Computational Algebraic Geometry with Macaulay2

Takehiko Yasuda

This is an English traslation from lecture notes “Macaulay2 niyoru keisandaisūkika” written in Japanese for an intensive lecture which the author gave at Tokyo Metropolitan University in October 2012. The translation was made in November 2024 with the aid of machine translation. Some codes may not work with the latest version of Macaulay2 and some descriptions may be outdated.

CHAPTER 1

Introduction

These notes were for an intensive lecture at Tokyo Metropolitan University in October 2012.

1. Goals

The goals of this lecture are as follows:

(1) Become familiar with Macaulay2.
(2) Gain a rough understanding of what can be computed.
(3) Learn to write functions and perform simple programming.
(4) Learn how to look up the manual and continue improving your skills independently.

I have never systematically studied Macaulay2 myself; I learned it gradually by trial and error, and now I use it frequently in my research. Generally, this method seems to be a shortcut to mastering programming (though it might be different for those aiming to become professional programmers). In this lecture, I plan to develop skills by dealing with examples of calculations that are likely to be performed in algebraic geometry and commutative algebra.

Each individual's required calculations are different, and there are time constraints, so there is a limit to what can be taught. After this lecture, I hope you will read the manual and refer to other people's programs to improve your skills. As references, I recommend [**2, 5**]. There is also a Japanese reference included in Knoppix/Math, [**7**].[1]

2. Recommendations for Experiments

The use of computers in mathematical research is gradually increasing, but the image of "mathematics being done solely with paper and pencil" is still strong. However, I highly recommend numerical experiments using computers. Here are some things you can do with a computer:

(1) To verify conjectures: Especially for conjectures that can be quickly shown to be incorrect through computation, this prevents wasting time trying to prove them.
(2) To formulate conjectures: Observing computational results and discovering patterns is the essence of mathematics. To explore unknown areas, observation is essential.
(3) To support proofs: Many modern mathematical proofs are complex and leave doubts about their correctness. Numerical experiments can help reinforce these proofs.

These points also apply to manual calculations, but the scope of what computers can handle is entirely different from that of manual calculations. On the other hand, there are

[1] "KNOPPIX/Math is a project to archive free mathematical software and free mathematical documents and offer them on KNOPPIX. It provides a desktop for mathematicians that can be set up easily and quickly." (T. Hamada and Knoppix/Math committers, "KNOPPIX/Math," http://www.math.kobe-u.ac.jp/HOME/taka/2008/knoppix-en-2008.pdf) There is a direct descendant of KNOPPIX/Math Project, called MathLibre. (https://www.mathlibre.org/index.html)

many theories that cannot be captured by computer calculations. Mathematics often deals with infinities, which must be cleverly avoided in computations. In the future, mathematics will likely become more like the natural sciences, with theory and experimentation forming two pillars.

3. Prerequisites

Basic knowledge of commutative algebra and algebraic geometry is assumed. If there are any unclear points, please refer to standard textbooks as needed.

4. Exercise Solutions

For the exercises marked with an asterisque * in the notes, solutions are provided at the end of the notes.

5. Starting and Exiting

In this lecture, we will use Macaulay2 within Emacs.

To start Macaulay2, in Knoppix/Math (which the students in this lecture should be using), select Macaulay2 from the mathematics software menu to launch both Emacs and Macaulay2. Alternatively, you can start Emacs alone and then press the F12 key to launch Macaulay2.

When Macaulay2 starts, the following screen will appear, with the cursor blinking to the right of `i1 :` , indicating that it is ready for input.

```
+ M2 --no-readline --print-width 88
Macaulay2, version 1.4
with packages: ConwayPolynomials, Elimination, IntegralClosure, LLLBases,
               PrimaryDecomposition, ReesAlgebra, TangentCone

i1 :
```

For example, if you enter 1+1 and press the return key, the result of the calculation will be displayed as follows.

```
i1: 1+1

o1 = 2
```

To exit the program, type `quit`.

```
i2 : quit

Process M2 finished
```

Exercise 1. Restart Macaulay2 with F12 and exit with `quit`.

If you need any further assistance, feel free to ask!

6. Manual

When using Macaulay2 (hereafter abbreviated as M2), you may need to look up how to use function names or find functions that perform the calculations you want. For this, you need to refer to the manual. By using `viewHelp`, a browser will open, allowing you to view the online manual. It is convenient to bookmark this page. Additionally, if you want to look

up a specific function name, such as `quit`, you can use `viewHelp quit`. The Index page of the manual (accessible via a link at the top of the manual's main page) is also frequently used to find functions.

Moreover, the M2 website (url: http://www.math.uiuc.edu/Macaulay2/) contains useful information, so it is worth checking out. You can also access it through links from the online manual.

EXERCISE 2.
(1) Use `viewHelp quit` to view the online manual for `quit`. Compare it with `help quit`.
(2) From the online manual, find the function that calculates binomial coefficients. Also, find the function that computes the remainder of integer division.
(3) Using the search box at the top left of the M2 website, find information related to Emacs.

CHAPTER 2

Fields, Polynomial Rings, Ideals, Quotient Rings – Essential Preparations

1. Fields

Many calculations in M2 require first setting up a ring (usually a quotient ring of a polynomial ring) to be considered. Let's go through the steps to set up a quotient ring. First, let's look at the field that will serve as the coefficient field. Try entering `QQ`.

```
i1 : QQ

o1 = QQ

o1 : Ring
```

This represents the field of rational numbers. You can perform calculations within the field of rational numbers as follows.

```
i2 : 2/3+6/7

     32
o2 = --
     21

o2 : QQ
```

Finite fields can also be used. For example, the field with 5 elements is

```
i3 : ZZ/5

     ZZ
o3 = --
      5

o3 : QuotientRing
```

To perform calculations within this field, do the following.

```
i10 : R = ZZ/5

o10 = R

o10 : QuotientRing

i11 : 4_R
```

```
o11 = -1

o11 : R

i12 : 3_R^5

o12 = -2

o12 : R
```

(Oops, the numbering skipped. In the future, please don't mind if the input/output numbers skip or go back a bit, even if it looks a bit awkward.) Here, by adding `_R`, we are considering elements of R.

EXERCISE 3. Using finite fields, verify Fermat's Little Theorem ($a^{p-1} \equiv 1 \mod p$) for various values.

1.0.1. *Notes on Coefficient Fields.* In M2, the base fields most commonly used are the prime fields $\mathbb{Q}$ and $\mathbb{F}_p$. Although finite extensions of these fields can also be used, they are less frequently employed. Real and complex number fields can be used, but many calculations cannot be performed over these fields, so they are rarely used. Since many calculations are performed over fields that are not algebraically closed, some caution is needed. Basic knowledge of commutative algebra and scheme theory is desirable.

Although I don't use it often, even for problems of characteristic 0, M2 sometimes performs calculations over $\mathbb{F}_p$ for a large prime p to speed up the computations. In many cases, the results match those in characteristic 0. In Schenck's book [**5**], the prime 101 is mainly used, while in Eisenbud's article in [**2**], the largest usable prime 32749 is used.

EXERCISE 4. Check the online manual to find out how to handle finite fields other than prime fields.

2. Polynomial Rings

Let's consider polynomial rings.

```
i14 : S = QQ[x,y,z]

o14 = S

o14 : PolynomialRing
```

This is a polynomial ring in three variables with rational coefficients. You can input polynomials in this ring as follows.

```
i19 : f = (2*x*y^3*z^2-5*x^6)*(x-y+z)^2

          8      7      6 2      7       6        6 2     3 3 2     2 4 2       5 2
o19 = - 5x  + 10x y - 5x y  - 10x z + 10x y*z - 5x z  + 2x y z  - 4x y z  + 2x*y z  + 4x
      ------------------------------------------------------------------------------
          3 4
      2x*y z
```

```
o19 : S
```

Note that you cannot omit the **multiplication symbol** * between coefficients and variables. Next, let's perform calculations with a finite field as the coefficient.

```
i20 : T = ZZ/7[x,y,z];

i21 : g = (3*x+y-z)^7

        7    7    7
o21 = 3x  + y  - z

o21 : T
```

The **output was suppressed with a semicolon** ;. To return to the previous polynomial ring, use `use S`.

```
i22 : use S; (x+y)^7

       7     6       5 2      4 3      3 4      2 5       6    7
o23 = x  + 7x y + 21x y  + 35x y  + 35x y  + 21x y  + 7x*y  + y

o23 : S
```

You can **connect multiple statements with a semicolon**. If there is no semicolon at the end, the last statement will be evaluated and output.

EXERCISE 5. (1) Type `x^3+3*x^2*y+3*x*y^2+y^3 == (x+y)^3` and confirm that `true` is output. Look up `==` in the manual.
(2) Find and use the function that returns the degree of a polynomial from the manual.*

3. Ideals

If the symbols you have set become cluttered, **reset** them.

```
i24 : clearAll
```

Let's consider ideals in a polynomial ring.

```
i25 : S = QQ[x,y]; I = ideal (x+y)^3

              3     2        2    3
o26 = ideal(x  + 3x y + 3x*y  + y )

o26 : Ideal of S
```

Let's compute the radical of this ideal.

```
i27 : radical I

o27 = ideal(x + y)

o27 : Ideal of S
```

Verify that cubing this returns to the original ideal.

```
i28 : I == oo^3

o28 = true
```

Here, I forgot to name the radical ideal, so I reused the previous result with `oo`. **You can explicitly specify the number, such as `o27`, to use earlier results. You can also use `ooo` and `oooo` for results two and three steps back.**

EXERCISE 6. (1) Use M2 to verify that the ideal $(xy, x^2) \subset k[x, y]$ is not a radical ideal.*
(2) Look up the usage of `isSubset` and verify that the ideal (xy, x^2) is contained in (x).
(3) Find out how to compute a Gröbner basis from an ideal. How can you change the monomial order used for computation?

4. Quotient Rings

Let's use quotient rings of polynomial rings.

```
i29 : S/I

                 S
o29 = ----------------------
       3     2         2    3
      x  + 3x y + 3x*y  + y

o29 : QuotientRing

i30 : dim oo

o30 = 1
```

The dimension of this quotient ring is 1. You can also write it directly as follows.

```
i31 : R = ZZ/13[x,y,z]/( x+y-z^3, y^4-8*x^2+z^2)

o31 = R

o31 : QuotientRing
```

In this ring, $x + y - z^3 = 0$.

```
i32 : x+y-z^3

o32 = 0

o32 : R
```

EXERCISE 7. (1) Use `describe S` to check the data contained in the polynomial ring S.
(2) Define various quotient rings and use `dim` to compute their dimensions, confirming that the values are as expected. Also, use M2 to check whether they are normal rings (Hint: package).
(3) Look up the function `ambient`. How does this function work on quotient rings?
(4) Similarly, look up `Spec`.

5. Saving and Reusing Calculation Results

To save M2 calculation results in Emacs, use C-x, C-w to save the file as a text file, or click the toolbar to save. For example, name the file with the extension .m2, such as abcde.m2.

When you reopen the saved file in Emacs, it will be displayed with syntax highlighting. To reuse the calculation results (or rather the previous inputs), do the following: Start M2 in a separate frame with F12. Press F11 on any input line in the opened file to input it into the currently running M2 session.

Exercise 8. Try the above steps.

CHAPTER 3

Singularities and Inflection Points of Plane Curves – Getting Started

1. Singularities

First, let's consider a curve called Pascal's Limaçon.

```
i1 : limacon = QQ[x,y]/((x^2+y^2-2*x)^2-(x^2+y^2))

o1 = limacon

o1 : QuotientRing
```

Let's compute the singularities of this plane curve.

```
i2 : singularLocus limacon

                                                        QQ[x, y]
o2 = ------------------------------------------------------------------------------------
       4     2 2    4     3       2     2    2    3       2      2     2         2
     (x  + 2x y  + y  - 4x  - 4x*y  + 3x  - y , 4x  + 4x*y  - 12x  - 4y  + 6x, 4x y + 4y

o2 : QuotientRing
```

This is in the form of a quotient ring, and the ideal in the denominator may not be radical.

```
i3 : radical ideal oo

o3 = ideal (y, x)

o3 : Ideal of QQ[x, y]
```

This shows that the singularity is only at the origin. Here, `oo` allows you to reuse the previous output. The composed command `radical ideal` applies the function `ideal` to `oo` and then applies `radical`. Next, let's consider the projective closure of this curve.

```
i5 : Plimacon = QQ[x,y,z]/((x^2+y^2-2*x*z)^2-(x^2+y^2)*z^2)

o5 = Plimacon

o5 : QuotientRing
```

We homogenized with the variable z. Let's compute the singularities again. This time, we apply three functions at once.

```
i6 : radical ideal singularLocus Plimacon

                      2    2
```

```
o6 = ideal (y*z, x*z, x  + y )

o6 : Ideal of QQ[x, y, z]

i7 : dim oo

o7 = 1
```

The dimension is 1. This makes sense because we are considering an affine cone. Typing `singularLocus` can be tedious, but you can use the **TAB key to autocomplete** after typing `sing`. If there are multiple candidates, they will be displayed in a separate frame, and you can click the desired one (this applies when using Emacs). You can also **reuse previous inputs with Control+up/down keys**.

Let's perform a primary decomposition.

```
i8 : decompose o6

              2    2
o8 = {ideal (x  + y , z), ideal (y, x)}

o8 : List
```

I reused the sixth output with `o6`. The ideal (y, x) corresponds to the singularity at the origin. The other ideal gives two points defined by $x^2 + y^2 = 0$ on the line at infinity $z = 0$ (homogeneous coordinates x, y). These two points are not defined over reals.

EXERCISE 9. (1) Automatize the homogenization of defining equations.*
(2) Find all singularities of the projective closure of Cayley's sextic curve $4(x^2+y^2-x)^3 = 27(x^2+y^2)^2$.
(3) Find the non-normal locus of the Roman surface $x^2y^2 + y^2z^2 + z^2x^2 + xyz = 0$ using M2 (Hint: package, homogenization).*

2. Inflection Points

Consider a plane curve $C = (f = 0) \subset \mathbb{P}^2$. A point $p \in C$ is an **inflection point** if it is a non-singular point and the tangent line L to C at p intersects C at p with multiplicity at least 3. Inflection points can be found using the Hessian determinant

$$\det\left(\frac{\partial f}{\partial x_i \partial x_j}\right)_{i,j=0,1,2}$$

To compute partial derivatives, you can search the manual and find the function `jacobian`.

```
i37 : jacobian ideal Plimacon

o37 = {1} | 4x3+4xy2-12x2z-4y2z+6xz2 |
      {1} | 4x2y+4y3-8xyz-2yz2       |
      {1} | -4x3-4xy2+6x2z-2y2z      |

                            3                    1
o37 : Matrix (QQ[x, y, z])  <--- (QQ[x, y, z])
```

```
i38 : jacobian transpose oo

o38 = {-3} | 12x2+4y2-24xz+6z2 8xy-8yz          -12x2-4y2+12xz |
      {-3} | 8xy-8yz           4x2+12y2-8xz-2z2 -8xy-4yz       |
      {-3} | -12x2-4y2+12xz    -8xy-4yz         6x2-2y2        |

                           3                    3
o38 : Matrix (QQ[x, y, z])  <--- (QQ[x, y, z])

i39 : hess = det oo

            6       4 2       2 4       6        5         3 2           4         4 2
o39 = - 288x  - 864x y  - 864x y  - 288y  + 1152x z + 2304x y z + 1152x*y z - 1440x z  -
      ------------------------------------------------------------------------------
          3 3         2 3       2 4      2 4
      576x z  + 576x*y z  + 216x z  - 72y z

o39 : QQ[x, y, z]
```

Next, compute the points where the Hessian determinant vanishes on the curve.

```
i41 : J = ideal Plimacon + ideal hess; decompose J

o41 : Ideal of QQ[x, y, z]

                                    2    2                    2     2
o42 = {ideal (y, x), ideal (z, x  + y ), ideal (x - 3z, y  + 5z )}

o42 : List
```

Among the obtained list, the first two correspond to singular points. The last one corresponds to two inflection points.

EXERCISE 10. (1) Find the inflection points of Cayley's sextic curve.

(2) Let $f = x-2(xz+y^2)y-(xz+y^2)^2z$ and $g = y+(xz+y^2)z$. Compute the Jacobian determinant of the automorphism of $\mathbb{C}[x,y,z]$ defined by $x \mapsto f$, $y \mapsto g$, $z \mapsto z$ (Keywords: Nagata's automorphism, Jacobian conjecture).

CHAPTER 4

Frobenius Maps, Kunz's Theorem, and Peskine-Szpiro's Theorem – Using Ring Homomorphisms and Modules

1. Frobenius Maps and Kunz's Theorem

In this chapter, we will practice handling ring homomorphisms and modules in Macaulay2 using Frobenius maps and Kunz's theorem as examples. First, let's consider a surface called Whitney's umbrella in characteristic 3.

```
i89 : whitney = ZZ/3[x,y,z, Degrees => {3,2,2}]/(x^2-y^2*z);
```

For later use, we set the degrees of x, y, z to $3, 2, 2$ to make it a graded ring.

EXERCISE 11. Use `singularLocus` to confirm that the singular locus of this is $x = y = 0$.

Let's calculate this singular locus in another way.

In general, for a ring R of characteristic p, the Frobenius map $F : R \to R$, $f \mapsto f^p$ is a ring homomorphism. Note that the Frobenius map of the prime field $\mathbb{F}_p$ is the identity map. In Macaulay2, a ring homomorphism is defined as follows:

```
i90 : whitney' = ZZ/3[x,y,z, Degrees => {9,6,6}]/(x^2-y^2*z);

i91 : use whitney; F = map(whitney, whitney', {x^3,y^3,z^3})

                                2    3   3
o92 = map(whitney,whitney',{x*y z, y , z })

o92 : RingMap whitney <--- whitney'

i95 : isWellDefined F

o95 = true

i96 : isHomogeneous F

o96 = true
```

For later use, we prepared another ring `whitney'` with changed degrees to ensure the ring homomorphism preserves degrees. We define the ring homomorphism by specifying the images of x, y, z as x^7, y^7, z^7. However, for quotient rings, we need to separately check if it is well-defined. We also confirmed that it is a homogeneous homomorphism using `isHomogeneous`.

Kunz's theorem states that **"a ring R is regular (non-singular) $\Leftrightarrow$ the Frobenius map is flat"**. This means that for the map $F : R \to R$, when considering the right R as a module over the left R (i.e., push-forward F_*R), it is flat (locally free). There is a function `pushForward` to compute the push-forward of a module, which can be applied when the ring,

ring homomorphism, and module are all homogeneous. (Even if not homogeneous, there is a function on my website to compute the pushforward by the Frobenius map (with the coefficient field being a prime field). URL: http://www4.math.sci.osaka-u.ac.jp/ takehikoya-suda/codes/MyPackage.m2) Let's compute F_*R.

```
i97 : FR = pushForward (F, whitney^1)

o97 = cokernel {0} | 0  0  0  0  0  0  0  0  0  0  0  0  0  0  |
               {3} | x  0  0  0  0  0  0  yz 0  0  0  0  0  0  |
               {5} | 0  x  0  0  0  0  0  0  yz 0  0  0  0  0  |
               {7} | 0  0  0  0  0  0  -z 0  0  0  0  0  0  -x |
               {7} | 0  0  0  -y 0  0  0  0  0  0  -x 0  0  0  |
               {9} | 0  0  0  0  -y 0  0  0  0  0  0  -x 0  0  |
               {5} | 0  0  -y 0  0  0  0  0  0  -x 0  0  0  0  |
               {7} | 0  0  0  0  0  -y 0  0  0  0  0  0  -x 0  |
               {2} | 0  0  x  0  0  0  0  0  0  yz 0  0  0  0  |
               {4} | 0  0  0  x  0  0  0  0  0  0  yz 0  0  0  |
               {6} | 0  0  0  0  x  0  0  0  0  0  0  yz 0  0  |
               {8} | 0  -y 0  0  0  0  0  0  -x 0  0  0  0  0  |
               {4} | 0  0  0  0  0  x  0  0  0  0  0  0  yz 0  |
               {6} | -y 0  0  0  0  0  0  -x 0  0  0  0  0  0  |
               {2} | 0  0  0  0  0  0  0  0  0  0  0  0  0  0  |
               {4} | 0  0  0  0  0  0  x  0  0  0  0  0  0  y2 |

                                                    16
o97 : whitney'-module, quotient of whitney'
```

`whitney^1` represents a rank 1 free module over the ring `whitney`. The matrix represents the map between free modules, and the module is displayed as the cokernel.

EXERCISE 12. Use M2 to confirm that the rank of this module is $3^2 = 9$.

If we can identify the places where this module `FR` is not flat, it should coincide with the singular locus. For this, we can use Fitting ideals. In general, for a module M and a non-negative integer i, the i-th Fitting ideal of M determines the places where M is not locally generated by i elements. In our example, F_*R has rank $3^2 = 9$, so the 9th Fitting ideal determines the places where it is not flat.

```
i98 : radical fittingIdeal(9,FR)

o98 = ideal (y, x)

o98 : Ideal of whitney'
```

The expected result was obtained. This calculation takes time. If you are using a less powerful computer, the calculation may not finish quickly. In that case, press **Control+C twice to interrupt the calculation**.

EXERCISE 13. (1) Use M2 and Kunz's theorem to verify the location of the singularity for the cusp curve $y^2 = x^3$ in characteristic 3.

(2) For the surface $y(y-x)(y-2x)-xz^2=0$ (a simple elliptic singularity) in characteristic 0, construct the module of differential forms $\Omega_{R/k}$ in M2. Also, compute $\bigwedge^2 \Omega_{R/k}$. (Hint: Use `jacobian` and `exteriorPower`.)

2. Peskine-Szpiro's Theorem

THEOREM 1 (Peskine-Szpiro's Theorem). *Let R be a Noetherian ring of positive characteristic, and M a finitely generated R-module. Assume that the projective dimension of M is finite. Then,* $\operatorname{Tor}_i^R(M, F_*R) = 0$ *for* $i > 0$.

Let's verify this with a concrete example.

```
i1 : cusp = ZZ/3[x,y,Degrees=>{2,3}]/(y^2-x^3); cusp' = newRing(cusp,Degrees=>{6,9});
```

We define the cusp singularity. Then, we define the Frobenius map as before and compute F_*R (intermediate steps are omitted).

```
i8 : FR = pushForward(F,cusp^1)

o8 = cokernel {0} | y  0  0  x2 0  0  |
              {2} | 0  y  0  0  x2 0  |
              {4} | 0  0  y  0  0  x2 |
              {7} | 0  0  -x 0  0  -y |
              {5} | 0  -x 0  0  -y 0  |
              {3} | -x 0  0  -y 0  0  |

                                     6
o8 : cusp'-module, quotient of cusp'
```

Here, we consider two modules.

```
i9 : use cusp'; M1 = coker matrix{{x,y}}; M2 = coker matrix{{x}};

i15 : res(M1,LengthLimit => 10)

           1          2          2          2          2          2          2
o15 = cusp'  <--- cusp'  <--- cusp'  <--- cusp'  <--- cusp'  <--- cusp'  <--- cusp'  <--

      0          1          2          3          4          5          6          7

o15 : ChainComplex

i16 : res(M2,LengthLimit => 10)

           1          1
o16 = cusp'  <--- cusp'  <--- 0

      0          1          2

o16 : ChainComplex
```

It is inferred that `M1` has infinite projective dimension, while `M2` has a projective dimension of 1. **Note that if `LengthLimit` is not set, the free resolution is truncated at the length**

of the number of variables. The same caution applies to `pdim`, which calculates the projective dimension. This is not a concern when considering polynomial rings.

EXERCISE 14.
(1) Refer to a textbook on commutative algebra to understand why `M2` has infinite projective dimension.
(2) Calculate the Tor modules of the two modules and verify that Peskine-Szpiro's theorem holds. Refer to the manual for Tor calculations.
(3) For a ChainComplex `C`, you can view the boundary operators with `C.dd`. Observe the periodicity of the boundary operators in the free resolution of `M1`.
(4) Perform similar calculations for simple singularities in dimension 3 in characteristic 2.

CHAPTER 5

Powers of Ideals and Associated Prime Ideals – Creating Functions

In this chapter, we will learn basic programming in M2. Trying to learn programming systematically from the beginning can be time-consuming and discouraging. Therefore, in this lecture, we will keep the preparation to a minimum and aim to get accustomed to practical examples quickly. Those who want to learn more can read the manual or imitate the examples in [**2, 5**] to gradually improve their skills.

1. Creating Functions

In research, you often encounter situations where you need to repeat similar calculations with slightly different data. Repeating the same operations every time can be tedious, so it's better to have the computer do it automatically. For this, you can write **functions**. Programming in M2 essentially involves writing many functions.

First, let's set up the calculation target.

```
i36 : R = ZZ/3[t,x,y]/(x^4+t*x^2*y^2+y^4);

i41 : I = ideal(x,y);

o41 : Ideal of R
```

(This example is taken from the paper [**6**].)

Exercise 15. Use `isPrime` to confirm that `R` is an integral domain and `I` is a prime ideal.

Now, let's write a simple function. Define a function to calculate the square as shown below.

```
i17 : square = i -> i^2

o17 = square

o17 : FunctionClosure
```

`square` is the function name, and it returns `i^2` when given `i`. A **new function is defined as follows:**

```
functionName = input -> output
```

Function names and variable names (not polynomial ring variables, but names given to calculation targets like `cusp`) can use (half-width) alphanumeric characters and apostrophes "'". The first character must not be a number. Apart from naming rings or modules as `R` or `M`, it is customary to start with lowercase letters and use uppercase letters to indicate word boundaries, like `fittingIdeal`.

The defined function can be used as follows.

```
i21 : square(-3)

o21 = 9

i42 : square I

               2        2
o42 = ideal (x , x*y, y )

o42 : Ideal of R
```

EXERCISE 16. (1) Create a function to find the remainder when a given integer is divided by 5.*
(2) Create a function to compute the square of the radical of a given ideal.

2. Composition of functions

By combining multiple functions, you can define more complex functions.

You can compose functions you have defined yourself, but here we will use the existing function `associatedPrimes` to find the associated prime ideals of an ideal. A function that squares an ideal and then finds its associated prime ideals can be defined as follows, though it is a bit verbose.

```
i46 : f = I -> (
      J := square I;
      associatedPrimes J);

i47 : f I

o47 = {ideal (y, x)}

o47 : List
```

In this function, we first compute the square of `I` and assign it to the local variable (valid only within the function) `J` with `J :=`. This does not become the function's output and continues to the next calculation with a semicolon `;`. The result of the final `associatedPrimes J` becomes the function's output.

The same function can also be defined using the function composition operator `@@` as follows.

```
i50 : f = associatedPrimes @@ square;
```

However, I personally do not use this method often. For slightly more complex functions, it seems easier to define local variables one after another with semicolons rather than using `@@`.

EXERCISE 17. (1) Write a function to find the irreducible components of the singular locus of a given quotient ring.*
(2) Write a function to compute the Hessian determinant from a polynomial.
(3) Create a function to find the number of associated prime ideals of a given ideal. (Hint: What is the function to find the number of elements in a list?)
(4) Create a function to find the dimension of the support of a given module.

3. Multivariable Functions

You can also define functions with multiple inputs. Let's write a function to find the associated prime ideals of the n-th power of an ideal.

```
i55 : assPrimePower = (I,n) -> associatedPrimes (I^n);

i57 : assPrimePower (I,3)

o57 = {ideal (y, x)}

o57 : List
```

Exercise 18. Use this function to investigate the associated prime ideals that appear as you vary n. (According to Brodmann, the set of associated prime ideals that appear becomes constant as n increases.)

Next, let's consider Frobenius powers instead of ordinary powers. The e-th Frobenius power of an ideal I, denoted $I^{[p^e]}$, is the ideal generated by f^{p^e} for $f \in I$. If I is generated by $f_1, \ldots, f_l$, then $I^{[p^e]}$ is generated by $f_1^{p^e}, \ldots, f_l^{p^e}$. Let's write a function to compute the Frobenius power.

```
i69 : frobeniusPower = (I,e) -> (
      p := char ring I;
      G := flatten entries gens I;
      G' := apply(G, i -> i^(p^e));
      ideal G');
```

`G` is the list of generators of the ideal. `apply(list, function)` **returns a list where the function is applied to each element of the list.**

Exercise 19.
(1) Check the manual to understand the functions used in the above definition and confirm that they work as expected. In particular, try `apply` with your own examples. (The function `gens` is an abbreviation for `generators`.)
(2) Define a function to find the associated prime ideals of Frobenius powers, similar to `assPrimePower`. Then, confirm that the number of associated prime ideals increases as e becomes larger. (According to [**6**], the number of associated prime ideals that appear is known to be infinite. The first example of an ideal with infinitely many associated prime ideals in Frobenius powers was given by Katzman.)

4. Saving and Reusing Functions

You would want to use the functions you created even after restarting a Macaulay2 session. To do this, create a text file like the one below and save it in an appropriate location with a name like MyFuncs.m2.

```
assPrimePower = (I,n) -> associatedPrimes (I^n);

frobeniusPower = (I,e) -> (
      p := char ring I;
      G := flatten entries gens I;
      G' := apply(G, i -> i^(p^e));
      ideal G');
```

Then, after restarting the session, you can load the file to use the functions.

```
i7 : load "/Users/highernash/MyFuncs.m2"
```

If you save the file in the directory where Macaulay2 is running, which you can check with `currentDirectory()`, you can simply use `load "MyFuncs.m2"`.

EXERCISE 20. Try it out.

CHAPTER 6

$\mathbb{Q}$-Cartier Divisors – More Functions, and if, for, while

1. if, for, while – Conditional Branching and Loops

Like many programming languages, M2 also has basic constructs like if, for, and while. You can use "if" for conditional branching and "for" and "while" for loops to repeat the same calculations. Let's briefly look at how to use these.

1.0.1. *if.* Given an ideal $I \subset R$, a function that finds the fraction field of the integral domain R/I if I is a prime ideal, and returns an error message otherwise, can be written as follows.

```
i24 : quotField = I -> (
      if isPrime I
      then frac(ring I / I)
      else "This is not a prime ideal.");
```

Note that the line breaks are for readability and are not necessary. In the structure "if p then x else y", if p evaluates to true, x is evaluated and returned; if false, y is evaluated and returned. The else part is optional. (If omitted and p is fdalse, null is returned.)

Exercise 21. Create a function that, given an input k, returns the univariate polynomial ring $k[t]$ if k is a field, and "This is not a field" otherwise.

1.0.2. *for.* First, an example.

```
i5 : for i from 1 to 10 list 2^i

o5 = {2, 4, 8, 16, 32, 64, 128, 256, 512, 1024}

o5 : List
```

As you can see, this creates a list of 2^i for i from 1 to 10.

Another example.

```
i11 : j = 1; for i to 10 do (j = j^2+1); j

o13 = 20673320404242167718163471873403688937727437938335771679340586582331709536877565726
      78163591448431217381755798837087854896124558264182613121663664140479479045161051919
      21136113094251414331751648388702965701984820995989371335396043070504172130112866206
      80326165917911353727803752577858443670237676134236078659942965754297741698914181600
```

Starting from $j = 1$, this operation squares it and adds 1, repeating this 11 times. (If "from" is omitted, it starts from 0, so it repeats 11 times.)

Exercise 22. (1) Create a list of prime numbers under 300. (Hint: continue)*

(2) Write a function that, given two univariate polynomials f and g, returns the pair of polynomials f', g' obtained after applying the Euclidean algorithm n times. If the greatest common divisor is obtained within n steps, return it.

1.0.3. *while.*

```
i22 : i = 0; while i < 10^10 list (i=i^2+1;i)

o23 = {1, 2, 5, 26, 677, 458330, 210066388901}

o23 : List
```

Starting with $i = 0$, as long as $i < 10^{10}$, it repeats the operation of squaring i and adding 1, returning the list of resulting numbers. The structure "while p list x do y" evaluates x and lists the result as long as the condition p is satisfied, and then executes y.

In the next example, the condition part is set to true, so the condition is always satisfied. The loop is exited with if ... break.

```
i30 : smoothSingularCenter = I -> (
      R := ring I;
      currentI := I;
      while true do (
        singI := radical ideal singularLocus currentI;
        if singI == ideal (1_R) then break;
        currentI = singI);
      currentI);
```

EXERCISE 23. Understand what this function is calculating.

2. 2-Cartier Check

Consider the following quadratic cone and its ideal.

```
i101 : qCone = QQ[x,y,z]/ideal(x*y-z^2)

o101 = qCone

o101 : QuotientRing

i102 : I = radical ideal(y)

o102 = ideal (z, y)

o102 : Ideal of qCone
```

The irreducible divisor D defined by this ideal is not Cartier. This can be confirmed as follows.

```
i105 : M = module I

o105 = image | z y |

                                       1
o105 : qCone-module, submodule of qCone

i106 : radical fittingIdeal(1,M)
```

```
o106 = ideal (z, y, x)

o106 : Ideal of qCone
```

It is found that the ideal, viewed as a module, is not flat at the origin. However, $2D$ is Cartier. This can be confirmed by checking that $(M^{\otimes 2})^{\vee\vee}$ is flat as follows.

```
i117 : dual dual (M**M)

o117 = image {1} | 1 |

                                          1
o117 : qCone-module, submodule of qCone
```

EXERCISE 24. Confirm that the irreducible divisor defined by the ideal $\sqrt{(x+y)} \subset R$ in the cubic Fermat hypersurface $R = \mathbb{C}[x,y,z]/(x^3+y^3+z^3)$ is not 2-Cartier.

Write a function `is2Cartier` to check if an ideal `I` defining a divisor is 2-Cartier. The function checks if the ideal is 2-Cartier using the same operations as above.

```
i10 :  is2Cartier = I -> (
       M := module I;
       M2 := M**M;
       DDM2 := dual dual M2;
       J := fittingIdeal(1,DDM2);
       R := ring M;
       J == ideal 1_R);
```

In this way, it is good to assign intermediate calculation results to easily understandable variable names with `:=` and proceed with the calculations step by step. The final `==` checks if both sides are equal, returning a Bool value (`true` or `false`). If both sides are ideals of the same ring, it judges them as equal if they are the same ideal, even if their representations (generators) are different. Here, using Gröbner bases allows this, but for the theoretical background, please refer to an appropriate textbook.

Below are examples of applying the above function.

```
i11 : is2Cartier I

o11 = true

i23 : R = QQ[x,y,z]/ideal(x*y-z^5); I = radical ideal y;

o24 : Ideal of R

i25 : is2Cartier I

o25 = false
```

3. Calculating the Cartier Index

Next, let's write a function to check sequentially whether an ideal is r-Cartier for each natural number r.

```
i40 :  cartierIndex = (I,N) -> (
  M := module I;
  r:=1;
  while r <= N do (
    Mr := M^**r;
    DDMr := dual dual Mr;
    J := fittingIdeal(1,DDMr);
    R := ring M;
    if J == ideal 1_R then break;
    r = r+1);
  if r <= N then r else 0);

i41 : cartierIndex(I,6)

o41 = 5
```

The function above uses a `while` **loop**. It has the form

`while condition do Z`

As long as `condition` is satisfied, `Z` continues to be evaluated. In the example above, starting from $r = 1$, r is incremented by 1 each time the loop runs. If $r = N + 1$ is reached or if it becomes r-Cartier in the middle, the loop is exited with `break`. After the loop ends, if $r \leq N$, the value of r is returned; if $r = N + 1$, 0 is returned.

EXERCISE 25. (1) Rewrite `cartierIndex` using `for` instead of `while`.
(2) Write a function to find the Gorenstein index of a given normal quotient ring (or normal projective variety in the next chapter).
(3) Given an algebraic variety $X \subset \mathbb{C}^n$, write a function that repeatedly projects X via the map $\mathbb{C}^n \ni (x_1, \ldots, x_n) \mapsto (x_1, \ldots, x_{n-1}) \in \mathbb{C}^{n-1}$ and finds the first image that becomes singular. (Projection corresponds to variable elimination algebraically.)*

CHAPTER 7

Resolution of Curve Singularities – Algorithms

1. Resolution of a Cusp Singularity

The blowup of Spec$R$ at an ideal $I \subset R$ is defined as Proj$(\bigoplus_{n=0}^{\infty} I^n)$. The ring that appears here (a graded R-algebra) is called the Rees algebra. Let's compute the blowup at the origin of a cusp.

```
i2 : R = ZZ/101[x,y]/(y^2-x^3);

i3 : reesAlgebra ideal(x,y)

                  R[w , w ]
                     0   1
o3 = -------------------------------------
                   2              2    2
     (y*w  - x*w , x w  - y*w , x*w  - w )
        0      1     0      1     0    1

o3 : QuotientRing
```

As it is, it is difficult to examine the properties of the blowup (such as whether it is non-singular), so let's take an affine cover. In this example, it is covered by two affine open sets, each obtained by setting $w_i = 1$. For example, setting $w_0 = 1$:

```
i6 : Bl = o3; use Bl; R0 = Bl / ideal(w_0 - 1)

o8 = R0

o8 : QuotientRing

i9 : describe R0

                   R[w , w ]
                      0   1
      -------------------------------------
                    2              2    2
      (y*w  - x*w , x w  - y*w , x*w  - w )
         0      1     0      1     0    1
o9 = -------------------------------------
                     w  - 1
                      0
```

```
i27 : minimalPresentation R0

       ZZ
o27 = ---[w ]
      101  1

o27 : PolynomialRing
```

This affine open set is isomorphic to an affine line, and in particular, it is non-singular.

For the data of the resolution of singularities, we need not only the coordinate rings of the affine open sets but also the data of the maps from the affine open sets to the original curve.

```
i35 : f = map(R0,R)

o35 = map(R0,R,{x, y})

o35 : RingMap R0 <--- R

i42 : g = R0.minimalPresentationMap

            ZZ                  2   3
o42 = map(---[w ],R0,{1, w , w , w })
          101  1          1   1   1

                ZZ
o42 : RingMap ---[w ] <--- R0
              101  1

i43 : g*f

            ZZ         2   3
o43 = map(---[w ],R,{w , w })
          101  1      1   1

                ZZ
o43 : RingMap ---[w ] <--- R
              101  1
```

EXERCISE 26. Given an ideal $I \subset R$, create a function that returns a list of homomorphisms $R \to S_i$ for the coordinate rings S_i of the affine cover of the blowup at I. Use `minimalPresentation`*

2. Creating and Implementing Algorithms

By blowing up and computing the affine charts, and blowing up again if singularities remain, you can resolve the singularities. However, to compute this with a computer, you need to write down the detailed steps more precisely. Instead of starting to write the program immediately, let's first write down the algorithm in Japanese.

Input: Quotient ring R (integral domain)
Output: A collection of ring homomorphisms $f_i : R \to R_i$ such that $\operatorname{Spec} R_i$ is an affine cover of some resolution of singularities, and f_i is determined by that resolution.
Algorithm:

(1) Set SmoothCharts $= \emptyset$, SingularCharts $= \emptyset$, WaitingCharts $= \{(R, id_R)\}$.
(2) If WaitingCharts is empty, go to step 4. Otherwise, select the first $f : R \to S$ from WaitingCharts and remove it from WaitingCharts. Compute the radical ideal $J \subset S$ defining the singular locus. If $J = (1)$, add f to SmoothCharts; otherwise, add (f, J) to SingularCharts. Go to step 3.
(3) If SingularCharts is empty, go to step 2. Otherwise, select one (f, I) from SingularCharts and remove it from SingularCharts. Blow up S at I. Compute the affine cover and the maps. Add all the obtained maps to WaitingCharts.
(4) Output SmoothCharts.

Exercise 27. Implement the above algorithm as a function in M2.*

Exercise 28. Modify the created function as follows:

(1) Blow up at one maximal ideal at a time.
(2) Record how many times each affine chart was obtained by blowing up.

CHAPTER 8

Projective Varieties and Sheaf Cohomology

1. Hodge Numbers

As a subject for calculation, let's consider the Fermat quartic hypersurface $(x_0^4+\cdots+x_3^4 = 0) \subset \mathbb{P}^3$. Let's write its homogeneous coordinate ring in a slightly different way than before.

```
i5 : R = QQ[x_0..x_3]/(sum(4,i->x_i^4))

o5 = R

o5 : QuotientRing
```

For polynomial rings with many variables, you can write them more concisely using subscripts. `sum(n,f(i))` calculates $f(0) + f(1) + \cdots + f(n-1)$.

Now, given a homogeneous ring, you can create the corresponding projective variety with `Proj`.

```
i6 : X = Proj R

o6 = X

o6 : ProjectiveVariety
```

While it can be easier to handle the ring directly, for calculations involving sheaf cohomology, it is better to use the projective variety.

First, let's confirm that X is non-singular.

```
i7 : singularLocus X

         /QQ[x , x , x , x ]\
         |    0   1   2   3 |
o7 = Proj|-------------------|
         \         1        /

o7 : ProjectiveVariety
```

Since the denominator is 1, we know that the singular locus is empty.

Let's calculate the Hodge numbers of X. You can use `hh^(p,q)` for this. To find all the Hodge numbers at once, do the following.

```
i10 : for p to 2 list (for q to 2 list hh^(p,q)(X))

o10 = {{1, 0, 1}, {0, 20, 0}, {1, 0, 1}}

o10 : List
```

This calculates the dimensions of $H^q(\Omega^p_X)$. By taking appropriate alternating sums of these, you can find the topological Euler characteristic, which can be calculated in M2 with `euler`. However, you must confirm for yourself that X is non-singular for this to be valid. The arithmetic genus can also be calculated with `genus`, and this does not require X to be non-singular (I believe).

EXERCISE 29. (1) Change the dimension, degree, and defining equations of the variety and calculate the Hodge numbers. Observe which examples finish quickly and which do not.
(2) Improve `hh` to return an error message if the variety is singular.

2. Cohomology of Coherent Sheaves

Several standard coherent sheaves can be easily created from a projective variety. First, the (twisted) structure sheaf.

```
i13 : OO_X

o13 = OO
       X

o13 : SheafOfRings

i14 : OO_X(3)

         1
o14 = OO  (3)
       X

o14 : coherent sheaf on X, free
```

The notation should be self-explanatory.

Next, the tangent sheaf, cotangent sheaf, and exterior powers of the cotangent sheaf.

```
i20 : TX = tangentSheaf X

o20 = image {-2} | -x_0x_2^3   x_1x_2^3     x_2^4+x_3^4 0           x_1x_3^3     -x_0x_3
            {-2} | x_1^4+x_2^4 x_0^3x_1     x_0^3x_2    -x_1x_3^3   0            x_2x_3^
            {-2} | x_0x_1^3    -x_1^4-x_3^4 -x_1^3x_2   -x_0x_3^3   x_2x_3^3     0
            {-2} | -x_1^3x_3   -x_0^3x_3    0           x_2^4+x_3^4 x_0^3x_2     x_1^3x_
            {-2} | x_2^3x_3    0            x_0^3x_3    x_1x_2^3    x_0^3x_1     x_1^4+x
            {-2} | 0           x_2^3x_3     -x_1^3x_3   x_0x_2^3    -x_1^4-x_2^4 x_0x_1^

                                                 6
o20 : coherent sheaf on X, subsheaf of OO  (2)
                                       X

i21 : cotangentSheaf X;

i22 : cotangentSheaf(2,X);
```

To twist any coherent sheaf, you can use, for example, `TX(3)` (the tensor product of the tangent sheaf and $\mathcal{O}_X(3)$).

You can create new coherent sheaves from the obtained sheaves F and G using `F ++ G`, `F ** G`, `F ^ n`, `exteriorPower(n,F)`, `dual F`, `sheafHom(F,G)`, and `sheafExt^n(F,G)`. The first two mean direct sum and tensor product, and n copies of the direct sum. The rest should be inferred from the notation.

The cohomology groups of coherent sheaves can be calculated with `HH^n`.

```
i130 : HH^1(TX)

         20
o130 = QQ

o130 : QQ-module, free
```

EXERCISE 30. Verify Serre's duality theorem, vanishing theorem, and Kodaira's vanishing theorem with examples of your choice.

3. Creating Coherent Sheaves from Modules

If you want to create more coherent sheaves, you can do so from graded modules over the homogeneous coordinate ring. As an example, let's construct the structure sheaf of the intersection $H_1 \cap H_2$ of two general hyperplane sections $H_1, H_2 \subset X$ (as an $\mathcal{O}_X$-module). Randomly take two degree 1 homogeneous elements from the homogeneous coordinate ring R as follows.

```
i31 : f=random(1,R)

           1     5
o31 = x  + -x  + -x  + 8x
       0   2 1   8 2     3

o31 : R

i32 : g=random(1,R);
```

Next, define the cokernel of the map $R^2 \to R$ defined by the matrix (f, g), and take the associated coherent sheaf to obtain the desired result.

```
i34 : M = coker matrix{{f,g}}

o34 = cokernel | x_0+1/2x_1+5/8x_2+8x_3 1/5x_0+7/3x_1+2/9x_2+5/4x_3 |

                                1
o34 : R-module, quotient of R

i38 : F = sheaf M

o38 = cokernel | x_0+1/2x_1+5/8x_2+8x_3 1/5x_0+7/3x_1+2/9x_2+5/4x_3 |
```

```
o38 : coherent sheaf on X, quotient of OO
                                          X

i39 : HH^0(F)

        4
o39 = QQ

o39 : QQ-module, free
```

EXERCISE 31. (1) Create a module by taking only the parts of the coordinate ring R of degree 3 or higher. Use `truncate`.
(2) Create a coherent sheaf `F` from this module and calculate `HH^0(F(*))`.
(3) Compare with `module F`.

EXERCISE 32. Show that the canonical sheaf $\omega_X = \Omega_X^2$ of X treated in this chapter is a trivial invertible sheaf without using the adjunction formula, following these steps:
(1) Create a corresponding graded module over the homogeneous coordinate ring from ω_X using `module`.
(2) Use this module to find the restriction of ω_X to a general hyperplane.
(3) Show that the restriction of ω_X to the hyperplane has degree 0 using the Riemann-Roch formula and cohomology calculations.
(4) Confirm that $\dim H^0(\omega_X) = 1$.

EXERCISE 33. (Research Task) Create a function to determine whether two given coherent sheaves are isomorphic. (There seems to be an algorithm for finite-dimensional modules, so you might be able to do this by referring to it: [**4**][**1**])

CHAPTER 9

Miscellaneous

In this chapter, we will touch on some topics that were not covered in the lectures but might be good to know. For more details, please read the "The Macaulay2 language" section of the manual.

1. Hash Tables

A hash table is a collection of key-value pairs, where you can specify a key to retrieve its corresponding value.

```
i1 : legends = new HashTable from
     {"Hilbert" => "David", "Euler"=>"Leonhard", "Newton"=>"Issac"}

o1 = HashTable{Euler => Leonhard}
               Hilbert => David
               Newton => Issac

o1 : HashTable

i2 : legends#"Hilbert"

o2 = David
```

Exercise 34. Create a hash table with countries as keys and their capitals as values.

2. Using Packages

Functions that handle specialized calculations are organized into packages. In version 1.4 of M2, the packages ConwayPolynomials, Elimination, IntegralClosure, LLLBases, PrimaryDecomposition, ReesAlgebra, and TangentCone are loaded at startup, so you can use the functions included in these packages. However, note that the descriptions of these functions are in the respective package sections of the manual and are not listed in the Index of Macaulay2Doc.

If you want to use packages other than these, you need to load them with a command. For example, to use the Depth package, which deals with depth and Cohen-Macaulay properties, you can check if a quotient ring is Cohen-Macaulay as follows:

```
i1 : loadPackage "Depth";

i3 : isCM (QQ[x,y]/ideal(x*y))

o3 = true
```

```
i4 : isCM (QQ[x,y,z]/intersect(ideal(x,y),ideal(z)))

o4 = false
```

You can also create your own packages. I don't know how to do it, but it's written in the manual.

3. Classes

In M2, everything has a **class (type)**.

```
i5 : class (QQ[x])

o5 = PolynomialRing

o5 : Type

i6 : class {1,2,3}

o6 = List

o6 : Type
```

The behavior of functions depends on the class of the variables (inputs). For example, the function `ideal` returns the ideal that defines a quotient ring when applied to a quotient ring, and the ideal generated by a polynomial when applied to a polynomial.

You can also create your own classes. A good example is in "Macaulay2Doc > mathematical examples > Tutorial: Divisors," where a new class for handling divisors on algebraic varieties is created.

4. Reading and Writing Files

You can write calculation results to a file or read a file and apply functions to it. However, I have never done it. Please read the manual.

CHAPTER 10

Exercise Solutions

Exercise 5.2

Use the function `degree`.

Exercise 6.1

```
i10 : QQ[x,y]; I = ideal(x*y,x^2);

o11 : Ideal of QQ[x, y]

i12 : I == radical I

o12 = false
```

Exercise 9.1

(Example) Move an element f of the polynomial ring in two variables to the polynomial ring in three variables using `substitute`, and homogenize it with the new variable z using `substitute(f,z)`:

```
i13 : QQ[x,y]; f = x^2+y+1;

i15 : f = substitute(f,QQ[x,y,z]);

i16 : homogenize (f,z)

       2          2
o16 = x  + y*z + z

o16 : QQ[x, y, z]
```

Exercise 9.3

The conductor ideal of the integral closure map defines the non-normal locus. However, since the function `conductor` can only be used in the homogeneous case, we consider the homogenized Roman surface.

```
i36 : homoRoman = QQ[x,y,z,w]/(x^2*y^2+y^2*z^2+z^2*x^2+x*y*z*w);

i37 : I = conductor icMap homoRoman

o37 = ideal (y*z, x*z, x*y)
```

By setting $w = 1$, we obtain the desired ideal.

Exercise 16.1

```
i1 : div5 = n -> n % 5;
```

Exercise 17.1

```
i32 : singularComponents = R -> (
        singR := singularLocus R;
        singIdeal := ideal singR;
        decompose singIdeal);
```

Exercise 22.1

```
i2 : for i to 300 list (if isPrime i then i else continue)

o2 = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61,
     ------------------------------------------------------------------------
     67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137,
     ------------------------------------------------------------------------
     139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211,
     ------------------------------------------------------------------------
     223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283,
     ------------------------------------------------------------------------
     293}

o2 : List
```

Exercise 25.3

```
isSmooth = I -> (
     R := ring I;
     singI := radical ideal singularLocus I;
     singI == ideal 1_R);

singularProjection = I -> (
     R := ring I;
     J := I;
     while true do (
          R := ring J;
          Vs := flatten entries vars R;
          if length Vs == 1 then break;
          lastVar := last Vs;
         J = radical eliminate(lastVar,J);
         Vs' := drop(Vs,-1);
         KK = coefficientRing R;
         R = KK[Vs'];
         J = sub(J,R);
         if not isSmooth J then break;
         );
```

```
     J);
```

Exercise 26

```
blowupCharts = {Variable => w} >> o -> I -> (
     R := ring I;
     rees := reesAlgebra(I,Variable => o.Variable);
     reesAmb := ambient rees;
     G := gens reesAmb;
     l := length G;
     charts := for w in G list rees/ideal(w-1_rees);
     RToCharts := for T in charts list map(T,R);
     for T in charts list (minimalPresentation T);
     minMaps := for T in charts list T.minimalPresentationMap;
     for i to l-1 list ((minMaps_i) * (RToCharts_i))
     );
```

For the next exercise, the variable name can be changed with an option.

Exercise 27

```
desing = R -> (
     smCharts := {};
     singCharts := {};
     waitingCharts := {id_R};
     numBlowups := 0;

  while waitingCharts != {} do (
   f := waitingCharts_0;
   waitingCharts = drop(waitingCharts,1);
   S := target f;
   singIdeal := radical(sub(ideal singularLocus S,S));

   if singIdeal == ideal(1_S)
     then smCharts = append(smCharts,f)
     else singCharts = append(singCharts, (f,singIdeal));

  while singCharts != {} do (
       (g,J) := singCharts_0;
       singCharts = drop(singCharts,1);
       numBlowups = numBlowups + 1;

       newMaps = for h in blowupCharts(J,Variable=> vars(numBlowups))
                            list (h*g);
       waitingCharts = join(waitingCharts, newMaps);
       );

       );
```

```
    smCharts
    );
```

Bibliography

[1] P. A. Brooksbank and E. M. Luks. Testing isomorphism of modules. **J. Algebra**, 320(11):4020–4029, 2008.

[2] D. Eisenbud, D. R. Grayson, M. Stillman, and B. Sturmfels, editors. **Computations in algebraic geometry with Macaulay 2**, volume 8 of **Algorithms and Computation in Mathematics**. Springer-Verlag, Berlin, 2002.

[3] D. R. Grayson and M. E. Stillman. Macaulay 2, a software system for research in algebraic geometry. Available at http://www.math.uiuc.edu/Macaulay2/.

[4] K. M. Lux and M. Szőke. Computing decompositions of modules over finite-dimensional algebras. **Experiment. Math.**, 16(1):1–6, 2007.

[5] H. Schenck. **Computational algebraic geometry**, volume 58 of **London Mathematical Society Student Texts**. Cambridge University Press, Cambridge, 2003.

[6] A. K. Singh and I. Swanson. Associated primes of local cohomology modules and of Frobenius powers. **Int. Math. Res. Not.**, (33):1703–1733, 2004.

[7] H. Yokota. Macaulay2 no shōkai. Available at http://durian2.math.kobe-u.ac.jp/KnoppixMathdoc/ponpoko/Macaulay2.pdf