows 95, or NeXTstep on a PC; Hewlett Packard workstations with HP-UX; Sun workstations with Solaris or SunOS; IRIS workstations with IRIX; and Silicon Graphics workstations. The version for the Macintosh PowerPC should be available soon.

The current version of *Macaulay 2* at the time of writing is 0.8.17, indicating that the design of the language is not in its final state; up-to-date versions of this appendix will be included in distributions of *Macaulay 2*.

**Contents**

a. Mathematical background
b. Macaulay2 routines
c. Example: the Hodge diamond of a K3 surface

## C.1 Elementary uses of Macaulay 2

In this section we introduce a number of basic operations using Gröbner bases, and at the same time become familiar with a range of useful *Macaulay 2* constructs.

## C.1.a. First steps

Start Macaulay 2 with the command `M2`, and you will be presented with "`i1 :`" as input prompt. An expression entered at the keyboard will be evaluated – no punctuation is required at the end of the line.

```
i1 : 2+2

o1 = 4
```

The answer, 4, is displayed after the output label "`o1 =`". Multiplication is indicated with the traditional `*`.

```
i2 : 1*2*3*4

o2 = 24
```

Powers are obtained as follows.

```
i3 : 2^100

o3 = 1267650600228229401496703205376
```

Because some answers can be very long, it is a good idea to run the program in a window which does not wrap output lines, and allows the user to scroll horizontally to see the rest of the output.

```
i4 : 100!

o4 = 933262154439441526816992388562667004907159682643816214685929 ...
```

Here is some arithmetic with fractions.

```
i5 : 3/5 + 7/11

     68
o5 = --
     55

o5 : QQ
```

Notice how the system displays the type of thing the output value is on an additional output line: `QQ`. (We reserve single letter symbols such as `Q` for use as variables in rings. Hence we must use `QQ` to stand for the field of rational numbers; it may remind you of the "blackboard bold" font of AMSTeX.

It is possible to obtain integer quotients and remainders with the following operators.

```
i6 : 1234//100

o6 = 12

i7 : 1234%100
```

```
o7 = 34
```

The operator for equality testing is a double equal sign.

```
i8 : 2+2==4

o8 = true
```

Multiple expressions may be separated by semicolons.

```
i9 : 1;2;3*4

o11 = 12
```

A semicolon at the end of the line suppresses the printing of the value.

```
i12 : 4*5;
```

The output from the previous line can be obtained with `oo`, even if a semicolon prevented it from being printed.

```
i13 : oo

o13 = 20
```

Lines before that can be obtained with `ooo` and `oooo`. Alternatively, the symbol labeling an output line can be used to retrieve the value.

```
i14 : o11 + 1

o14 = 13
```

A list of expressions can be formed with braces.

```
i15 : {1, 2, s}

o15 = {1, 2, s}

o15 : List
```

A function can be created with the arrow operator.

```
i16 : cube = i -> i^3

o16 = cube

o16 : Function
```

To evaluate a function, place its argument to the right of the function.

```
i17 : cube 5

o17 = 125
```

Functions of more than one variable take a parenthesized sequence of arguments.

```
i18 : mult = (x,y) -> x * y

o18 = mult

o18 : Function

i19 : mult(6,9)

o19 = 54
```

The function `apply` can be used to apply a function to each element of a list.

```
i20 : apply({1,2,3,4}, cube)

o20 = {1, 8, 27, 64}
```

```
o20 : List
```

The function `scan` is analogous to `apply` except that no value is returned. It may be used to implement loops in programs.

```
i21 : scan({1,2,3,4}, i -> print (i, i^3))
(1, 1)
(2, 8)
(3, 27)
(4, 64)
```

Most computations with polynomials take place in rings that may be specified in usual mathematical notation.

```
i22 : R = ZZ/5[x,y,z]

o22 = R

o22 : PolynomialRing
```

Here `ZZ` refers to the ring of integers.

```
i23 : (x+y)^5

       5    5
o23 = x  + y

o23 : R
```

A free module can be created as follows.

```
i24 : F = R^3

       3
o24 = R

o24 : R - module, free
```

A list of indices can be used to produce the homomorphism corresponding to the corresponding basis vectors.

```
i25 : F_{1,2}

o25 = {0} | 0 0 |
      {0} | 1 0 |
      {0} | 0 1 |

              3       2
o25 : Matrix R  <--- R
```

We can create a homomorphism between free modules with `matrix` by providing a list of the rows of the matrix, each of which is in turn a list of ring elements.

```
i26 : ff = matrix {{x,y,z}}

o26 = {0} | x y z |

              1       3
o26 : Matrix R  <--- R
```

Use `image` to get the image.

```
i27 : image ff

o27 = image {0} | x y z |
```

```
                                    1
o27 : R - module, submodule of R
```

The corresponding ideal can be obtained with `ideal`.

```
i28 : ideal (x,y,z)

o28 = ideal (x, y, z)

o28 : Ideal of R
```

We may use `kernel` to compute the kernel of `ff`.

```
i29 : kernel ff

o29 = image {1} | 0  -y -z |
            {1} | -z x  0  |
            {1} | y  0  x  |

                                    3
o29 : R - module, submodule of R
```

The answer comes out as a module which is expressed as the image of a homomorphism whose matrix is displayed. In case the matrix itself is desired, it can be obtained with `generators`.

```
i30 : generators oo

o30 = {1} | 0  -y -z |
      {1} | -z x  0  |
      {1} | y  0  x  |

              3       3
o30 : Matrix R  <--- R
```

We may use `rank` to compute the rank.

```
i31 : rank kernel ff

o31 = 2
```

A presentation for the kernel can be obtained with `presentation`.

```
i32 : presentation kernel ff

o32 = {2} | x  |
      {2} | z  |
      {2} | -y |

              3       1
o32 : Matrix R  <--- R
```

We can produce the cokernel with `cokernel`.

```
i33 : cokernel ff

o33 = cokernel {0} | x y z |

                                    1
o33 : R - module, quotient of R
```

The direct sum is formed as follows.

```
i34 : N = kernel ff ++ cokernel ff

o34 = subquotient ({1} | 0  -y -z 0 |, {1} | 0 0 0 |)
                  {1} | -z x  0  0 |  {1} | 0 0 0 |
                  {1} | y  0  x  0 |  {1} | 0 0 0 |
                  {0} | 0  0  0  1 |  {0} | x y z |
```

```
                                      4
o34 : R - module, subquotient of R
```

The answer is expressed in terms of the `subquotient` function, which produces subquotient modules. Each subquotient module is accompanied by its matrix of generators and its matrix of relations. These matrices can be recovered with the functions `generators` and `relations`.

```
i35 : generators N

o35 = {1} | 0  -y -z 0 |
      {1} | -z x  0  0 |
      {1} | y  0  x  0 |
      {0} | 0  0  0  1 |

               4       4
o35 : Matrix R  <--- R

i36 : relations N

o36 = {1} | 0 0 0 |
      {1} | 0 0 0 |
      {1} | 0 0 0 |
      {0} | x y z |

               4       3
o36 : Matrix R  <--- R
```

The function `prune` can be used to convert a subquotient module to a quotient module.

```
i37 : prune N

o37 = cokernel {2} | 0 0 0 -y |
               {2} | 0 0 0 z  |
               {2} | 0 0 0 x  |
               {0} | z y x 0  |

                                     4
o37 : R - module, quotient of R
```

To compute the Gröbner basis of the ideal $(x^2, xy+y^2)$ we proceed as follows. We form the ideal:

```
i38 : I = ideal(x^2,x*y+y^2)

              2         2
o38 = ideal (x , x*y + y )

o38 : Ideal of R

i39 : p = generators gb I

o39 = {0} | xy+y2 x2 y3 |

              1       3
o39 : Matrix R  <--- R
```

From this result we can for example compute the codimension, dimension, degree, and the whole Hilbert function and polynomial.

As a parenthetical remark, we note that *Macaulay 2* tends to display a matrix in a compact horizontal format in which the exponents are written to the right of the corresponding variables. Hence `3xy2+y3z` would represent $3xy^2 + y^3z$. Here is a way to see the matrix above in a more legible two-dimensional form.

```
i40 : expression p

      |         2   2   3 |
o40 = |  x*y + y   x   y  |

o40 : MatrixExpression
```

## C.1.b. A monomial curve example

This will be more fun if we work with an example having some meaning. We choose to work with the ideal defining the rational quartic curve in $\mathbf{P}^3$ given parametrically in an affine representation by

$$t \mapsto (t, t^3, t^4).$$

(The reader who doesn't understand this terminology may ignore it for the moment, and treat the ideal given below as a gift from the gods.) First we introduce our favorite field.

```
i41 : KK = ZZ/31991;
```

We obtain the ideal by first making the polynomial ring in 4 variables (the homogeneous coordinate ring of $\mathbf{P}^3$).

```
i42 : R = KK[a..d];
```

Then we use the *Macaulay 2* function `monomialCurve`, which we shall treat for now as a black box.

```
i43 : I = monomialCurve(R,{1,3,4})

                          3      2     2    2    3    2
o43 = ideal (b*c - a*d, c  - b*d , a*c  - b d, b  - a c)

o43 : Ideal of R
```

The codimension of the ideal $I$ is 2, and its dimension is 2.

```
i44 : codim I

o44 = 2

i45 : dim I

o45 = 2
```

If we wanted to regard $I$ as a module instead of an ideal, and take the codimension of the support of that module, we would write

```
i46 : codim module I

o46 = 0
```

The dimension of I as an ideal is by definition the same as the dimension of the module $R/I$. If we simply write $R/I$, *Macaulay 2* will interpret it as a ring, so we write $R^1$ for the free module of rank 1 over $R$, and the dimension may be computed again as

```
i47 : dim (R^1/I)

o47 = 2
```

We could also make the generators of $I$ into a matrix and take the cokernel to get the same thing:

```
i48 : M = coker generators I
```

```
o48 = cokernel {0} | bc-ad c3-bd2 ac2-b2d b3-a2c |

                                     1
o48 : R - module, quotient of R

i49 : codim M

o49 = 2

i50 : dim M

o50 = 2
```

And similarly for the degree:

```
i51 : degree I

o51 = 4

i52 : degree M

o52 = 4
```

As one might expect, the degree of the quartic is 4! The Hilbert polynomial may be obtained as

```
i53 : hilbertPolynomial( M, Projective=>false)

o53 = 4$i + 1

o53 : QQ[$i]
```

Here `Projective=>false` is an "option" that tells *Macaulay 2* to display the Hilbert polynomial as an ordinary polynomial. We see from the polynomial that we are dealing with a curve of degree 4 and genus 0. The default display would show the polynomial as a linear combination of the Hilbert polynomials of projective spaces of different dimensions:

```
i54 : hilbertPolynomial M

o54 = - 3*P  + 4*P
           0      1

o54 : ProjectiveHilbertPolynomial
```

Similarly, we can display the Hilbert series as a rational function:

```
i55 : hilbertSeries M

          3      2
      - $T  + 2$T  + 2$T + 1
o55 = ----------------------
                        2
             (- $T + 1)

o55 : Divide
```

Another way to get information about the module M is to see its free resolution

```
i56 : Mres = res M

       1      4      4      1
o56 = R  <-- R  <-- R  <-- R

      0      1      2      3
```

```
o56 : ChainComplex
```

To get more precise information about `Mres`, we could do

```
i57 : betti Mres
    total: 1 4 4 1
        0: 1 . . .
        1: . 1 . .
        2: . 3 4 1
```

The display is chosen for compactness: the first line gives the total Betti numbers, the same information given when we type the resolution. The remaining lines express the degrees of each of the generators of the free modules in the resolution. The $j$-th column after the colons (counting from 0) gives the degrees of generators of the $j$-th module; an $n$ in the $j$-th column in the row headed by $d$ means that the $j$-th free module has $n$ generators of degree $d + j$. Thus for example in our case, the generator of the third free module in the resolution has degree $3 + 2 = 5$.

## C.1.c. Random regular sequences

An interesting and illustrative open problem is to understand the initial ideal (and the Gröbner basis) of a "generic" regular sequence. To study a very simple case we take a matrix of 2 random forms in a polynomial ring in 3 variables.

```
i58 : R = KK[x,y,z];

i59 : F = random(R^1, R^{-2,-3})

o59 = {0} | -8514x2+5827xy-9613y2+7886xz+5702yz+11835z2 -15791x3+1 ...

              1       2
o59 : Matrix R  <--- R
```

The `random` command makes a $1 \times 2$ matrix $F$ whose elements have degrees $\{2, 3\}$ (that is, $F$ is a random map to the free module $R^1 = R(0)$, which has its one generator in the default degree, namely 0, from the free module $R(-2) \oplus R(-3)$ with generators in degrees, $\{2, 3\}$). We can compute:

```
i60 : GB = generators gb F

o60 = {0} | x2-14140xy+5100y2+12053xz+14443yz+7878z2 xy2-6589y3-20 ...

              1       3
o60 : Matrix R  <--- R

i61 : LT = leadTerm generators gb F

o61 = {0} | x2 xy2 y4 |

              1       3
o61 : Matrix R  <--- R

i62 : betti LT
    total: 1 3
        0: 1 .
        1: . 1
        2: . 1
        3: . 1
```

The `betti` command shows that there are Gröbner basis elements of degrees 2, 3, and 4. This result is dependent on the monomial order in the ring $R$; for example we could take the lexicographic order

```
i63 : R = KK[x,y,z, MonomialOrder => Lex];
```

For more possibilities, use the *help* facility:

```
i64 : help MonomialOrder

Documentation for MonomialOrder:

'MonomialOrder' -- a key used with monoids to indicate a
     monomial order other than the default (graded reverse lexicog ...

Values:
    'GRevLex' -- graded reverse lexicographic order (the default)
    'GLex' -- graded lexicographic order
    'Lex' -- lexicographic order
    'RevLex' -- reverse lexicographic order
    'Eliminate' -- elimination order
    'ProductOrder' -- product order
Eventually, more general monomial orders will be allowed.
```

We get

```
i65 : F = random(R^1, R^{-2,-3})

o65 = {0} | -9394x2+8312xy+12075xz-3147y2-11542yz+3907z2 -13293x3- ...

              1       2
o65 : Matrix R  <--- R

i66 : GB = generators gb F

o66 = {0} | x2+2417xy+2168xz+13360y2-639yz+14449z2 xy2+198xyz-4686 ...

              1       5
o66 : Matrix R  <--- R

i67 : LT = leadTerm generators gb F

o67 = {0} | x2 xy2 xyz2 xz4 y6 |

              1       5
o67 : Matrix R  <--- R

i68 : betti LT
    total: 1 5
        0: 1 .
        1: . 1
        2: . 1
        3: . 1
        4: . 1
        5: . 1
```

and there are Gröbner basis elements of degrees 2, 3, 4, 5, and 6.

## C.1.d. Division With Remainder

A major application of Gröbner bases is to decide whether an element is in a give ideal, and whether two elements reduce to the same thing modulo an ideal. For example, everyone knows that the trace of a nilpotent matrix over a field is 0. We can produce an ideal $I$ that defines the variety X of nilpotent $3 \times 3$ matrices by taking the ideal generated by the entries of the cube of a generic matrix. Here's how:

```
i69 : R = KK[a..i];
```

```
i70 : M = genericMatrix(R,a,3,3)

o70 = {0} | a d g |
      {0} | b e h |
      {0} | c f i |

               3       3
o70 : Matrix R  <--- R

i71 : N = M^3

o71 = {0} | a3+2abd+bde+2acg+bfg+cdh+cgi        a2d+bd2+ade+de2+cd ...
      {0} | a2b+b2d+abe+be2+bcg+ach+ceh+bfh+chi abd+2bde+e3+bfg+cd ...
      {0} | a2c+bcd+abf+bef+c2g+cfh+aci+bfi+ci2 acd+cde+bdf+e2f+cf ...

               3       3
o71 : Matrix R  <--- R

i72 : I = ideal N

               3                                                    ...
o72 = ideal (a  + 2a*b*d + b*d*e + 2a*c*g + b*f*g + c*d*h + c*g*i, ...

o72 : Ideal of R
```

But the trace is not in $I$! This is obvious from the fact that the trace has degree 1, but the polynomials in $I$ are of degree 3. However, even the 6th power of the trace is not in $I$ (the seventh is; Bernard Mourrain has worked out a general formula telling which power is necessary). We could also check by division with remainder:

```
i73 : Tr = trace M

o73 = a + e + i

o73 : R

i74 : Tr^6 % I

         2 2 2          2 2        3 2      4 2        2   2       ...
o74 = 90a e i  + 90b*d*e i  + 90a*e i  + 90e i  + 90c*e g*i  + 90a ...

o74 : R

i75 : Tr^7 % I

o75 = 0

o75 : R
```

If we actually want to see the quotient, we need to specify the order of the generators of the ideal, that is, we have to give the matrix. The quotient is

```
i76 : Q = Tr^6 // generators I

o76 = {3} | a3+6a2e+3bde+15ae2+22e3+3bfg+3cdh-12efh+6a2i+30aei+60e ...
      {3} | 18de2+18efg+18dei+18fgi                                ...
      {3} | 18deh+18egi+18dhi+18gi2                                ...
      {3} | -27abe-63be2-36ach+9ceh+18bfh-72abi-162bei+90chi       ...
      {3} | -2a3+15a2e+21bde+6ae2+e3-6bfg-6cdh-36afh+15efh+60a2i+7 ...
      {3} | 63beg+36a2h+18bdh-18aeh+18e2h+9cgh+18fh2+117bgi-162ahi ...
      {3} | -18a2c+18ace+18abf-18bef+36cfh+45aci+18cei-81bfi+153ci ...
      {3} | -18cde-36a2f-36bdf+72aef+9e2f-45cfg-18f2h-108cdi+162af ...
      {3} | 16a3+18abd-30a2e-42bde-30ae2-44e3+18acg-18ceg+3bfg+3cd ...
```

```
                9       1
o76 : Matrix R  <--- R
```

and then

```
i77 : (generators I)*Q + (Tr^6 % I) == Tr^6

o77 = true
```

gives "true".

The trace is nonzero mod $I$ simply because the generators of $I$ (the entries of the cube of the matrix) do not generate the ideal of all forms vanishing on $X$ — this we could find (with a little time!) using the `radical` command as follows.

```
i78 : time rI = radical I
     -- used 73.27 seconds

                                      2                          2          ...
o78 = ideal (a + e + i, b*d + e  + c*g + f*h + e*i + i , c*e*g - b ...

o78 : Ideal of R

i79 : J = ideal (trace M,
                 trace exteriorPower(2,M),
                 trace exteriorPower(3,M))

o79 = ideal (a + e + i, - b*d + a*e - c*g - f*h + a*i + e*i, - c*e ...

o79 : Ideal of R

i80 : rI == J

o80 = true
```

Thus we see that the radical is generated by the trace, the determinant, and the sum of the principal $2 \times 2$ minors, that is, by the coefficients of the characteristic polynomial.

## C.1.e. Elimination Theory

Consider the problem of projecting the "twisted cubic", a curve in $\mathbf{P}^3$ defined by the three $2 \times 2$ minors of a certain $2 \times 3$ matrix. Such problems can be solved in a simple direct way using Gröbner bases. The technique lends itself to many extensions and in its developed form can be used to find the closure of the image of any map of affine varieties.

In this section we shall give first a direct treatment of the problem above, and then show how to use *Macaulay 2* as a general tool to solve the problem.

We first clear the earlier meaning of `x` to make it into a subscripted variable,

```
i81 : erase quote x;
```

and then set

```
i82 : R = KK[x_0..x_3];
```

the homogeneous coordinate ring of $\mathbf{P}^3$ and

```
i83 : M = matrix(table(2,3, (i,j)->x_(i+j)))
```

```
o83 = {0} | x_0 x_1 x_2 |
      {0} | x_1 x_2 x_3 |

              2       3
o83 : Matrix R  <--- R

i84 : I = minors(2,M)

                2                            2
o84 = ideal (- x  + x x , - x x  + x x , - x  + x x )
                1    0 2     1 2    0 3     2    1 3

o84 : Ideal of R
```

the ideal of the twisted cubic.

As projection center we take the point defined by

```
i85 : pideal = ideal(x_0+x_3, x_1, x_2)

o85 = ideal (x  + x , x , x )
             0    3   1   2

o85 : Ideal of R
```

To find the image we must intersect the ideal $I$ with the subring generated by the generators of `pideal`. We make a change of variable so that these generators become the last three variables in the ring; that is, we write the ring as $KK[y_0, \ldots, y_3]$ where

$$y_0 = x_0, \quad y_1 = x_1, \quad y_2 = x_2, \quad y_3 = x_0 + x_3$$

and thus $x_3 = y_3 - y_0$, etc. We want the new ring to have an "elimination order" for the first variable.

```
i86 : erase quote y;

i87 : S = KK[y_0..y_3,MonomialOrder => Eliminate 1];
```

Here is one way to make the substitution

```
i88 : I1 = substitute(I, matrix{{y_0,y_1,y_2,y_3-y_0}})

                     2     2                              2
o88 = ideal (y y  - y , - y  + y y  - y y , - y y  - y  + y y )
              0 2    1     0    0 3    1 2     0 1    2    1 3

o88 : Ideal of S
```

The elimination of one variable from the matrix of Gröbner basis elements proceeds as follows:

```
i89 : J = selectInSubring(1,generators gb I1)

o89 = {0} | y_1^3+y_2^3-y_1y_2y_3 |

              1       1
o89 : Matrix S  <--- S
```

and gives (a matrix whose single entry is) the cubic equation of a rational curve with one double point in the plane. However, we are still in a ring with 4 variables, so if we really want a plane curve (and not the cone over one) we must move to yet another ring:

```
i90 : S1 = KK[y_1..y_3];

i91 : J1 = substitute(J, S1)
```

```
o91 = {0} | y_1^3+y_2^3-y_1y_2y_3 |

              1        1
o91 : Matrix S1  <--- S1
```

This time we didn't have to give so much detail to the `substitute` command because of the coincidence of the names of the variables.

Having shown the primitive method, we now show a much more flexible and transparent one: we set up a ring map from the polynomial ring in 3 variables (representing the plane) to $R/I$, taking the variables $y$ to the three linear forms that define the projection center. Then we just take the kernel of this map! ("Under the hood", *Macaulay 2* is doing a more refined version of the same computation as before.)

Here is the ring map

```
i92 : Rbar = R/I;

i93 : f = map(Rbar, S1,
              matrix(Rbar,{{x_0+x_3, x_1,x_2}})
          )

o93 = map(Rbar,S1,{0} | x_0+x_3 x_1 x_2 |)

o93 : RingMap Rbar <--- S1
```

and the desired ideal

```
i94 : J1 = ker f

              3             3
o94 = ideal(y  - y y y  + y )
              2    1 2 3    3

o94 : Ideal of S1
```

## C.1.f. Quotients and saturation

Another typical application of Gröbner bases and syzygies is to the computation of ideal quotients and saturations. Again we give an easy example that we can treat directly, and then introduce the tool used in *Macaulay 2* to treat the general case.

If $I$ and $J$ are ideals in a ring $R$, we define $(I : J)$, the ideal quotient, by

$$(I : J) = \{f \in R \mid fJ \subset I\}$$

In our first examples we consider the case where $J$ is generated by a single element g. This arises in practice, for example, in the problem of homogenizing an ideal. Suppose we consider the affine space curve parametrized by $t \mapsto (t, t^2, t^3)$. The ideal of polynomials vanishing on the curve is easily seen to be $(b - a^2, c - a^3)$ (where we have taken $a, b, c$ as the coordinates of affine space). To find the projective closure of the curve in $\mathbf{P}^3$, we must homogenize these equations with respect to a new variable $d$, getting $(db - a^2, d^2c - a^3)$. But these forms do not define the projective closure! In general, homogenizing the generators of the ideal $I$ of an affine variety one gets an ideal $I1$ that defines the projective closure up to a component supported on the

hyperplane at infinity (the hyperplane $d = 0$). To see the ideal of the closure we must remove any such components, for example by replacing $I1$ by the union $I2$ of all the ideals $(I1 : d^n)$, where $n$ ranges over positive integers. This is not so hard as it seems: First of all, we can successively compute the increasing sequence of ideals $(I1 : d)$, $(I1 : d^2), \ldots$, until we get two that are the same; all succeeding ones will be equal, so we have found the union. A second method involves a special property of the reverse lex order, and is much more efficient in this case. We shall illustrate both. First we set up the example above:

```
i95 : R = KK[a,b,c,d];

i96 : I1 = ideal(d*b-a^2, d^2*c-a^3)

                  2           3      2
o96 = ideal (- a  + b*d, - a  + c*d )

o96 : Ideal of R
```

We discuss now a method for computing the ideal quotient. If $I$ is generated by $f_1, \ldots, f_n$. We see that $s \in (I : J)$ if and only if there are ring elements $r_i$ such that

$$\sum_{i=1}^{n} r_i f_i + sg = 0.$$

Thus it suffices to compute the kernel (syzygies) of the $1 \times (n+1)$ matrix

$$(f_1, \ldots, f_n, g)$$

and collect the coefficients of $g$, that is, the entries of the last row of a matrix representing the kernel. Thus in our case we may compute $(I1 : d)$ by concatenating the matrix for $I1$ with the variable $d$.

```
i97 : I1aug = generators I1 | d

o97 = {0} | -a2+bd -a3+cd2 d |

               1       3
o97 : Matrix R  <--- R

i98 : augrelations = generators ker I1aug

o98 = {2} | -a    d     |
      {3} | 1     0     |
      {1} | ab-cd a2-bd |

               3       2
o98 : Matrix R  <--- R
```

There are 3 rows (numbered 0, 1, 2) and 2 columns (numbered 0, 1), so to extract the last row we may do this.

```
i99 : I21 = submatrix(augrelations, {2}, {0,1})

o99 = {1} | ab-cd a2-bd |
```

```
                1       2
o99 : Matrix R  <--- R
```

But this is not an "ideal", properly speaking: first of all, it is a matrix, rather than a submodule of $R^1$, and second of all its target is not $R^1$ but $R(-1)$, the free module of rank 1 with generator in degree 1. We can fix both of these problems with

```
i100 : I21 = ideal I21

                           2
o100 = ideal (a*b - c*d, a  - b*d)

o100 : Ideal of R
```

This is larger than the original ideal, having two quadratic generators instead of a quadric and a cubic, so we continue. Instead of doing the same computation again, we introduce the built-in command for computing ideal quotients.

```
i101 : I22 = I21 : d

                 2                     2
o101 = ideal (b  - a*c, a*b - c*d, a  - b*d)

o101 : Ideal of R
```

The result is again larger than $I21$, having three quadratic generators. We repeat.

```
i102 : I23 = I22 : d

                 2                     2
o102 = ideal (b  - a*c, a*b - c*d, a  - b*d)

o102 : Ideal of R
```

We get an ideal which is the same as $I22$. Thus the homogeneous ideal $I2$ of the projective closure is equal to $I23$ (this is the homogeneous ideal of the twisted cubic, already encountered above).

A more perspicuous way of approaching the computation of the union of the ideals $(I : d^n)$, which is called the saturation of I with respect to d, and written $(I : d^\infty)$, is first to compute a reverse lex Gröbner basis.

```
i103 : generators gb I1

o103 = {0} | a2-bd abd-cd2 b2d2-acd2 |

                1       3
o103 : Matrix R  <--- R
```

We see that the second generator is divisible by $d$, and the third is divisible by $d^2$. General theory says that we get the right answer simply by making these divisions, that is, the saturation is

$$(a^2 - cd, ac - cd, c^2 - cd),$$

as previously computed. The same thing can be accomplished in one line by

```
i104 : I2 = divideByVariable(generators gb I1,d)

o104 = {0} | a2-bd ab-cd b2-ac |
```

```
                 1       3
o104 : Matrix R  <--- R
```

This ideal $I2$ is the ideal of the projective closure that we wanted to compute.

## C.2 Local cohomology of graded modules.

## C.2.a Mathematical background.

If $R$ is a ring, $\mathfrak{m}$ is an ideal of $R$, and $M$ is an $R$-module, we define the zero-th local cohomology (with supports in $\mathfrak{m}$) to be the submodule

$$H^0_{\mathfrak{m}}(M) \subset M$$

which is the set of those elements of $M$ that are annihilated by some power of $\mathfrak{m}$. It is easy to see that $H^0_{\mathfrak{m}}$ is a left-exact functor on the category of $R$-modules, and we define the $i$-th local cohomology functor $H^i_{\mathfrak{m}}$ to be the $i$-th right derived functor of $H^0_{\mathfrak{m}}$. The local cohomology functors appear frequently in commutative algebra. They generalize the (Zariski) cohomology of coherent sheaves (on a projective or toric variety) and the computation of local cohomology in the corresponding cases gives part of the cohomology of sheaves. In this tutorial we learn how to handle these computations. We shall state without proof the mathematical results on which these computations are based. We refer the reader to the original book *Local Cohomology* by A. Grothendieck (Springer, Lecture Notes in Mathematics ***) or to the somewhat more accesible summaries of special cases given in Eisenbud, *Commutative Algebra with a View Toward Algebraic Geometry*, Appendix 4, Springer Verlag, 1995, Bruns-Herzog, *Cohen-Macaulay Rings*, Cambridge University Press, 1993, and Chapter 8 of this book for more information.

One useful property of the local cohomology is its independence of the ring over which it is computed. We may codify this property as follows.

**Proposition 1.** *Suppose that $R' \to R$ is a homomorphism of rings, and $\mathfrak{m}' \subset R'$ is an ideal. Let $\mathfrak{m} = \mathfrak{m}'R$. If $M$ is an $R$-module, and $M'$ is the same set as $M$, regarded as an $R'$-module, then the local cohomology of $M$ with respect to $\mathfrak{m}$ is the same as the local cohomology of $M'$ with respect to $\mathfrak{m}'$.*

Because of Proposition 1, we may, in computing the local cohomology of a module over a finitely generated algebra, always suppose that the algebra is simply a polynomial ring.

The modules $H^i_{\mathfrak{m}}(M)$ may not be finitely generated, but in the situation where $R$ and $M$ are positively graded, and $\mathfrak{m}$ is the homogeneous maximal ideal of $R$, the module $H^i_{\mathfrak{m}}(M)$ is graded. Moreover, for any integer $e$, the submodule generated by the homogeneous elements of degrees at least $e$ is finitely generated (in fact, $H^i_{\mathfrak{m}}(M)$ is nonzero only in finitely many degrees).

A more refined version of this assertion is the key to the most general method of computing local cohomology. To express it we write $a_i(M)$ for

the maximum of the degrees of the minimal generators of the $i$-th syzygy module of $M$.

**Theorem 2.** *Let $R_0$ be a Noetherian ring, and let $R = R_0[x_1, \ldots, x_n]$ be a graded polynomial ring over $R$. Let $\mathfrak{m} = (x_1, \ldots, x_n) \subset R$ and let $M$ be a finitely generated graded $R$-module. If $J$ is any homogeneous ideal of $R$ containing a power of $\mathfrak{m}$ then for $i \geq 1$ there is a natural map $\mathrm{Ext}^i_R(R/J, M) \to H^i_{\mathfrak{m}}(M)$ which is an isomorphism in all large degrees. The map is an isomorphism in all degrees $\geq e$ if $J$ is contained in $\mathfrak{m}^d$, where $d = \max(a_{n-i}(M), a_{n-i+1}(M)) - n + 1 - e$.*

*Proof sketch.* The $i$-th local cohomology is the homology of the subcomplex

$$\ldots \to E_{i-1} \to E_i \to E_{i+1} \to \ldots$$

of the graded injective resolution of $M$ consisting of all elements annihilated by some power of $\mathfrak{m}$; the Ext is the homology of the subcomplex of elements annihilated by $J$. Thus they agree in degrees $\geq e$ if $J$ annihilates the parts of $E_{i-1}$ and $E_i$ of degrees $\geq e$, and this will be true as soon as $J$ is contained in the $d$-th power of $\mathfrak{m}$ for $d = a - e + 1$, where a is the maximum degree of a socle element of $E_{i-1}$ or $E_i$.

The degrees of the socle elements of $E_i$ are (by local duality – see below) minus the degrees of the generators of

$$Ext^{n-i}(M, R(-n)) = Ext^{n-i}(M, R)(-n).$$

Since $Ext^{n-i}(M, R)$ is a subquotient of the dual of the free resolution of $M$, we see that

$$a \leq \max(a_{n-i}(M), a_{n-i+1}(M)) - n,$$

which yields the desired estimate. ∎

It happens that in many important cases the local cohomology is nonzero in only finitely many degrees, and then of course we can compute it all.

Another way to get at the whole local cohomology module is through the local duality theorem, which also allows quick computation of the local cohomology in some simple cases. Here is the version we use:

**Theorem 3.** *Let $k$ be a field, and let $R = k[x_1, \ldots, x_n]$ be a graded polynomial ring over $R$. Let $\mathfrak{m} = (x_1, \ldots, x_n) \subset R$ and let $M$ be a finitely generated graded $R$-module. Writing $W = R(-n)$ (the "relative canonical module") we have*

$$H^i_{\mathfrak{m}}(M) = \mathrm{Hom}(\mathrm{Ext}^{n-i}_R(M, W), k).$$

Finally, to know where to look for interesting invariants, it helps to know the following:

**Theorem 4.** *Let $k$ be a field, and let $R = k[x_1, \ldots, x_n]$ be a graded polynomial ring over $R$. Let $\mathfrak{m} = (x_1, \ldots, x_n) \subset R$ and let $M$ be a finitely generated graded $R$-module. The local cohomology $H^i_{\mathfrak{m}}(M)$ is zero for $i < \operatorname{depth} M$ and for $i > \dim M$. It is nonzero for $i = \operatorname{depth} M$ or $i = \dim M$ (and may be zero or not for values in between).*

## C.2.b Macaulay 2 routines

With these techniques in hand, we can explain the routines that *Macaulay 2* uses to compute local cohomology. Here is the routine using Theorem 2.

```
i1 : localCohomology = method();

i2 : localCohomology(ZZ,Module,ZZ) := (i,M,e) -> (
          -- Use the formula
          -- H^i(M) = Ext^i(R/J,M) in degrees >= e,
          -- valid whenever J is an ideal contained in the
          -- p th power of the max ideal (of finite
          -- colength),
          -- where
          -- p >= max(a(r-i)-r+1, a(r-i+1)-r+1)-e,
          -- and
          -- a(i) = max degree of an i th syzygy
          -- of M over a polynomial ring R in r variables.
          -- We need to place M into a polynomial ring,
          -- and compute the resolution there, at least
          -- r-i+2 steps back.
          -- To find the degree bound we need, we must
          -- work over a polynomial ring, so we move
          -- to the presenting ring, if necessary:
          A := ring M;
          F := presentation A;
          R := ring F;
          -- Make a copy N of M over the polynomial ring
          N := coker lift(presentation M,R) ** coker F;
          -- now compute the resolution.  We need to get
          -- the r-i+1 differential right.
          r := numgens R;
          C := res (N,LengthLimit => r-i+2);
          -- at this point it costs nothing to check
          -- whether depth M <= i <= dim M, and
          -- return 0 otherwise
          if i > dim N or length C < r-i
          then A^0
          else (
               -- Find a degree bound p for the cohomology
               -- computation to come:
               deg1 := first max degrees C_(r-i);
               p := deg1-r+1-e;
               if length C >= r-i+1 then (
                    deg2 := first max degrees C_(r-i+1);
                    p = max(p, deg2-r+1-e);
                    );
               p = max(p,0);
               -- Find J. Since we are going to do Koszul
               -- homology, we can work directly in A = ring M.
               -- We compute Ext^i(R/J,M)
               -- as the cohomology of
               -- the Koszul complex on J,
               -- or equivalently as the homology,
               -- in complementary degree, twisted by rp:
               J := map(A^1,A^{r:-p},
                         {apply(numgens A, j -> A_j^p)});
               HM := homology(M**koszul(r-i,J),
                              M**koszul(r-i+1,J));
```

```
          -- since a nonminimal presentation is not so
          -- useful, we use prune to get a minimal one.
          prune (A^{r*p} ** HM)));
```

To implement the routine using Theorem 3, we need a method to ascertain the largest and smallest degrees in which a module of finite length has a nonzero component. The routine `degreeRange` below provides, for a module M, a list of those degrees in which M has a nonzero component.

```
i3 : degreeList = (M) -> (
          if dim M > 0
          then error "expected module of finite length";
          H := poincare M;
          T := (ring H)_0;
          H = H // (1-T)^(numgens ring M);
          exponents H / first);
```

Here are the main routines. In the first, `localCohomology1`, we produce a module $M$ that is, in a certain range of degrees, the dual of the module we want. We begin with a routine to find the dual of the interesting piece of $M$. More precisely, this routine assumes that the underlying ring of $M$ is a polynomial ring. It returns the dual of the finite length module $M_{<e}$ that agrees with $M$ in degrees $< e$ and is zero in larger degrees.

```
i4 : truncatedDual = method();

i5 : truncatedDual(Module,ZZ) := (M,e) -> (
          -- find (k-dual M), truncated in degrees >= e.
          R := ring M;
          n := numgens R;
          ww := R^{-n};
          M1 := prune (M / (truncate(-e+1,M)));
          Ext^n(M1,ww));
```

Now the routine that computes local cohomology via local duality:

```
i6 : localCohomology1= method();

i7 : localCohomology1(ZZ,Module,ZZ) := (i,M,e) -> (
          -- compute the degree >= e part of the
          -- local cohomology H^i(M).
          -- The method used here is local duality:
          -- H^i(M) is k-dual to Ext^(n-i)(M,R(-n)), where
          -- the ring R of M is a polynomial ring in n
          -- variables.
          --
          -- First step: bring M back to a polynomial ring,
          -- if necessary.
          A := ring M;
          F := presentation A;
          R := ring F;
          M = coker lift(presentation M,R) ** coker F;
          -- Second step: compute the Ext
          n := numgens R;
          ww := R^{-n};
          E := prune Ext^(n-i)(M,ww);
          -- Third step: dualize.  If E is finite length,
          -- this may be done as Ext^n(E,ww).
          -- Otherwise, we must truncate degrees (or else the
          -- result will not be finitely generated).
          result := (
               if dim E <= 0
               then Ext^n(E,ww)
               else truncatedDual(E,e)
               );
          -- Finally, we put the module back into
          -- the original ring, if A is not R.
          prune (result ** A)
```

```
);
```

## C.2.c Example: The rational quartic curve in $\mathbf{P}^3$

As an example we compute the local cohomology of the ring A which is the homogeneous coordinate ring of the smooth rational quartic curve in $\mathbf{P}^3$.

First we set up the homomogeneous coordinate ring of projective space.

```
i8 : kk = ZZ/32003;

i9 : R = kk[x_0..x_3];
```

Then we construct the ideal of the quartic.

```
i10 : I = monomialCurve(R,{1,3,4})

                          3      2     2    2     3    2
o10 = ideal (x x  - x x , x  - x x , x x  - x x , x  - x x )
             1 2    0 3   2    1 3   0 2    1 3   1    0 2

o10 : Ideal of R

i11 : S = R/I;
```

And now we compute the local cohomology in degrees $\geq -5$.

```
i12 : X0 = localCohomology(0, S^1, -5)

o12 = 0

o12 : S - module
```

Nothing there; we're below depth $S$.

```
i13 : X1 = localCohomology(1, S^1, -5)

o13 = cokernel {1} | x_3 x_2 x_1 x_0 |

                                    1
o13 : S - module, quotient of S
```

One dimensional, concentrated in degree 1; this is the "Hartshorne-Rao module" of the quartic.

```
i14 : X2 = localCohomology(2, S^1, -5)

o14 = cokernel {-8}  | 0    0    0    0    0    0   0    -x_3 0    ...
               {-8}  | 0    0    0    0    0    0   -x_3 0    0    ...
               {-8}  | 0    0    0    0    0    0   0    x_2  0    ...
               {-8}  | 0    0    0    0    0    0   x_2  0    -x_3 ...
               {-8}  | 0    0    0    0    0    0   0    0    0    ...
               {-8}  | 0    0    0    0    0    0   0    0    x_2  ...
               {-8}  | 0    0    0    0    0    x_3 0    0    0    ...
               {-8}  | 0    0    0    0    0    0   0    0    0    ...
               {-10} | -x_3 0    -x_2 -x_1 0    0   0    0    0    ...
               {-10} | x_2  -x_3 0    x_0  -x_1 0   0    0    0    ...
               {-10} | 0    x_2  x_1  0    x_0  0   0    0    0    ...

                                    11
o14 : S - module, quotient of S
```

This module is more complicated! The second local cohomology is the dual of the canonical module, and is in particular infinite-dimensional.

```
i15 : X3 = localCohomology(3, S^1, -5)

o15 = 0
```

```
o15 : S - module
```

Again 0; we're above dim $S$.

## C.3 Cohomology of a coherent sheaf

## C.3.a Mathematical background

Next we turn to the problem of computing the cohomology of a sheaf $F$ on a projective variety $X$. We suppose that $X$ is already embedded in a projective space $P^r$. We may thus represent the variety by giving its homogeneous coordinate ring $R$. Given a graded $R$-module $M$ there is an associated sheaf $F = \tilde{M}$, and every quasicoherent sheaf $F$ can be represented in this way. If $M$ is finitely generated, then $F$ is coherent, and every coherent sheaf can be represented by a finitely generated graded module. However, these representations are not unique, and sometimes one may even represent a coherent sheaf by modules that are not finitely generated.

A result corresponding to Proposition 1 above shows that for the computation of cohomology it does not really matter whether we regard $F$ as a sheaf on $X$ or on $P^r$. We will therefore write $H^i(F)$, not mentioning $X$, for the $i$-th cohomology of $F$.

Given a graded module $M$ we can produce another one by "shifting" degrees: we define $M(n)$ to be the module whose degree d part is $M_{n+d}$. The corresponding sheaf is

$$\widetilde{M(n)} = F(n) = F \otimes \mathcal{O}_{\mathbf{P}^r}(n),$$

called the $n$-th "twist" of $F$, where $\mathcal{O}_{\mathbf{P}^r}(n)$ is the $n$-th tensor power of the tautological bundle on $\mathbf{P}^r$. There is a also a canonical graded module representing $F$, given by

$$M_\infty = H^0_*(F) = \bigoplus_{n \in Z} H^0(F(n))$$

but this may fail to be finitely generated even when $F$ is coherent (this happens when $F$ has associated varieties of dimension 0 in the projective space).

We now assume that $F$ is coherent. For any integer $e$, the module obtained as the sum of the global sections of all the twists $F(n)$ for $n \geq e$, that is

$$M = H^0_{\geq e}(F) = \bigoplus_{n \geq e} H^0(F(n))$$

has $\tilde{M} = F$, and our first task is to compute $M$. We call it a *truncation* of $M_\infty$. Of course we will also want to compute each of the corresponding truncations of the higher cohomology sums,

$$H^i_{\geq e}(F) = \bigoplus_{n \geq e} H^i(F(n)).$$

The computations for $i > 0$ reduce to the computations of local cohomology because of the isomorphism

$$H^i_*(F) = H^{i+1}_{\mathfrak{m}}(M),$$

valid for any module $M$ representing a sheaf $F$ and any integer $i > 0$, where $\mathfrak{m}$ is the maximal homogeneous ideal of the homogeneous coordinate ring of the projective space. Thus it would suffice to do the case $i = 0$. However, there is a formulation which works for $i = 0$ and is the same for all $i$, and we shall present this version. Again we let $a_i(M)$ be the maximal degree of a minimal generator of the $i$-th syzygy module for $M$. The following result corresponds to the computation of local cohomology made in Theorem 2.

**Theorem 5.** *Let $k$ be a field, and let $R = k[x_0, \dots, x_r]$ be the graded polynomial ring over $R$. Let $\mathfrak{m} = (x_0, \dots, x_r) \subset R$, and set $n = r + 1$, the number of variables. Let $M$ be a finitely generated graded module over $R$ containing no nonzero submodule of finite length, and let $\tilde{M}$ be the corresponding sheaf on $P^r$. If $J$ is any homogeneous ideal of $R$ containing some power of $\mathfrak{m}$ then for $i \geq 0$ there is a natural map*

$$\operatorname{Ext}^i_R(J, M) \to H^i_*(\tilde{M})$$

*which is an isomorphism in all large degrees. The map is an isomorphism in all degrees $\geq e$ if $J$ is contained in $\mathfrak{m}^d$, where*

$$d \geq \max(a_{n-i}(M), a_{n-i+1}(M)) - n + 1 - e.$$

*Proof sketch.* If $i > 0$ then

$$\operatorname{Ext}^i_R(J, M) = \operatorname{Ext}^{i+1}_R(R/J, M)$$

and

$$H^i_*(M) = H^{i+1}_{\mathfrak{m}}(M)$$

and the result reduces to Theorem 2 above (note that we have not used the hypothesis that $M$ has no nonzero submodule of finite length in this part).

Now suppose that $i = 0$. Since $J = R$ locally on the punctured spectrum of $R$, any map $J \to M$ of degree $d$ induces a global section of $\widetilde{M(d)}$, and this defines a natural map

$$\operatorname{Hom}(J, M) \to H^0_*(\tilde{M}).$$

If $M$ contains no nonzero submodule of finite length we use the short exact sequence

$$0 \to J \to R \to R/J \to 0$$

to obtain the exact sequence

$$0 \to Hom(R, M) \to Hom(J, M) \to Ext^1(R/J, M) \to 0$$

and with the exact sequence

$$0 \to M \to H^0_*(\tilde{M}) \to H^1_{\mathfrak{m}}(M) \to 0$$

we get the following commutative diagram.

$$\begin{array}{ccccccccc} 0 & \longrightarrow & M & \longrightarrow & \mathrm{Hom}(J,M) & \longrightarrow & \mathrm{Ext}^1(R/J,M) & \longrightarrow & 0 \\ & & \| & & \downarrow & & \downarrow & & \\ 0 & \longrightarrow & M & \longrightarrow & H^0_*(\tilde{M}) & \longrightarrow & \mathrm{H}^1_{\mathfrak{m}}(M) & \longrightarrow & 0. \end{array}$$

From the snake Lemma we see at once that the map $\mathrm{Hom}(J, M) \to H^0_*(\tilde{M})$ is an isomorphism in degree $d$ if and only if the map

$$\mathrm{Ext}^1(R/J, M) \to H^1_{\mathfrak{m}}(M)$$

is an isomorphism in degree $d$. Thus the result again reduces to Theorem 2 above. ■

## C.3.b Macaulay 2 routines for sheaf cohomology

The following routines call one version or another of the localCohomology routines for the higher cohomology, and the globalSections routine for $H^0$.

```
i16 : sheafCohomology = method();

i17 : sheafCohomology(ZZ, Module, ZZ) := (i,M,e) -> (
          if i === 0
          then globalSections(M,e)
          else localCohomology(i+1, M, e)
          );

i18 : sheafCohomology1 = method();

i19 : sheafCohomology1(ZZ, Module, ZZ) := (i,M,e) -> (
          if i === 0
          then globalSections(M,e)
          else localCohomology1(i+1, M, e)
          );

i20 : globalSections = method();

i21 : globalSections(Module,ZZ) := (M,e) -> (
          -- if M has a submodule of finite length,
          -- kill it:
          M = M / saturate 0_M;
          -- To compute degree bounds we need to work over
          -- a polynomial ring, so we make a copy of
          -- M over a polynomial ring if we're not already
          -- there:
          A := ring M;
          F := presentation A;
          R := ring F;
```

```
N := coker lift(presentation M,R) ** coker F;
r := numgens R;
wR := R^{-r};
-- Find the degree bound for the ideal J below.
if pdim N < r-1
then M
else (
     -- We must find an ideal J that
     -- annihilates  H^1(local,M) or,
     --  equivalently, annihilates the dual ext
     E1 := Ext^(r-1)(N,wR);
     -- If E1 has finite length, then we don't
     -- need to truncate the answer below, and  we
     -- may use the spread of degrees in which
     --  E1 is nonzero to estimate the annihilator
     -- (1) If E1 is finite length,
     --     let p := max deg E1 - min deg E1 + 1;
     -- (2) If E1 is not finite length,
     -- but the part of
     --     degree >= e is desired, choose
     --     p := min deg E1 + e + 1
     p := max(0,
          if dim E1 <= 0
          then (
               max degreeList E1
               - min degreeList E1 + 1
               )
          else min degrees E1 + e + 1
          );
     -- We move back to A and M:
     J := ideal apply(numgens A, j -> A_j^p);
     Hom(module J,M))
);
```

## C.3.c Example: The Hodge diamond of a K3 surface.

The Hodge diamond of a projective variety X of dimension $n$ is the set of numbers $h^{p,q} = H^q(\Omega_X^p)$, where $\Omega_X^p$ denotes the $p$-th exterior power of the cotangent bundle of $X$. If $X$ is smooth there are equalities $h^{p,q} = h^{q,p}$ and $h^{p,q} = h^{n-q,n-p}$. For a K3 surface the numbers are

$$\begin{array}{ccccc} & & h^{2,2} & & \\ & h^{3,0} & & h^{0,3} & \\ h^{2,0} & & h^{1,1} & & h^{0,2} \\ & h^{1,0} & & h^{0,1} & \\ & & h^{0,0} & & \end{array} \quad = \quad \begin{array}{ccccc} & & 1 & & \\ & 0 & & 0 & \\ 1 & & 20 & & 1 \\ & 0 & & 0 & \\ & & 1 & & \end{array}$$

Any nonsingular quartic surface in $\mathbf{P}^3$, for example the "Fermat" quartic below, is a K3 surface.

```
i22 : kk = ZZ/32003;

i23 : R = kk[x_0..x_3];

i24 : F = sum(4,i->x_i^4)

       4    4    4    4
o24 = x  + x  + x  + x
       0    1    2    3
```

```
o24 : R

i25 : S = R/F;
```

We compute the cotangent bundle as the homology of the complex

$$S^g \xrightarrow{J} S^{r+1} \xrightarrow{xx} S$$

where $xx$ is the row of variables and $J$ is the Jacobian matrix of the ideal presenting $S$.

```
i26 : cotangentBundle = (S) -> (
          F := presentation S;
          J := jacobian F ** S;
          xx:= vars ring F ** S;
          prune homology(xx,J));
```

Here is the computation of the Hodge diamond for the Fermat quartic in $\mathbf{P}^3$ using the routine `sheafCohomology`, presented in the format

$$\begin{matrix} h^{0,0} & h^{0,1} & h^{0,2} \\ h^{1,0} & h^{1,1} & h^{1,2} \\ h^{2,0} & h^{2,1} & h^{2,2}. \end{matrix}$$

```
i27 : Omega = cotangentBundle S

o27 = cokernel {2} | 0    0    -x_1 -x_2 x_3^3 0      0      x_0^3 ...
               {2} | 0    -x_1 0    x_3  x_2^3 0      x_0^3  0     ...
               {2} | 0    x_2  x_3  0    x_1^3 x_0^3  0      0     ...
               {2} | -x_2 0    x_0  0    0     -x_3^3 0      x_1^3 ...
               {2} | x_3  x_0  0    0    0     -x_2^3 x_1^3  0     ...
               {2} | x_1  0    0    x_0  0     0      -x_3^3 x_2^3 ...

                                6
o27 : S - module, quotient of S

i28 : time matrix table(3,3,(p,q)->
         hilbertFunction(0,
            sheafCohomology(q,exteriorPower(p,Omega),0)
            ))
     -- used 51.81 seconds

o28 = | 1 0  1 |
      | 0 20 0 |
      | 1 0  1 |

              3       3
o28 : Matrix ZZ  <--- ZZ
```

Here is the corresponding computation using `sheafCohomology1`.

```
i29 : time matrix table(3,3,(p,q)->
         hilbertFunction(0,
            sheafCohomology1(q,exteriorPower(p,Omega),0)
            ))
     -- used 31.97 seconds

o29 = | 1 0  1 |
      | 0 20 0 |
      | 1 0  1 |

              3       3
o29 : Matrix ZZ  <--- ZZ
```

Use Macaulay2 to compute minimal primes of complicated ideal. I tried computing the minimal primes of a fairly complex ideal using the online Macauley2 interface. I start by letting R=QQ[z] and commutative-algebra computer-algebra-systems macaulay2. asked Apr 19 '19 at 5:02. Benighted. 2,13266 silver badges1313 bronze badges. Macaulay2 is a free computer algebra system created by Daniel Grayson (from the University of Illinois at Urbanaâ€“Champaign) and Michael Stillman (from Cornell University) for computation in commutative algebra and algebraic geometry. Macaulay2 is built around fast implementations of algorithms useful for computation in commutative algebra and algebraic geometry. This core functionality includes arithmetic on rings, modules, and matrices, as well as algorithms for GrÃ¶bner bases, free resolutions Contents 1. Elementary uses of Macaulay 2 a. First steps b. A monomial curve example c. Random regular sequences d. Division with remainder e. Elimination theory f. Quotients and saturation 2. Local cohomology of graded modules a. Mathematical Background b. Macaulay 2 routines c. Example: the rational quartic curve in P3 3. Zariski cohomology of coherent sheaves. 1.Â  Start Macaulay 2 with the command M2, and you will be presented with â€œi1 :â€ as input prompt.