9 Polynomial Systems

This chapter completes the two preceding ones. The objects we consider are systems of equations in several variables, as in Chapter 8. These equations, as in Chapter 7, are polynomial. Compared to univariate polynomials, those with several variables yield nice mathematical properties, but also new difficulties, related in particular to the fact that the ring $K[x_1, \ldots, x_n]$ is not principal. The theory of Gröbner bases provides tools to overcome this limitation. In the end, we have at our disposal powerful methods to study polynomial systems, with uncountable applications in various domains.

A large part of the chapter only requires basic knowledge on multivariate polynomials. Some parts are however at the level of a commutative algebra course of third or fourth year at university. For more details, a very good and complete reference is the book of Cox, Little and O'Shea [CLO07].

9.1 Polynomials in Several Variables

9.1.1 The Rings $A[x_1, \ldots, x_n]$

We consider here polynomials in several indeterminates or variables, also called multivariate polynomials.

Similarly to other algebraic structures available in Sage, before being able to construct polynomials, we have to define a family of indeterminates living all in the same ring. The syntax is almost the same as with a single variable (cf. §7.1.1):

```
sage: R = PolynomialRing(QQ, 'x,y,z')
sage: x,y,z = R.gens() # gives the tuples of indeterminates
```

or in short:

```
sage: R.<x,y,z> = QQ[]
```

(or even `R = QQ['x,y,z']`). The `PolynomialRing` constructor also allows to create a family of indeterminates with the same name, and integer indices:

```
sage: R = PolynomialRing(QQ, 'x', 10)
```

Assigning the n-tuple returned by `gens` to the variable `x` then allows to easily access the indeterminate x_i via `x[i]`:

```
sage: x = R.gens()
sage: sum(x[i] for i in xrange(5))
x0 + x1 + x2 + x3 + x4
```

The order of the variables matters. The comparison with `==` between `QQ['x,y']` and `QQ['y,x']` returns false, and a given polynomial prints differently if seen as element of the former or of the latter:

```
sage: def test_poly(ring, deg=3):
....:     monomials = Subsets(
....:         flatten([(x,)*deg for x in (1,) + ring.gens()]),
....:         deg, submultiset=True)
....:     return add(mul(m) for m in monomials)
```

```
sage: test_poly(QQ['x,y'])
x^3 + x^2*y + x*y^2 + y^3 + x^2 + x*y + y^2 + x + y + 1
sage: test_poly(QQ['y,x'])
y^3 + y^2*x + y*x^2 + x^3 + y^2 + y*x + x^2 + y + x + 1
sage: test_poly(QQ['x,y']) == test_poly(QQ['y,x'])
True
```

Exercise 33. Explain the behaviour of the `test_poly` function defined above.

More generally, writing polynomials in canonical form requires choosing a way to order their monomials. Ordering them by degree is natural for univariate polynomials, however for multivariate polynomials, no monomial order is satisfactory in all cases. Therefore Sage allows us to choose between several orders, thanks to the `order` option of `PolynomialRing`. For example, the `deglex` order first ranks monomials according to their total degree, then by lexicographic order of the degrees of indeterminates in case of same total degree:

```
sage: test_poly(PolynomialRing(QQ, 'x,y', order='deglex'))
x^3 + x^2*y + x*y^2 + y^3 + x^2 + x*y + y^2 + x + y + 1
```

The main available orders are described in more detail in §9.3.1. We will see that the choice of the monomial order does not only determine the output, but also matters for some computations.

Exercise 34. Define the ring $\mathbb{Q}[x_2, x_3, \ldots, x_{37}]$ whose indeterminates are indexed by prime numbers less than 40, and the variables `x2`, `x3`, ..., `x37` to access the indeterminates.

Finally, it can be useful, in some cases, to play with multivariate polynomials in *recursive representation*, i.e., seen as elements of a polynomial ring with coefficients that are themselves polynomials (see the sidebar on page 128).

Construction of polynomial rings	
ring $A[x,y]$	`PolynomialRing(A, 'x,y')` *or* `A['x,y']`
ring $A[x_0,\dots,x_{n-1}]$	`PolynomialRing(A, 'x', n)`
ring $A[x_0,x_1,\dots,y_0,y_1,\dots]$	`InfinitePolynomialRing(A, ['x','y'])`
n-tuple of generators	`R.gens()`
1st, 2nd... generator	`R.0`, `R.1`, ...
indeterminates of $R = A[x,y][z][\dots]$	`R.variable_names_recursive()`
conversion $A[x_1,x_2,y] \to A[x_1,x_2][y]$	`p.polynomial(y)`

Access to coefficients	
support, non-zero coefficients	`p.exponents()`, `p.coefficients()`
coefficient of a monomial	`p[x^2*y]` *or* `p[2,1]`
degree (total, in x, partial)	`p.degree()`, `p.degree(x)`, `p.degrees()`
leading monomial/coefficient/term	`p.lm()`, `p.lc()`, `p.lt()`

Basic operations	
transformation of coefficients	`p.map_coefficients(f)`
partial derivative $\mathrm{d}/\mathrm{d}x$	`p.derivative(x)`
evaluation $p(x,y)\|_{x=a,y=b}$	`p.subs(x=a, y=b)` *or* `p(x=a, y=b)`
homogenisation	`p.homogenize()`
common denominator ($p \in \mathbb{Q}[x,y,\dots]$)	`p.denominator()`

TABLE 9.1 – Multivariate polynomials.

9.1.2 Polynomials

Just as univariate polynomials are in the class `Polynomial`, multivariate polynomials (in rings with a finite number of variables) are in the class `MPolynomial`[1]. For the usual base rings (like $\mathbb{Z}$, $\mathbb{Q}$ or $\mathbb{F}_q$), Sage calls the Singular computer algebra system, which is specialised in fast polynomial computations. In the other cases, a generic and much slower implementation is used.

Multivariate polynomials are always encoded in sparse representation[2]. Why this choice? A dense polynomial with n variables of total degree d contains $\binom{n+d}{d}$ monomials: for $n = d = 10$, it amounts to 184 756 coefficients to store! It is thus very difficult to manipulate large dense polynomials like we do with univariate ones. Besides, even when the polynomials are dense, the supports (the exponents of non-zero monomials) encountered in practice have various forms. If for example a polynomial with n variables, dense up to the total degree $d-1$, is represented by a rectangular array $d \times \cdots \times d$, for large d, only about one coefficient over $n!$ is non-zero. On the contrary, the sparse representation by dictionary is well adapted to the shape of the support, and also to the monomial order.

[1] Contrary to `Polynomial`, this class is not directly available from the command line: we have to use its full name. For example, we can check whether an object is of type "multivariate polynomial" by `isinstance(p, sage.rings.polynomial.multi_polynomial.MPolynomial)`.

[2] The recursive representation (see sidebar page 128) yields nevertheless partially dense polynomials. In the memory representation of a polynomial from $A[x][y]$, each coefficient of y^k occupies (in general) a space proportional to its degree in x, to which we should add a space proportional to the degree in y for the polynomial itself.

The rings $A[(x_n, y_n, \dots)_{n\in\mathbb{N}}]$

It happens that we do not know, at the beginning of a computation, how many variables will be necessary. This makes the use of `PolynomialRing` rather painful: we first have to compute in a first domain, then extend it and convert all elements each time we want to introduce a new variable.

Polynomial rings with an infinite number of variables provide a more flexible data structure. Their elements can contain variables in one or several infinite families of indeterminates. Each generator of the ring corresponds not only to a single variable, but to a family of variables indexed by integers:

```
sage: R.<x,y> = InfinitePolynomialRing(ZZ, order='lex')
sage: p = mul(x[k] - y[k] for k in range(2)); p
x_1*x_0 - x_1*y_0 - x_0*y_1 + y_1*y_0
sage: p + x[100]
x_100 + x_1*x_0 - x_1*y_0 - x_0*y_1 + y_1*y_0
```

We get back to some usual polynomial ring `PolynomialRing` thanks to the `polynomial` method, which returns the image of an element from `InfinitePolynomialRing` in a sufficiently large ring to contain all elements of the ring with an infinite number of variables which have been produced so far. The obtained ring is generally not the smallest one with this property.

As a counterpart of this facility, these rings are less efficient than the rings `PolynomialRing`. Also, their *ideals* cannot replace those of usual polynomial rings for computations on polynomial systems, which is the main topic of this chapter.

9.1.3 Basic Operations

Let us fix the terminology. Let $R = A[x_1, \dots, x_n]$ be a polynomial ring. We call *monomial* an expression of the form $x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$, i.e., a product of indeterminates, and we note it $\boldsymbol{x}^{\boldsymbol{\alpha}}$ in short. The integer n-tuple $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \dots, \alpha_n)$ is the *exponent* of the monomial $\boldsymbol{x}^{\boldsymbol{\alpha}}$. A *term* is a monomial multiplied by an element of A, its *coefficient*.

Since there is no unique way to order the terms, the elements of R do not have, as mathematical objects, a dominant coefficient. However, once an order has been chosen at the ring construction, it is possible and useful to define a leading monomial, the leftmost one in the writing order. The methods `lm`, `lc` and `lt` of a multivariate polynomial return respectively its leading monomial, its leading coefficient, and the term they form together:

```
sage: R.<x,y,z> = QQ[]
sage: p = 7*y^2*x^2 + 3*y*x^2 + 2*y*z + x^3 + 6
sage: p.lt()
7*x^2*y^2
```

The arithmetic operations `+`, `-` and `*`, as well as the methods `coefficients`, `dict`, and several others, work like their univariate variants. Among the small

differences, the square-bracket operator `[]` to extract a coefficient accepts as parameter either a monomial, or its exponent:

```
sage: p[x^2*y] == p[(2,1,0)] == p[2,1,0] == 3
True
```

Likewise, the evaluation of a polynomial requires giving values to all variables, or to make explicit those to substitute:

```
sage: p(0, 3, -1)
0
sage: p.subs(x = 1, z = x^2+1)
2*x^2*y + 7*y^2 + 5*y + 7
```

The `subs` method might also substitute any number of variables at once, see its documentation for advanced examples. The degree might be either total or partial:

```
sage: print("total={d}    (in x)={dx}    partial={ds}"\
....:   .format(d=p.degree(), dx=p.degree(x), ds=p.degrees()))
total=4    (in x)=3    partial=(3, 2, 1)
```

Other constructions get trivial adaptations, for example, the `derivative` method takes as parameter the variable with respect to which we want to differentiate.

9.1.4 Arithmetic

Beyond syntactic and elementary arithmetic operations, available functions in Sage are in general limited to polynomials over a field, and sometimes over $\mathbb{Z}$ or $\mathbb{Z}/n\mathbb{Z}$. For the rest of this chapter, unless otherwise noted, we will consider polynomials over a field.

The Euclidean division of polynomials makes sense only in one variable. In Sage, the `quo_rem` method and the associated operators `//` and `%` remain nonetheless defined for multivariate polynomials. The "division with remainder" they compute satisfies

```
(p//q)*q + (p%q) == p
```

and matches the Euclidean division when p and q depend on one variable only, but it is not a Euclidean division and it is not canonical. It is however useful when the division is exact, or when the divisor is a monomial. In the other cases, we will prefer to `quo_rem` and its variants the `mod` method, described in §9.2.3, which reduces a polynomial modulo an ideal while taking into account the monomial order of the ring:

```
sage: R.<x,y> = QQ[]; p = x^2 + y^2; q = x + y
sage: print("({quo})*({q}) + ({rem}) == {p}".format( \
....:         quo=p//q, q=q, rem=p%q, p=p//q*q+p%q))
(-x + y)*(x + y) + (2*x^2) == x^2 + y^2
sage: p.mod(q)  # is NOT equivalent to p%q
2*y^2
```

Operations on multivariate polynomials	
divisibility $p \mid q$	`p.divides(q)`
factorisation	`p.factor()`
gcd, lcm	`p.gcd(q)`, `p.lcm(q)`
square-free test	`p.is_squarefree()`
resultant $\mathrm{Res}_x(p, q)$	`p.resultant(q, x)`

TABLE 9.2 – Arithmetic.

The methods `divides`, `gcd`, `lcm` or `factor` have the same meaning as in one variable. Since multivariate polynomial rings are not Euclidean in general, the first ones are not available for arbitrary coefficient rings, but work on usual fields, for example on number fields:

```
sage: R.<x,y> = QQ[exp(2*I*pi/5)][]
sage: (x^10 + y^5).gcd(x^4 - y^2)
x^2 + y
sage: (x^10 + y^5).factor()
(x^2 + y) * (x^2 + (a^3)*y) * (x^2 + (a^2)*y) * (x^2 + (a)*y) * (x^2 +
    (-a^3 - a^2 - a - 1)*y)
```

9.2 Polynomial Systems and Ideals

We now consider the central topic of this chapter. Sections 9.2.1 and 9.2.2 give an overview of the different ways to find and understand the solutions of a system of polynomial equations with the help of Sage. Section 9.2.3 is devoted to ideals associated to these systems. The last sections come back in a more detailed manner on algebraic elimination and system solving tools.

9.2.1 A First Example

Let us revisit the polynomial system from Section 2.2,

$$\begin{cases} x^2yz = 18 \\ xy^3z = 24 \\ xyz^4 = 6. \end{cases} \tag{9.1}$$

The `solve()` function from Sage was only able to find numerical solutions. Let us now see how Sage is able to solve the system exactly, and, with a little help from the user, to find simple closed forms for all solutions[3].

Enumerating Solutions. Let us first translate the problem in more algebraic terms, by constructing the ideal of $\mathbb{Q}[x, y, z]$ generated by the equations:

[3]Our purpose being here to illustrate the tools to solve polynomial systems, we neglect the possibility of reducing (9.1) to linear equations by taking the logarithm!

```
sage: R.<x,y,z> = QQ[]
sage: J = R.ideal(x^2 * y * z - 18,
....:             x * y^3 * z - 24,
....:             x * y * z^4 - 6)
```

As we will see in Section 9.2.3, the following command enables us to check the ideal J is of dimension zero, i.e., the system (9.1) has a finite number of solutions in $\mathbb{C}^3$:

```
sage: J.dimension()
0
```

Once this is established, the first reflex should be to call the method `variety`, which computes all solutions of the system. Without any parameter, it gives the solutions in the base field of the polynomial ring:

```
sage: J.variety()
[{y: 2, z: 1, x: 3}]
```

The solution $(3, 2, 1)$ already found is thus the unique rational solution.

The next step is to enumerate the complex solutions. To perform this exactly, we work in the field of algebraic numbers. We find again the 17 solutions:

```
sage: V = J.variety(QQbar)
sage: len(V)
17
```

Explicitly, the last three have the following form:

```
sage: V[-3:]
[{z: 0.9324722294043558? - 0.3612416661871530?*I,
 y: -1.700434271459229? + 1.052864325754712?*I,
 x: 1.337215067329615? - 2.685489874065187?*I},
{z: 0.9324722294043558? + 0.3612416661871530?*I,
 y: -1.700434271459229? - 1.052864325754712?*I,
 x: 1.337215067329615? + 2.685489874065187?*I},
{z: 1, y: 2, x: 3}]
```

Each solution point is given by a dictionary whose keys are the generators of `QQbar['x,y,z']` (and not of `QQ['x,y,z']`, which explains the short detour below), and the associated coordinates of the point. Except for the rational solution already identified, the first coordinates are all algebraic numbers of degree 16:

```
sage: (xx, yy, zz) = QQbar['x,y,z'].gens()
sage: [ pt[xx].degree() for pt in V ]
[16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 1]
```

Computing with the Solutions and Identifying Their Structure. We have obtained an exact representation of the complex solutions from the system (9.1), however this representation is not really explicit. This is not a problem: having the coordinates as elements of `QQbar` is enough to pursue exact computations on these solutions.

For example, it is not difficult to see that if (x, y, z) is solution of the system (9.1), then so is $(|x|, |y|, |z|)$. Let us build the set of $(|x|, |y|, |z|)$ for (x, y, z) solution:

```
sage: Set(tuple(abs(pt[i]) for i in (xx,yy,zz)) for pt in V)
{(3, 2, 1)}
```

All the values of x (resp. y, z) have thus the same modulus. Even more, we can check that the substitution

$$(x, y, z) \mapsto (\omega x, \omega^9 y, \omega^6 z) \qquad \text{where} \quad \omega = e^{2\pi i/17} \tag{9.2}$$

leaves the system invariant. In particular, the last coordinates of the solutions are exactly the seventeenth roots of unity, which we check again thanks to the possibility to compute exactly with algebraic numbers:

```
sage: w = QQbar.zeta(17); w  # primitive root of 1
0.9324722294043558? + 0.3612416661871530?*I
sage: Set(pt[zz] for pt in V) == Set(w^i for i in range(17))
True
```

The solutions of the system are therefore the triples $(3\omega, 2\omega^9, \omega^6)$ for $\omega^{17} = 1$. This is much more explicit!

Exercise 35. Look for real solutions (and not only rational ones) of (9.1), to check directly there is only $(x = 3, y = 2, z = 1)$. Find again the substitution (9.2), including the value 17 for the order of ω as root of unity, by a computation with Sage.

We could have reached the same result by examining the minimal polynomials of the coordinates of points of V. We see indeed that a given coordinate has the same minimal polynomial for all solution points, apart from $(3, 2, 1)$. The common minimal polynomial of the third coordinates is nothing else than the cyclotomic polynomial Φ_{17}:

```
sage: set(pt[zz].minpoly() for pt in V[:-1])
{x^16 + x^15 + x^14 + x^13 + x^12 + x^11 + x^10 + x^9 + x^8 + x^7 + x^6
    + x^5 + x^4 + x^3 + x^2 + x + 1}
```

Those of the first and second coordinate are respectively $3^{16} \cdot \Phi_{17}(x/3)$ and $2^{16} \cdot \Phi_{17}(x/2)$.

Closed-Form Expressions. Getting an explicit form of the solutions is thus possible using the exponential notation of complex numbers:

```
sage: def polar_form(z):
....:     rho = z.abs(); rho.simplify()
....:     theta = 2 * pi * z.rational_argument()
....:     return (SR(rho) * exp(I*theta))
sage: [tuple(polar_form(pt[i]) for i in [xx,yy,zz]) for pt in V[-3:]]
[(3*e^(-6/17*I*pi), 2*e^(14/17*I*pi), e^(-2/17*I*pi)),
(3*e^(6/17*I*pi), 2*e^(-14/17*I*pi), e^(2/17*I*pi)), (3, 2, 1)]
```

Naturally, if we had had the idea of writing the elements of V in exponential notation, this would have been enough to conclude.

Simplifying the System. A different approach is possible. Instead of looking for solutions, let us try to compute a simpler form of the system itself. The fundamental tools offered by Sage for this purpose are the triangular decomposition and Gröbner bases. We will see later exactly what they compute; let us first use them on this example:

```
sage: J.triangular_decomposition()
[Ideal (z^17 - 1, y - 2*z^10, x - 3*z^3) of Multivariate
Polynomial Ring in x, y, z over Rational Field]
sage: J.transformed_basis()
[z^17 - 1, -2*z^10 + y, -3/4*y^2 + x]
```

We obtain in both cases the equivalent system

$$z^{17} = 1 \qquad y = 2z^{10} \qquad x = 3z^3,$$

or $x = 3y^2/4$ for the last equation with `transformed_basis`, i.e., $V = \{(3\omega^3, 2\omega^{10}, \omega) \mid \omega^{17} = 1\}$. This is an immediate parametrisation of the compact form of solutions found manually above.

9.2.2 What Does Solving Mean?

A polynomial system that has solutions often has an infinite number of solutions. The simple equation $x^2 - y = 0$ has an infinite number of solutions in $\mathbb{Q}^2$, not even considering $\mathbb{R}^2$ or $\mathbb{C}^2$. It is therefore not possible to enumerate them. The best we can do is to describe the set of solutions "as explicitly as possible", i.e., compute a representation of it from which we can easily extract useful information. The situation is analogous to linear systems, for which (in the homogeneous case) a basis of the system kernel is a good description of the set of solutions.

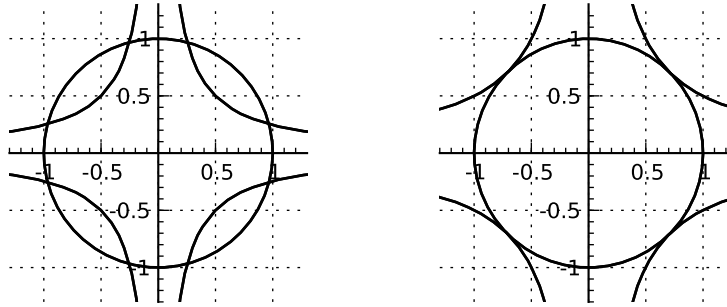
In the particular case where the number of solutions is finite, it becomes possible to "compute them". However, even in that case, do we want to enumerate the solutions in $\mathbb{Q}$, or in a finite field $\mathbb{F}_q$? To find real or complex numerical approximations? Or even, as in the example of last section, to represent them using algebraic numbers, i.e., to compute for example minimal polynomials of their coordinates?

This very example illustrates the fact that other representations of the set of solutions might be much more useful than a list of points, more so when the solutions are numerous. Therefore, enumerating the solutions is not always the best thing to do, even when possible. In the end, we do not really want to compute the solutions, but we want to compute *with* them, to deduce afterwards, according to the given problem, the information we are really interested in. The rest of this chapter investigates several useful tools for this purpose.

9.2.3 Ideals and Systems

If s polynomials $p_1, \ldots, p_s \in K[\boldsymbol{x}]$ vanish at a point $\boldsymbol{x}$ with coordinates in K or an extension of K, any element of the ideal they generate also vanishes at $\boldsymbol{x}$. It is thus natural to associate to the polynomial system

$$p_1(\boldsymbol{x}) = p_2(\boldsymbol{x}) = \cdots = p_s(\boldsymbol{x}) = 0$$



(a) (S_1) (b) (S_2)

```
sage: opts = {'axes':True, 'gridlines':True, 'frame':False,
....:     'aspect_ratio':1, 'axes_pad':0, 'fontsize':8,
....:     'xmin':-1.3, 'xmax':1.3, 'ymin':-1.3, 'ymax':1.3}
sage: (ideal(J.0).plot() + ideal(J.1).plot()).show(**opts)
```

FIGURE 9.1 – Intersection of two plane curves, see systems (9.3).

the ideal $J = \langle p_1, \dots, p_s \rangle \subset K[\boldsymbol{x}]$. Two polynomial systems generating the same ideal are equivalent in the sense that they share the same solutions. If L is a field containing K, we call *algebraic subvariety* of L^n associated to J the set

$$V_L(J) = \{\boldsymbol{x} \in L^n \mid \forall p \in J, p(\boldsymbol{x}) = 0\} = \{\boldsymbol{x} \in L^n \mid p_1(\boldsymbol{x}) = \cdots = p_s(\boldsymbol{x}) = 0\}$$

of solutions of the system with coordinates in L. Different ideals may have the same associated variety. For example, the equations $x = 0$ and $x^2 = 0$ have the same unique solution in $\mathbb{C}$, although we have $\langle x^2 \rangle \subsetneq \langle x \rangle$. The ideal generated by a polynomial system captures rather the notion of "solutions with multiplicities" (in the algebraic closure of K).

For instance, the following two systems both express the intersection of the unit circle and a curve of equation $\alpha\, x^2y^2 = 1$, union of two equilateral hyperbolas (see Figure 9.1):

$$(S_1) \begin{cases} x^2 + y^2 = 1 \\ 16\, x^2y^2 = 1 \end{cases} \qquad (S_2) \begin{cases} x^2 + y^2 = 1 \\ 4\, x^2y^2 = 1. \end{cases} \tag{9.3}$$

The system (S_1) has eight solutions in $\mathbb{C}$, all with real coordinates. When we deform it into (S_2) by varying the parameter α, the two solutions on each branch of the hyperbola move closer until they match. The system (S_2) has then only four solutions, each one in some sense of "multiplicity two". By decreasing α further, we would have no more real solution, but eight complex solutions.

Computing Modulo an Ideal. Just as for univariate polynomials, Sage allows us to define ideals[4] $J \subset K[\boldsymbol{x}]$, quotient rings $K[\boldsymbol{x}]/J$, and to compute naturally

[4]Warning: the objects `InfinitePolynomialRing` also have an `ideal` method, however it does not have the same meaning as for usual polynomial ring. (An ideal of $K[(x_n)_{n\in\mathbb{N}}]$ has no reason

with elements of these quotient rings. The ideal J_1 associated to (S_1) is built with:

```
sage: R.<x,y> = QQ[]
sage: J = R.ideal(x^2 + y^2 - 1, 16*x^2*y^2 - 1)
```

We can then form the quotient of $K[\boldsymbol{x}]$ by J_1, and project polynomials on it, compute with equivalence classes modulo J_1, and "lift" them into representatives:

```
sage: ybar2 = R.quo(J)(y^2)
sage: [ybar2^i for i in range(3)]
[1, ybar^2, ybar^2 - 1/16]
sage: ((ybar2 + 1)^2).lift()
3*y^2 + 15/16
```

There is a theoretical issue here. The elements of $K[\boldsymbol{x}]/J$ are represented in normal form, which is necessary to be able to check the equality between two elements. However, this normal form is not easy to define, for the reason already mentioned in §9.1.4: the division of a representative of an equivalence class $p + J$ by a principal generator of J, used to compute in $K[x]/J$, has no direct analogue in several variables. Let us admit for now that a normal form nevertheless exists, which depends on the order on the elements chosen at the ring construction, and builds on a particular system of generators of J called Gröbner basis. Section 9.3 at the end of this chapter defines Gröbner bases and shows how to use them in computations. Sage automatically computes Gröbner bases when required; however, these computations are sometimes very expensive, in particular when the number of variables is large, which might make computations in a quotient ring somewhat difficult.

Let us go back to playing with Sage. When $p \in J$, the command `p.lift(J)` rewrites p as a linear combination of generators of J with polynomial coefficients:

```
sage: u = (16*y^4 - 16*y^2 + 1).lift(J); u
[16*y^2, -1]
sage: u[0]*J.0 + u[1]*J.1
16*y^4 - 16*y^2 + 1
```

For any polynomial p, the expression `p.mod(J)` gives the normal form of p modulo J, seen as element of $K[\boldsymbol{x}]$:

```
sage: (y^4).mod(J)
y^2 - 1/16
```

Beware: while `J.reduce(p)` is equivalent to `p.mod(J)`, the variant `p.reduce([p1, p2, ...])` returns a representative of $p + J$ which is not necessarily the normal form (see §9.3.2):

```
sage: (y^4).reduce([x^2 + y^2 - 1, 16*x^2*y^2 - 1])
y^4
```

By combining `p.mod(J)` and `p.lift(J)`, we can decompose a polynomial p into a linear combination of generators of J with polynomial coefficients, plus a remainder which is zero if and only if $p \in J$.

to be finitely generated!) The rest of the chapter does not apply to these objects.

Ideals	
ideal $\langle p_1, p_2\rangle \subset R$	`R.ideal(p1, p2)` *or* `(p1, p2)*R`
sum, product, power	`I + J`, `I * J`, `I^k`
intersection $I \cap J$	`I.intersection(J)`
quotient $I : J = \{p \mid pJ \subset I\}$	`I.quotient(J)`
radical $\sqrt{J}$	`J.radical()`
reduction modulo J	`J.reduce(p)` *or* `p.mod(J)`
section of $R \twoheadrightarrow R/J$	`p.lift(J)`
quotient ring R/J	`R.quo(J)`
homogenised ideal	`J.homogenize()`

Some predefined ideals	
irrelevant ideal $\langle x_1, \ldots, x_n\rangle$	`R.irrelevant_ideal()`
Jacobian ideal $\langle \partial p/\partial x_i\rangle_i$	`p.jacobian_ideal()`
cyclic roots (9.11)	`sage.rings.ideal.Cyclic(R)`
field equations $x_i^q = x_i$	`sage.rings.ideal.FieldIdeal(GF(q)['x1,x2'])`

TABLE 9.3 – Ideals.

Radical of an Ideal and Solutions. The main correspondence between ideals and varieties lies in Hilbert's theorem of zeros, also known as *Nullstellensatz*. Let $\bar{K}$ be an algebraic closure of K.

THEOREM (*Nullstellensatz*). Let $p_1, \ldots, p_s \in K[\boldsymbol{x}]$, and let $Z \subset \bar{K}^n$ be the set of common zeros to the p_i. A polynomial $p \in K[\boldsymbol{x}]$ vanishes identically on Z if and only if there exists an integer k such that $p^k \in \langle p_1, \ldots, p_s\rangle$.

This result provides an algebraic criterion to check whether a polynomial system has some solutions. The constant polynomial 1 vanishes identically on Z if and only if Z is empty, thus the system $p_1(\boldsymbol{x}) = \cdots = p_s(\boldsymbol{x}) = 0$ has solutions in $\bar{K}$ if and only if the ideal $\langle p_1, \ldots, p_s\rangle$ does not contain 1. For example, the circles of radius 1 centered at $(0, 0)$ and $(4, 0)$ have a complex intersection:

```
sage: 1 in ideal(x^2+y^2-1, (x-4)^2+y^2-1)
False
```

However, after adding the condition $x = y$, the system does not have any solutions. We can give a trivial proof that it is inconsistent by exhibiting as *certificate* a combination of the equations that reduces to $1 = 0$ if they are satisfied. The computation

```
sage: R(1).lift(ideal(x^2+y^2-1, (x-4)^2+y^2-1, x-y))
[-1/28*y + 1/14, 1/28*y + 1/14, -1/7*x + 1/7*y + 4/7]
```

yields in our case the relation

$$\frac{1}{28}\Big((-y+2)(x^2+y^2-1) + (y+2)\big((x-4)^2+y^2-1\big) + (-4x+4y+16)(x-y)\Big) = 1.$$

In terms of ideals, the *Nullstellensatz* says that the set of polynomials vanishing identically on the variety $V_{\bar{K}}(J)$ associated to the ideal J is the *radical* of that ideal, defined by

$$\sqrt{J} = \{p \in K[\boldsymbol{x}] \mid \exists k \in \mathbb{N}, p^k \in J\}.$$

We have

$$V_{\bar{K}}(\sqrt{J}) = V_{\bar{K}}(J)$$

but intuitively, switching to the radical "forgets the multiplicities". Therefore, the ideal J_1 is its own radical (we say it is radical), whereas the ideal J_2 associated to (S_2) satisfies $J_2 \subsetneq \sqrt{J_2}$:

```
sage: J1 = (x^2 + y^2 - 1, 16*x^2*y^2 - 1)*R
sage: J2 = (x^2 + y^2 - 1,  4*x^2*y^2 - 1)*R
sage: J1.radical() == J1
True
sage: J2.radical()
Ideal (2*y^2 - 1, 2*x^2 - 1) of Multivariate Polynomial
Ring in x, y over Rational Field
sage: 2*y^2 - 1 in J2
False
```

Systems, ideals and cryptography

Some specific modules, `sage.rings.polynomial.multi_polynomial_sequence` and `sage.crypto.mq`, provide tools to manipulate polynomial systems by taking into account the particular form of equations, and not only the ideal they generate. This is useful to play with large structured systems, like those found in cryptography. The module `sage.crypto` also defines several polynomial systems associated to classical cryptographic constructions.

Operations on Ideals. It is also possible to compute with ideals themselves. Let us recall the definition of the sum of two ideals:

$$I + J = \{p + q \mid p \in I \text{ and } q \in J\} = \langle I \cup J \rangle.$$

It corresponds geometrically to the intersection of varieties:

$$V(I + J) = V(I) \cap V(J).$$

Hence, the ideal J_1 associated to (S_1) is the sum of $C = \langle x^2 + y^2 - 1 \rangle$ and $H = \langle 16\, x^2 y^2 - 1 \rangle$, which define respectively the circle and the double hyperbola. In Sage:

```
sage: C = ideal(x^2 + y^2 - 1); H = ideal(16*x^2*y^2 - 1)
sage: C + H == J1
True
```

This equality test is also based on computing a Gröbner basis.

Similarly, the intersection, the product and the quotient of ideals satisfy

$$\begin{aligned} I \cap J &= \{p \mid p \in I \text{ and } p \in J\} & V(I \cap J) &= V(I) \cup V(J) \\ I \cdot J &= \langle pq \mid p \in I, q \in J \rangle & V(I \cdot J) &= V(I) \cup V(J) \\ I : J &= \{p \mid pJ \subset I\} & V(I : J) &= \overline{V(I) \setminus V(J)} \end{aligned}$$

and are computed as indicated in Table 9.3. The notation $\bar{X}$ designs here the *Zariski closure* of X, i.e., the smallest algebraic variety containing X. For example, the curve of Figure 9.1a is the set of zeros of polynomials from $C \cap H$, and the quotient $(C \cap H) : \langle 4xy - 1 \rangle$ corresponds to the union of the circle with one of the two hyperbolas:

```
sage: CH = C.intersection(H).quotient(ideal(4*x*y-1)); CH
Ideal (4*x^3*y + 4*x*y^3 + x^2 - 4*x*y + y^2 - 1) of
Multivariate Polynomial Ring in x, y over Rational Field
sage: CH.gen(0).factor()
(4*x*y + 1) * (x^2 + y^2 - 1)
```

However, the curve obtained by removing from $V(H)$ a finite number of points is not an algebraic subvariety, so that:

```
sage: H.quotient(C) == H
True
```

Dimension. To each ideal of $J \subset K[\boldsymbol{x}]$ is also associated a *dimension*, which intuitively corresponds to the maximal "dimension" of the "components" of the variety $V(J)$ over an algebraically closed field[5]. We have for example:

```
sage: [J.dimension() for J in [J1, J2, C, H, H*J2, J1+J2]]
[0, 0, 1, 1, 1, -1]
```

Indeed, $V(J_1)$ and $V(J_2)$ have a finite number of points, $V(C)$ and $V(H)$ are curves, $V(H \cdot J_2)$ is the union of curves and isolated points, and $V(J_1 + J_2)$ is empty. The *zero-dimensional* systems, i.e., those that generate an ideal of dimension zero, or equivalently that have a finite number of solutions (over the algebraic closure of K), will be particularly studied in the rest of the chapter, since they are the systems we can "solve" the most explicitly.

9.2.4 Elimination

In a system of equations, *eliminating* a variable means finding "consequences", or better "all consequences", of the system which are independent of this variable. Said otherwise, we want to find equations satisfied by any solution, but which do not contain the eliminated variable, which makes them often easier to analyse.

[5]We give in §9.3.3 a more rigorous (but not necessarily clearer) definition. We refer the reader to the reference given at the beginning of the chapter for better explanations.

General polynomial systems: elimination, geometry	
elimination ideal $J \cap A[z,t] \subset K[x,y,z,t]$	`J.elimination_ideal(x, y)`
resultant $\mathrm{Res}_x(p,q)$	`p.resultant(q, x)`
dimension	`J.dimension()`
genus	`J.genus()`
Zero-dimensional systems	
solutions in $L \supseteq K$	`J.variety(L)`
dimension over K of the quotient	`J.vector_space_dimension()`
quotient basis	`J.normal_basis()`
triangular decomposition	`J.triangular_decomposition()`

TABLE 9.4 – Solving polynomial systems.

For example, we can eliminate x from the linear system

$$\begin{cases} 2x + \ y - 2z = 0 \\ 2x + 2y + \ z = 1 \end{cases} \tag{9.4}$$

by subtracting the first equation from the second one. It yields $y + 3z = 1$, which shows that any solution triple (x, y, z) of (9.4) is of the form $(x, 1 - 3z, z)$. We can then check that every "partial solution" $(1 - 3z, z)$ lifts to a (unique) solution $(\frac{5z-1}{2}, 1 - 3z, z)$ of (9.4). This illustrates that Gauss' pivoting algorithm solves linear systems by elimination, as opposed to, for example, Cramer's formulas.

Elimination Ideals. In the context of polynomial systems, the "consequences" of equations $p_1(\boldsymbol{x}) = \cdots = p_s(\boldsymbol{x}) = 0$ are elements of the ideal $\langle p_1, \ldots, p_s \rangle$. If J is an ideal of $K[x_1, \ldots, x_n]$, we call k-th *elimination ideal* of J the set

$$J_k = J \cap K[x_{k+1}, \ldots, x_n] \tag{9.5}$$

of elements of J which only contain the $n - k$ last variables. This is an ideal of $K[x_{k+1}, \ldots, x_n]$.

In Sage, the method `elimination_ideal` takes as input the list of variables to eliminate. Beware: it does not return $J_k \subset K[x_{k+1}, \ldots, x_n]$, but the ideal $\langle J_k \rangle$ of $K[x_1, \ldots, x_n]$ it generates. In the case of the linear system (9.4), we find

```
sage: R.<x,y,z> = QQ[]
sage: J = ideal(2*x+y-2*z, 2*x+2*y+z-1)
sage: J.elimination_ideal(x)
Ideal (y + 3*z - 1) of Multivariate Polynomial Ring in x, y, z
over Rational Field
sage: J.elimination_ideal([x,y])
Ideal (0) of Multivariate Polynomial Ring in x, y, z over Rational Field
```

Mathematically, we interpret these results as follows: we have $J \cap \mathbb{Q}[y,z] = \langle y + 3z - 1 \rangle \subset \mathbb{Q}[y,z]$ and $J \cap \mathbb{Q}[z] = \mathbb{Q}[z]$, i.e., $\mathbb{Q}[z] \subset J$. (Indeed, the ideal $\langle 0 \rangle$

corresponds to the system reduced to the sole trivial equation $0 = 0$, of which any polynomial is solution.) This is clearly not a recommended way of solving linear systems: the specific tools discussed in Chapter 8 are much more efficient!

For a slightly less trivial example, let us go back to the system (S_1) from Section 9.2.3 (see Figure 9.1a):

```
sage: R.<x,y> = QQ[]
sage: J1 = ideal(x^2 + y^2 - 1, 16*x^2*y^2 - 1)
```

Eliminating y yields an ideal of $\mathbb{Q}[x]$ — therefore principal — generated by a polynomial g whose roots are the abscissas

$$\frac{\pm\sqrt{2 \pm \sqrt{3}}}{2}$$

of the eight solutions of (S_1):

```
sage: g = J1.elimination_ideal(y).gens(); g
[16*x^4 - 16*x^2 + 1]
sage: SR(g[0]).solve(SR(x)) # solves by radicals
[x == -1/2*sqrt(sqrt(3) + 2), x == 1/2*sqrt(sqrt(3) + 2),
x == -1/2*sqrt(-sqrt(3) + 2), x == 1/2*sqrt(-sqrt(3) + 2)]
```

By re-injecting into (S_1) each of the found values of x, we obtain a (redundant) system of equations in y only, which allows to compute the corresponding values of y.

Eliminating = Projecting. The above example shows that eliminating y in a system corresponds geometrically to the *projection* π of the solution variety on a hyperplane of equation $y =$ constant. However, let us now consider separately the ideals $C = \langle x^2 + y^2 - 1 \rangle$ and $H = \langle 16\,x^2y^2 - 1 \rangle$ whose sum is J_1, and, once again, let us eliminate y:

```
sage: C.elimination_ideal(y).gens()
[0]
sage: H.elimination_ideal(y).gens()
[0]
```

Insofar as C is concerned, this is no surprise. The circle $\{(x, y) \in \mathbb{R}^2 \mid x^2+y^2 = 1\}$ projects on $[-1; 1]$, however it is clear that any value of x can be "re-injected" in the unique equation $x^2 + y^2 - 1 = 0$, and the obtained equation in y has *complex* solutions. The elimination of y in C corresponds to the projection on the first coordinate of the complex circle $\{(x, y) \in \mathbb{C}^2 \mid x^2+y^2 = 1\}$, which is $\mathbb{C}$ altogether.

The case of H is a bit more intricate. The equation $16\,x^2y^2 = 1$ has no solution, even complex, for $x = 0$. We have then

$$V_{\mathbb{C}}(H \cap \mathbb{Q}[x]) = \mathbb{C} \subsetneq \pi(V_{\mathbb{C}}(H)) = \mathbb{C} \setminus \{0\}.$$

Indeed, the projection of the hyperbola, $\mathbb{C} \setminus \{0\}$, is not an algebraic subvariety. Conclusion: elimination really corresponds to projection (over an algebraically closed field), but it does not compute the exact projection, only the Zariski closure of it.

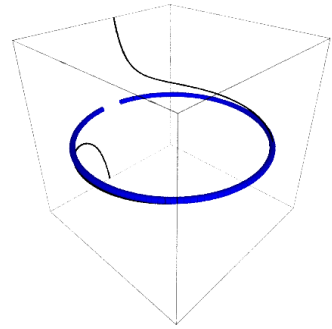


FIGURE 9.2 – A part of the curve in (x, y, t) defined by (9.6) and its projection on the plane $t = 0$.

Applications: Plane Geometry. If $X \subset \mathbb{C}^k$ is given by a rational parametrisation

$$X = \{\big(f_1(\boldsymbol{t}), f_2(\boldsymbol{t}), \ldots, f_k(\boldsymbol{t})\big)\}, \qquad f_1, \ldots, f_k \in \mathbb{Q}(t_1, \ldots, t_n),$$

finding an implicit equation for X consists of projecting the part of $\mathbb{C}^{k+n}$ defined by the equations $x_i = f_i(\boldsymbol{t})$ on the subspace $(x_1, \ldots, x_k) \simeq \mathbb{C}^k$. This is an elimination problem. Let us consider the classical parametrisation of the circle

$$x = \frac{1 - t^2}{1 + t^2} \qquad y = \frac{2t}{1 + t^2} \tag{9.6}$$

associated to the expression of $(\sin\theta, \cos\theta)$ in terms of $\tan(\theta/2)$. It translates into polynomial relations defining an ideal of $\mathbb{Q}[x, y, t]$:

```
sage: R.<x,y,t> = QQ[]
sage: Param = R.ideal((1-t^2)-(1+t^2)*x, 2*t-(1+t^2)*y)
```

Let us eliminate t:

```
sage: Param.elimination_ideal(t).gens()
[x^2 + y^2 - 1]
```

We obtain an equation of the circle. We can notice that this equation vanishes at $(x, y) = (-1, 0)$, although the parametrisation (9.6) does not hit that point, since the circle minus a point is not an algebraic subvariety.

Another example: let us draw a few of the circles $(\mathcal{C}_t)$ of equation

$$\mathcal{C}_t : \qquad x^2 + (y - t)^2 = \frac{t^2 + 1}{2} \tag{9.7}$$

using Sage commands (see Figure 9.3):

```
sage: R.<x,y,t> = QQ[]
sage: eq = x^2 + (y-t)^2 - 1/2*(t^2+1)
```

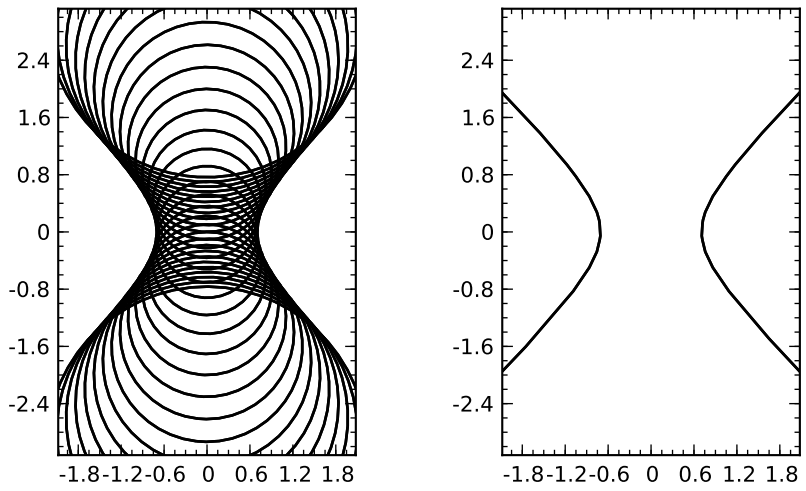


FIGURE 9.3 – A family of circles and its envelope.

```
sage: fig = add((eq(t=k/5)*QQ[x,y]).plot() for k in (-15..15))
sage: fig.show(aspect_ratio=1, xmin=-2, xmax=2, ymin=-3, ymax=3)
```

We see that the *envelope* of the family of circles $(\mathcal{C}_t)$ appears, a "limit curve" tangent to all $\mathcal{C}_t$, which we can describe informally as the set of "intersection points of circles infinitely close" to the family.

More precisely, if f is a differentiable function, and if the curve $\mathcal{C}_t$ is defined by $f(x, y, t) = 0$ for any t, the envelope of $(\mathcal{C}_t)$ is the set of points (x, y) such that

$$\exists t, \qquad f(x, y, t) = 0 \qquad \text{and} \qquad \frac{\partial f}{\partial t}(x, y, t) = 0. \tag{9.8}$$

In the case of circles (9.7), the function $f(x, y, t)$ is a polynomial. Their envelope is the projection on the (x, y) plane of the solutions from (9.8), thus we can determine an equation of it via the following elimination computation:

```
sage: env = ideal(eq, eq.derivative(t)).elimination_ideal(t)
sage: env.gens()
[2*x^2 - 2*y^2 - 1]
```

It remains only to draw the curve found:

```
sage: env.change_ring(QQ[x,y]).plot((x,-2,2),(y,-3,3))
```

Resultant and Elimination. The elimination performed in the preceding examples is implicitly based on Gröbner bases computed automatically by Sage. Yet, we have already encountered in this book another elimination tool: the resultant.

Let us consider two non constant polynomials $p, q \in K[x_1, \ldots, x_n, y]$. We denote by $\mathrm{Res}_y(p, q)$ the resultant of p and q, considered as polynomials in the

Inequalities

Let us consider the triangle with vertices $A = (0, 0)$, $B = (1, 0)$ and $C = (x, y)$. Assume the angles $\widehat{BAC}$ and $\widehat{CBA}$ are equal, and let us try to prove computationally that we have an isosceles triangle. By introducing the parameter $t = \tan \hat{A} = \tan \hat{B}$, the problem is encoded by the equations $y = tx = t(1 - x)$, and we have to show that they imply

$$x^2 + y^2 = (1 - x)^2 + y^2.$$

With Sage, we obtain:

```
sage: R.<x,y,t> = QQ[]
sage: J = (y-t*x, y-t*(1-x))*R
sage: (x^2+y^2) - ((1-x)^2+y^2) in J
False
```

This could have been expected: when $x = y = t = 0$, the hypotheses are satisfied, but the conclusion is false! Geometrically, we have to exclude the case of flat triangles, which can have two equal angles without being isosceles.

How to encode the constraint $t \neq 0$? The trick is to introduce an auxiliary variable u, and to force $tu = 1$. The computation becomes:

```
sage: R.<x,y,t,u> = QQ[]
sage: J = (y-t*x, y-t*(1-x), t*u-1)*R
sage: (x^2+y^2) - ((1-x)^2+y^2) in J
True
```

and we now have the expected result. Let us remark by the way that we can simultaneously force several expressions to not vanish with only one auxiliary variable, using an equation like $t_1 t_2 \cdots t_n u = 1$.

single variable y, with coefficients in $K[x_1, \ldots, x_n]$. We have seen in §7.3.3 that it is a polynomial from $K[x_1, \ldots, x_n]$, which vanishes at $\boldsymbol{u} \in K^n$ if and only if $p(u_1, \ldots, u_n, y)$ and $q(u_1, \ldots, u_n, y)$ (which are two polynomials of $K[y]$) have a common zero, except maybe when the leading coefficients (in y) of p and q themselves vanish at $\boldsymbol{u}$.

Some of our elimination computations involving only two polynomials can be replaced by resultants. For example, the equation of the envelope of circles (9.7) is

```
sage: eq.derivative(t).resultant(eq, t)
x^2 - y^2 - 1/2
```

The resultant $\mathrm{Res}_y(p, q)$ is an element of the elimination ideal $\langle p, q \rangle \cap K[x_1, \ldots, x_n]$. Even in the case $n = 1$ (where the elimination ideal is principal), though, and even if the leading coefficients with respect to y of p and q are coprime, the resultant does not necessarily generate the elimination ideal:

```
sage: R.<x,y> = QQ[]
```

```
sage: p = y^2 - x; q = y^2 + x
sage: p.resultant(q, y)
4*x^2
sage: ideal(p, q).elimination_ideal(y)
Ideal (x) of Multivariate Polynomial Ring in x, y over Rational Field
```

9.2.5 Zero-Dimensional Systems

We can deal with many problems just with the computation of elimination ideals, and Sage does not provide other "black-box" tools to solve general polynomial systems. The situation is somewhat different for zero-dimensional systems.

An ideal $J \subset K[\boldsymbol{x}]$ is said to have *dimension zero* when the quotient $K[\boldsymbol{x}]/J$ is a vector space of finite dimension. *Over an algebraically closed field*, it is equivalent to say that the variety $V(J)$ contains a finite number of points. For example, the systems (9.1) and (9.3) generate ideals of dimension zero — we say they are themselves zero-dimensional. On the contrary, the ideal $\langle (x^2+y^2)(x^2+y^2+1)\rangle$ of $\mathbb{Q}[x,y]$ is of dimension 1, despite its only real solution being $(0,0)$:

```
sage: R.<x,y> = QQ[]
sage: ((x^2 + y^2)*(x^2 + y^2 + 1)*R).dimension()
1
```

Zero-dimensional systems can be solved more explicitly than what is possible with the general tools from the previous section. We have already seen several of these methods in practice on the example of §9.2.1.

Enumerating the Solutions. First, having a finite number of solutions enables us to enumerate them, exactly or approximately.

The Sage expression `J.variety(L)` computes the variety $V_L(J)$. It raises an error if J is not zero-dimensional. By default, it looks for solutions with coordinates in the base field of the polynomial ring over which the system is defined. For example, the subvariety of $\mathbb{Q}^n$ defined by J_1 is empty:

```
sage: R.<x,y> = QQ[]
sage: J1 = (x^2 + y^2 - 1, 16*x^2*y^2 - 1)*R
sage: J1.variety()
[]
```

But, like the `roots` method for univariate polynomials, `variety` works for any kind of domain L. The most important case for now is the field of algebraic numbers. We can indeed show that the solutions of a zero-dimensional system with coefficients in K have coordinates in the algebraic closure of K. Therefore, it is possible to compute exactly the complex variety $V_{\mathbb{C}}(J) = V_{\bar{\mathbb{Q}}}(J)$ associated to an ideal $J \subset \mathbb{Q}[\boldsymbol{x}]$:

```
sage: J1.variety(QQbar)[0:2]
[{y: -0.9659258262890683?, x: -0.2588190451025208?},
{y: -0.9659258262890683?, x: 0.2588190451025208?}]
```

Exercise 36. Show that the solutions of (S_1) have coordinates in $\mathbb{Q}[\sqrt{2-\sqrt{3}}]$, and give them in terms of radicals.

Triangular Decomposition. Internally, `J.variety(L)` goes through a *triangular decomposition* of the ideal J. This decomposition is interesting in itself, since it sometimes gives a description of the variety J which is better for the rest of the computation, or even easier to grasp than the output of `variety` (see §9.2.1), particularly in case of numerous solutions.

A polynomial system is called *triangular* when of the following form

$$\left\{\begin{array}{rcl} p_1(x_1) & := & x_1^{d_1} + a_{1,d_1-1}\, x_1^{d_1-1} + \cdots + a_{1,0} = 0 \\ p_2(x_1,x_2) & := & x_2^{d_2} + a_{2,d_2-1}(x_1)\, x_2^{d_2-1} + \cdots + a_{2,0}(x_1) = 0 \\ & \vdots & \\ p_n(x_1,\ldots,x_n) & := & x_n^{d_n} + a_{n,d_n-1}(x_1,\ldots,x_{n-1})\, x_n^{d_n-1} + \cdots = 0 \end{array}\right.$$

or said otherwise, if each polynomial p_i only involves the variables $x_1,\ldots,x_i$, and is monic in the variable x_i. When a zero-dimensional system has such a form, its resolution reduces to a finite number of univariate polynomial equations to solve: it suffices to find the roots x_1 of p_1, to substitute them into p_2, to then find the roots x_2 of the latter, and so on. This strategy works both when looking for exact and approximate (numerical) solutions.

Not every system is equivalent to a triangular system. Consider for example the ideal J defined by:

```
sage: R.<x,y> = PolynomialRing(QQ, order='lex')
sage: C = ideal(x^2+y^2-1)
sage: D = ideal((x+y-1)*(x+y+1))
sage: J = C + D
```

For an image, see Figure 9.4 (left):

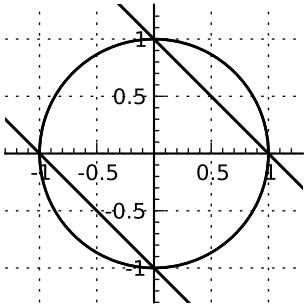
```
sage: opts = {'axes':True, 'gridlines':True, 'frame':False,
....: 'aspect_ratio':1, 'axes_pad':0, 'xmin':-1.3, 'xmax':1.3,
....: 'ymin':-1.3, 'ymax':1.3, 'fontsize': 8}
sage: show(C.plot() + D.plot(), figsize=[2,2], **opts)
```

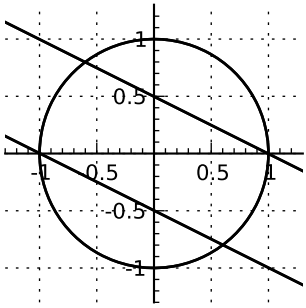
The variety $V(J)$ contains two points of abscissa 0 but only one point of abscissa -1, and likewise, one point of ordinate -1 against two points of zero ordinate. Hence the ideal J cannot be described by a triangular system.

We can however show that any zero-dimensional ideal can be written as a finite intersection of ideals generated by triangular systems. The `triangular_decomposition` method computes such a decomposition:

```
sage: J.triangular_decomposition()
[Ideal (y, x^2 - 1) of Multivariate Polynomial Ring in x, y
over Rational Field,
Ideal (y^2 - 1, x) of Multivariate Polynomial Ring in x, y
over Rational Field]
```

Geometrically, we obtain a representation of the variety $V(J)$ as a union of varieties associated to simpler systems, and often simple enough to give a good description of the solutions.





$\langle x^2+y^2-1, (x+y-1)(x+y+1)\rangle$ $\langle x^2+y^2-1, (x+2y-1)(x+2y+1)\rangle$

FIGURE 9.4 – In each case, the variety associated to the ideal J from the text is the intersection of a circle and the union of two lines.

Some Difficulties. We can rightfully wonder about the interest of the triangular decomposition to enumerate the solutions. After all, given a zero-dimensional system, it is always possible to find a univariate polynomial whose roots are exactly the first coordinates of the solutions, by computing some elimination ideal. By substituting its roots into the system, we decrease the number of variables, which allows to iterate the process, until we have completely solved the system.

However, the "substitution" in the system by propagating the partial results might be intricate. Let us slightly modify the preceding system:

```
sage: D = ideal((x+2*y-1)*(x+2*y+1)); J = C + D
sage: J.variety()
[{y: -4/5, x: 3/5}, {y: 0, x: -1}, {y: 0, x: 1}, {y: 4/5, x: -3/5}]
sage: [T.gens() for T in J.triangular_decomposition()]
[[y, x^2 - 1], [25*y^2 - 16, 4*x + 3*y]]
```

The shape of the triangular decomposition remains the same: for each component, we have an equation involving y only, and a second equation enabling to express x in terms of y.

Thus let us eliminate x, to get an equation in y only, which is the product of the two above equations in y:

```
sage: Jy = J.elimination_ideal(x); Jy.gens()
[25*y^3 - 16*y]
```

To find x, it now suffices to substitute the roots of this equation into the equations defining the ideal J. The first equation, $x^2+y^2-1=0$, yields:

```
sage: ys = QQ['y'](Jy.0).roots(); ys
[(4/5, 1), (0, 1), (-4/5, 1)]
sage: QQ['x'](J.1(y=ys[0][0])).roots()
[(-3/5, 1), (-13/5, 1)]
```

One of these two values is correct — we have $(-3/5, 4/5) \in V(J)$ — but the other one does not correspond to any solution: we have to check the found values using the second initial equation, $(x+2y-1)(x+2y+1)$, to eliminate it.

The problem becomes harder if one solves the univariate equations numerically, which is sometimes necessary due to the cost of operations on algebraic numbers:

```
sage: ys = CDF['y'](Jy.0).roots(); ys
[(-0.8000000000000002, 1), (0.0, 1), (0.8, 1)]
sage: [CDF['x'](p(y=ys[0][0])).roots() for p in J.gens()]
[[(-0.5999999999999999 - 1.306289919090511e-16*I, 1),
 (0.6000000000000001 + 1.3062899190905113e-16*I, 1)],
 [(0.6000000000000001 - 3.1350958058172247e-16*I, 1),
  (2.600000000000001 + 3.135095805817224e-16*I, 1)]]
```

Here, by substituting $y \simeq -0.8$ into the two generators of J, we find two values of x near from 0.6. How to ensure they are approximations of the coordinate x of the same exact solution $(x, y) \simeq (0.6, -0.8)$, and not some spurious roots as in the preceding example? This phenomenon gets trickier when the number of variables and equations grows. However, when the system is triangular, only one equation has to be considered at each substitution step, and since this equation is monic, the numerical approximations do not change the number of solutions.

Let us continue. For the following system, `J.variety()` computes (exactly) a triangular decomposition of J, then finds numerically the real solutions of the obtained system(s). This yields a unique real solution:

```
sage: R.<x,y> = QQ[]; J = ideal([ x^7-(100*x-1)^2, y-x^7+1 ])
sage: J.variety(RealField(51))
[{y: 396340.890166545, x: -14.1660266425312}]
```

Yet, by performing the computation exactly until the end, we see there are three real solutions, and the value of x in the above numerical solution is completely wrong:

```
sage: J.variety(AA)
[{x: 0.00999999900000035?, y: -0.999999999999990?},
{x: 0.01000000100000035?, y: -0.999999999999990?},
{x: 6.3055689986413 85?, y: 396340.8901665450?}]
```

Conclusion: the triangular decomposition does not solve all problems, and we should be careful in the interpretation of approximate computations.

A large number of other methods exist to parametrise and approximate the solutions of zero-dimensional systems, more or less well suited to a given problem, which are not implemented within Sage. Exercise 37 gives an overview of some ideas used.

Advanced mathematics

Sage also provides many functions for commutative algebra and algebraic geometry, which go beyond the scope of this book. We invite the interested reader to explore the documentation of the polynomial ideals, and that of the `sage.schemes` module. Other functionalities are also available through the interfaces of the specialised tools Singular, CoCoA and Macaulay2.

Quotient Algebra. The quotients by zero-dimensional ideals are much easier to manipulate than those by general ideals, since the computations in the quotient algebra reduce to linear algebra in finite dimension.

If $J \subset K[\boldsymbol{x}]$ is a zero-dimensional ideal, the dimension $\dim_K K[\boldsymbol{x}]/J$ of the quotient algebra as a K-vector space bounds the number of points of $V(J)$. (Indeed, for any $u \in V(J)$, there exists a polynomial with coefficients in K which equals 1 at u and 0 at any other point of $V(J)$. Two such polynomials cannot be equivalent modulo J.) We can consider this dimension as the number of solutions "with multiplicity" of the system over the algebraic closure of K. For example, we have noticed that the four solutions of system (S_2) introduced in §9.2.3 are each one the "double" intersection of both curves. This explains the following:

```
sage: len(J2.variety(QQbar)), J2.vector_space_dimension()
(4, 8)
```

The `normal_basis` method computes a list of monomials whose projections on $K[\boldsymbol{x}]/J$ constitute a basis:

```
sage: J2.normal_basis()
[x*y^3, y^3, x*y^2, y^2, x*y, y, x, 1]
```

The returned basis depends on the monomial order chosen at the construction of the polynomial ring; we will describe it more precisely in §9.3.3.

Exercise 37. Let J be a zero-dimensional ideal of $\mathbb{Q}[x, y]$. Let χ_x be the characteristic polynomial of the linear transformation

$$\begin{array}{rccc} m_x : & \mathbb{Q}[x,y]/J & \to & \mathbb{Q}[x,y]/J \\ & p + J & \mapsto & xp + J. \end{array}$$

Compute χ_x in the case $J = J_2 = \langle x^2 + y^2 - 1, 4x^2y^2 - 1\rangle$. Show that every root of χ_x is the abscissa of a point of the variety $V_{\mathbb{C}}(J)$.

9.3 Gröbner Bases

So far, we have used as black boxes the functions provided by Sage for the algebraic elimination and the resolution of polynomial systems. This section introduces some of the underlying mathematical and algorithmic tools. The goal is both to be able to call directly these tools, and to make a wise use of the high-level functions seen before.

The methods used by Sage for computation with ideals and elimination are based on the concept of Gröbner basis. We can consider a Gröbner basis as a multivariate extension of the representation by principal generator of ideals of $K[x]$. The main problem of this section is to define and compute a normal form for the elements of quotient algebras from $K[\boldsymbol{x}]$. Our point of view remains that of the user: we define Gröbner bases, we show how to obtain them with Sage and how they can be useful, but we do not discuss the algorithms used to compute them.

Main monomial orders, with the example of $\mathbb{Q}[x,y,z]$	
`lex`	$\boldsymbol{x}^{\boldsymbol{\alpha}} < \boldsymbol{x}^{\boldsymbol{\beta}} \iff \alpha_1 < \beta_1$ or $(\alpha_1 = \beta_1$ and $\alpha_2 < \beta_2)$ or $\dots$ or $(\alpha_1 = \beta_1, \dots, \alpha_{n-1} = \beta_{n-1}$ and $\alpha_n < \beta_n)$ $x^3 > x^2y > x^2z > x^2 > xy^2 > xyz > xy > xz^2 > xz > x > y^3 > y^2z > y^2 > yz^2 > yz > y > z^3 > z^2 > z > 1$
`invlex`	$\boldsymbol{x}^{\boldsymbol{\alpha}} < \boldsymbol{x}^{\boldsymbol{\beta}} \iff \alpha_n < \beta_n$ or $(\alpha_n = \beta_n$ and $\alpha_{n-1} < \beta_{n-1})$ or $\dots$ or $(\alpha_n = \beta_n, \dots, \alpha_2 = \beta_2$ and $\alpha_1 < \beta_1)$ $z^3 > yz^2 > xz^2 > z^2 > y^2z > xyz > yz > x^2z > xz > z > y^3 > xy^2 > y^2 > x^2y > xy > y > x^3 > x^2 > x > 1$
`deglex`	$\boldsymbol{x}^{\boldsymbol{\alpha}} < \boldsymbol{x}^{\boldsymbol{\beta}} \iff \lvert\boldsymbol{\alpha}\rvert < \lvert\boldsymbol{\beta}\rvert$ or $(\lvert\boldsymbol{\alpha}\rvert = \lvert\boldsymbol{\beta}\rvert$ and $\boldsymbol{x}^{\boldsymbol{\alpha}} <_{\texttt{lex}} \boldsymbol{x}^{\boldsymbol{\beta}})$ $x^3 > x^2y > x^2z > xy^2 > xyz > xz^2 > y^3 > y^2z > yz^2 > z^3 > x^2 > xy > xz > y^2 > yz > z^2 > x > y > z > 1$
`degrevlex`	$\boldsymbol{x}^{\boldsymbol{\alpha}} < \boldsymbol{x}^{\boldsymbol{\beta}} \iff \lvert\boldsymbol{\alpha}\rvert < \lvert\boldsymbol{\beta}\rvert$ or $(\lvert\boldsymbol{\alpha}\rvert = \lvert\boldsymbol{\beta}\rvert$ and $\boldsymbol{x}^{\boldsymbol{\alpha}} >_{\texttt{invlex}} \boldsymbol{x}^{\boldsymbol{\beta}})$ $x^3 > x^2y > xy^2 > y^3 > x^2z > xyz > y^2z > xz^2 > yz^2 > z^3 > x^2 > xy > y^2 > xz > yz > z^2 > x > y > z > 1$

Construction of monomial orders	
object representing a predefined order on n variables	`TermOrder('nom', n)`
matrix order: $\boldsymbol{x}^{\boldsymbol{\alpha}} <_M \boldsymbol{x}^{\boldsymbol{\beta}} \iff M\boldsymbol{\alpha} <_{\texttt{lex}} M\boldsymbol{\beta}$	`TermOrder(M)`
blocks: $\boldsymbol{x}^{\boldsymbol{\alpha}}\boldsymbol{y}^{\boldsymbol{\beta}} < \boldsymbol{x}^{\boldsymbol{\gamma}}\boldsymbol{y}^{\boldsymbol{\delta}} \iff \boldsymbol{\alpha} <_1 \boldsymbol{\gamma}$ or $(\boldsymbol{\alpha} = \boldsymbol{\gamma}, \quad \boldsymbol{\beta} <_2 \boldsymbol{\delta})$	`T1 + T2`

TABLE 9.5 – Monomial orders.

9.3.1 Monomial Orders

A *monomial order* or *admissible order* is a total order on the monomials $\boldsymbol{x}^{\boldsymbol{\alpha}}$ of a polynomial ring, which satisfies

$$\boldsymbol{x}^{\boldsymbol{\alpha}} < \boldsymbol{x}^{\boldsymbol{\beta}} \implies \boldsymbol{x}^{\boldsymbol{\alpha}+\boldsymbol{\gamma}} < \boldsymbol{x}^{\boldsymbol{\beta}+\boldsymbol{\gamma}} \qquad \text{and} \qquad \boldsymbol{\gamma} \neq 0 \implies 1 < \boldsymbol{x}^{\boldsymbol{\gamma}} \tag{9.9}$$

for all exponents $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}$. Equivalently, we can consider $<$ as an order on the exponents $\boldsymbol{\alpha} \in \mathbb{N}^n$ or on the terms $c\,\boldsymbol{x}^{\boldsymbol{\alpha}}$. The leading monomial, leading coefficient and leading term of a polynomial p (see §9.1.3) for the current monomial order are those of largest exponent; we denote them respectively by $\operatorname{lm} p$, $\operatorname{lc} p$ and $\operatorname{lt} p$.

The first condition in (9.9) states that the monomial order should be compatible with products: multiplying by a fixed monomial does not change the order. The second condition implies that $<$ is a well-order, i.e., an infinite sequence of decreasing monomials does not exist. Let us notice that the only monomial order on $K[x]$ is the usual one $x^n > x^{n-1} > \cdots > 1$.

We have seen in §9.1.1 that Sage allows an order to be chosen when defining a polynomial ring via constructions like

```
sage: R.<x,y,z,t> = PolynomialRing(QQ, order='lex')
```

Table 9.5 lists the main predefined monomial orders[6]: `lex` is the lexicographic order of the exponents, `invlex` is the lexicographic order of exponents read

[6]Sage also allows orders (called "local") where 1 is the largest monomial instead of the smallest one. For example, in the order `neglex` on $\mathbb{Q}[x,y,z]$, we have $1 > z > z^2 > z^3 > y >$

from right to left, and `deglex` sorts the monomials first by total degree, then by lexicographic order. The definition of `degrevlex` is slightly more complex: the monomials are sorted by total degree, then by *decreasing* lexicographic order of the exponents *read from the right*. This strange order is nevertheless used by default when we omit the `order` option, since it is more efficient than other orders for some computations.

We generally choose (but not always!) simultaneously the order of variables of the ring and that of monomials such that $x_1 > x_2 > \cdots > x_n$, and we then often speak, for example, of the "`lex` order such that $x > y > z$" instead of the "`lex` order on $K[x, y, z]$". The predefined orders `lex`, `deglex` and `degrevlex` obey that convention; for the `invlex` order on $K[x, y, z]$, it is also the `lex` order such that $z > y > x$, i.e., the `lex` order on $K[z, y, x]$.

9.3.2 Division by a Family of Polynomials

A monomial order $<$ being fixed, let $G = \{g_1, g_2, \ldots, g_s\}$ be a finite set of polynomials from $K[\boldsymbol{x}]$. We denote by $\langle G\rangle = \langle g_1, g_2, \ldots, g_s\rangle$ the ideal of $K[\boldsymbol{x}]$ generated by G.

The division of a polynomial $p \in K[\boldsymbol{x}]$ by G is a multivariate analogue of the Euclidean division in $K[x]$. Like the latter, it associates to p a remainder, given in Sage by the expression `p.reduce(G)`, which is a "smaller" polynomial belonging to the same equivalence class modulo $\langle G\rangle$:

```
sage: ((x+y+z)^2).reduce([x-t, y-t^2, z^2-t])
2*z*t^2 + 2*z*t + t^4 + 2*t^3 + t^2 + t
```

The remainder is obtained by subtracting from p, while possible, multiples of elements of G whose leading term cancels a term of p in the subtraction. Contrary to the univariate case, it might happen that one can thus cancel a term of p, but not the leading one: we then only require to cancel the largest term according to the monomial order.

Formally, for $p \in K[\boldsymbol{x}]$, let us denote by $\mathrm{lt}_G\, p$ the term of p of maximal exponent, which is divisible by a leading term of an element of G. Let us call *elementary reduction* each transformation of the form

$$p \mapsto \tilde{p} = p - c\,\boldsymbol{x}^{\boldsymbol{\alpha}}\, g, \qquad \text{where } g \in G \text{ and } \mathrm{lt}_G\, p = c\,\boldsymbol{x}^{\boldsymbol{\alpha}}\ \mathrm{lt}\, g. \tag{9.10}$$

An elementary reduction leaves unchanged the equivalence class of p modulo $\langle G\rangle$, and makes the largest cancelled monomial of p disappear: we have

$$\tilde{p} - p \in \langle G\rangle \qquad \text{and} \qquad \mathrm{lt}_G\, \tilde{p} < \mathrm{lt}_G\, p.$$

Since $<$ is a well-order, it is not possible to apply to a polynomial an infinite number of successive elementary reductions. Each sequence of elementary reductions ends

$yz > yz^2 > y^2 > y^2z > y^3 > x > xz > xz^2 > xy > xyz > xy^2 > x^2 > x^2z > x^2y > x^3$. The local orders are not well-orders in the sense of definition (9.9), and we do not use them in this book; however the curious reader will complete Table 9.5 by using the `test_poly` function defined in §9.1.1.

thus on a polynomial that cannot be reduced further, and which is the remainder of the division.

Let us notice that this process generalises some familiar elimination methods both for univariate polynomials and linear systems. In one variable, the division of a polynomial p by a singleton $G = \{g\}$ reduces exactly to the Euclidean division of p by g. In the other extreme case of multivariate polynomials, but whose monomials are all of degree 1, it becomes identical to the elementary reduction of the Gauss-Jordan method.

But contrary to what happens in those particular cases, in general, the remainder depends on the choice of the elementary reductions. (We then say that the system (9.10) of rewriting rules is not confluent.) Thus, changing the order in which we give the elements of G leads in the following example to different choices of reduction:

```
sage: R.<x,y> = PolynomialRing(QQ, order='lex')
sage: (g, h) = (x-y, x-y^2);  p = x*y - x
sage: p.reduce([g, h])  # two reductions by h
y^3 - y^2
sage: p.reduce([h, g])  # two reductions by g
y^2 - y
```

Even if the elements of G are considered in a deterministic order (such that the result is unique for given p and G), how to ensure, for example, that the chosen sequence of elementary reductions of p by $\{g, h\}$ will discover the following relation, which shows that $p \in \langle g, h\rangle$?

```
sage: p - y*g + h
0
```

9.3.3 Gröbner Bases

The limitations of multivariate division explain the difficulty mentioned in §9.2.3 to obtain a normal form for the elements of the algebras $K[\boldsymbol{x}]/J$: dividing by the generators of the ideal is not enough... At least in general! Indeed, some particular systems of generators exist for which the division is confluent, and computes a normal form. These systems are called *Gröbner bases*.

Staircases. A pleasant way to grasp Gröbner bases goes through the notion of ideal staircase. Let us attach to each non-zero polynomial from $K[x_1, \ldots, x_n]$ a point of $\mathbb{N}^n$ given by its leading exponent, and let us draw the part $E \subset \mathbb{N}^n$ occupied by an ideal J (see Figures 9.5 to 9.7). The resulting graph (which depends on the monomial order) has a staircase shape: indeed, we have $\boldsymbol{\alpha} + \mathbb{N}^n \subset E$ for any $\boldsymbol{\alpha} \in E$. The elements of $J \setminus \{0\}$ are in the grey zone, above the staircase or at its frontier. The points strictly "under the staircase" correspond exclusively to polynomials from $K[\boldsymbol{x}] \setminus J$, but not all the polynomials from $K[\boldsymbol{x}] \setminus J$ are under the staircase.

For example, in a polynomial of the ideal $\langle x^3, xy^2z, xz^2\rangle$, each monomial, either a leading monomial or not, is multiple of one of the polynomials x^3, xy^2z

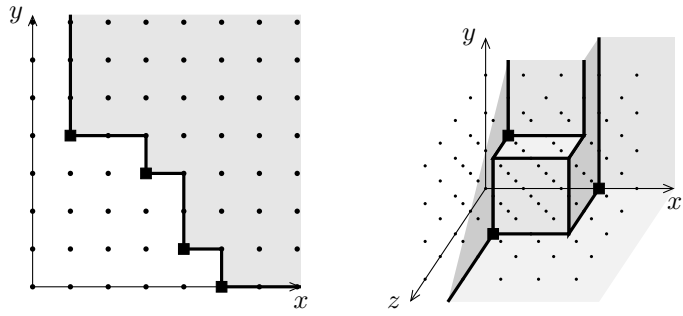


FIGURE 9.5 – Ideal staircases generated by monomials.

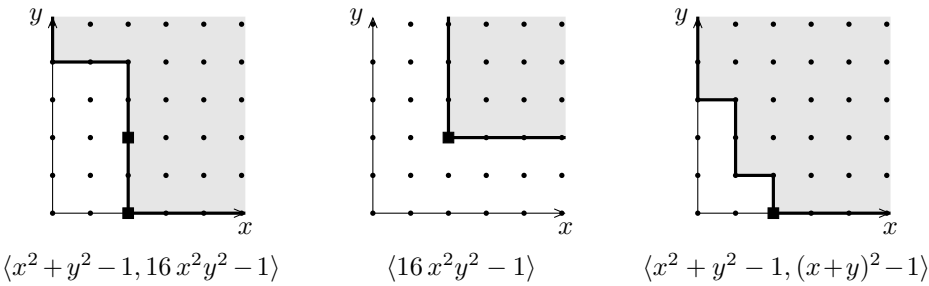


FIGURE 9.6 – Ideal staircases of $\mathbb{Q}[x, y]$ encountered in this chapter. In the three cases, the staircases and the location of generators are the same for the monomial orders `lex`, `deglex` and `degrevlex`.

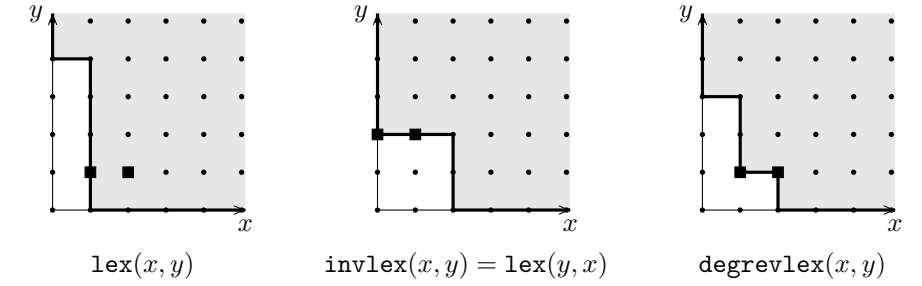


FIGURE 9.7 – Staircases of the ideal $\langle xy + x + y^2 + 1, x^2y + xy^2 + 1\rangle \subset \mathbb{Q}[x, y]$ relative to different monomial orders.

In each diagram, the grey zone corresponds to the leading terms of *elements* of the ideal. The black squares give the location of *generators* used to describe it.

and xz^2. The leading monomials are thus exactly the $x^\alpha y^\beta z^\gamma$ verifying one of the inequalities $(\alpha,\beta,\gamma) \geq (3,0,0)$, $(\alpha,\beta,\gamma) \geq (1,2,1)$ or $(\alpha,\beta,\gamma) \geq (1,0,2)$ component by component (Figure 9.5). A polynomial whose leading exponent does not satisfy these conditions, for example x^2+xz^2 if the monomial order is the lexicographic one with $x > y > z$, cannot belong to that ideal. Some polynomials like $x^3 + x$ are not in the ideal either, despite a leading monomial above the staircases. The situation is analogous for any ideal generated by monomials.

For a random ideal, the staircase structure cannot be easily read on the generators. For instance, by denoting $\boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_s$ the leading exponents of the generators, we have $\bigcup_{i=1}^{s}(\boldsymbol{\delta}_i + \mathbb{N}^n) \subsetneq E$ in all examples of Figures 9.6 and 9.7 except the second one. We can nevertheless show that E can always be written as a finite union of sets of the form $\boldsymbol{\alpha} + \mathbb{N}^n$, i.e., intuitively, that the staircase has a finite number of corners. This result is sometimes called Dickson's lemma.

Gröbner Bases. A Gröbner basis is simply a family of generators that captures the shape of the staircase, more precisely that contains a polynomial corresponding to each corner.

DEFINITION. A Gröbner basis of an ideal $J \subset K[\boldsymbol{x}]$ relative to a monomial order $<$ is a finite part G of J such that for any non-zero $p \in J$, there exists $g \in G$ whose leading monomial $\operatorname{lm} g$ (for the order $<$) divides $\operatorname{lm} p$.

Checking whether the generators defining an ideal form a Gröbner basis is done in Sage with the `basis_is_groebner` method. We have already noticed that any set of monomials is a Gröbner basis:

```
sage: R.<x,y> = PolynomialRing(QQ, order='lex')
sage: R.ideal(x*y^4, x^2*y^3, x^4*y, x^5).basis_is_groebner()
True
```

However, the system $\{x^2 + y^2 - 1, 16\,x^2y^2 - 1\}$ which encodes the intersection of the circle and the hyperbolas of Figure 9.1a is not a Gröbner basis:

```
sage: R.ideal(x^2+y^2-1, 16*x^2*y^2-1).basis_is_groebner()
False
```

According to the staircase shape (Figure 9.6), it lacks a polynomial from J_1 of leading monomial y^4.

The reasoning based on Dickson's lemma, mentioned above, shows that every ideal has Gröbner bases[7]. Let us compute Gröbner bases of J_1 and of the other ideals whose staircases are shown in Figure 9.6. In the case of J_1, it yields:

```
sage: R.ideal(x^2+y^2-1, 16*x^2*y^2-1).groebner_basis()
[x^2 + y^2 - 1, y^4 - y^2 + 1/16]
```

The leading monomials x^2 and y^4 appear as expected. Their presence explains the way the staircase closes itself on the axes; we will see it is characteristic of zero-dimensional systems. For the double hyperbola alone, we find:

[7] We can see this result as an effective version of the Hilbert Basis Theorem, which states that ideals from $K[\boldsymbol{x}]$ are generated by a finite number of elements. A classical proof of this theorem is very similar to the construction of a Gröbner basis for the lexicographic order.

```
sage: R.ideal(16*x^2*y^2-1).groebner_basis()
[x^2*y^2 - 1/16]
```

i.e., a multiple of the generator. In general, each singleton is a Gröbner basis by itself. The third example shows that a Gröbner basis might contain more polynomials than a system of generators:

```
sage: R.ideal(x^2+y^2-1, (x+y)^2-1).groebner_basis()
[x^2 + y^2 - 1, x*y, y^3 - y]
```

Due to the simplicity of the previous examples, these three Gröbner bases do not depend much, if at all, on the monomial order. The general situation is quite different. Figure 9.7 represents the staircases associated to the same ideal from $\mathbb{Q}[x, y]$ for three classical monomial orders. Corresponding Gröbner bases are:

```
sage: R_lex.<x,y> = PolynomialRing(QQ, order='lex')
sage: J_lex = (x*y+x+y^2+1, x^2*y+x*y^2+1)*R_lex; J_lex.gens()
[x*y + x + y^2 + 1, x^2*y + x*y^2 + 1]
sage: J_lex.groebner_basis()
[x - 1/2*y^3 + y^2 + 3/2, y^4 - y^3 - 3*y - 1]
```

```
sage: R_invlex = PolynomialRing(QQ, 'x,y', order='invlex')
sage: J_invlex = J_lex.change_ring(R_invlex); J_invlex.gens()
[y^2 + x*y + x + 1, x*y^2 + x^2*y + 1]
sage: J_invlex.groebner_basis()
[y^2 + x*y + x + 1, x^2 + x - 1]
```

```
sage: R_drl = PolynomialRing(QQ, 'x,y', order='degrevlex')
sage: J_drl = J_lex.change_ring(R_drl); J_drl.gens()
[x*y + y^2 + x + 1, x^2*y + x*y^2 + 1]
sage: J_drl.groebner_basis()
[y^3 - 2*y^2 - 2*x - 3, x^2 + x - 1, x*y + y^2 + x + 1]
```

The Gröbner basis for the `lex` order clearly demonstrates the rewriting rule $x = \frac{1}{2}\,y^3 - y^2 - \frac{3}{2}$, thanks to which we can express the elements of the quotient algebra in terms of the variable y only. The stretched out form of the corresponding staircase translates this rule. Similarly, the Gröbner basis for the `invlex` order indicates that we can eliminate powers of y via the equality $y^2 = -xy - x - 1$. We will come back to this at the end of next section.

9.3.4 Gröbner Basis Properties

Gröbner bases are used to implement the operations studied in Section 9.2. We use them in particular to compute normal forms for ideals in polynomial rings and for elements in quotients by these ideals, to eliminate variables in polynomial systems, or to determine characteristics of the solutions such as their dimension.

Reduction	
multivariate division of p by G	`p.reduce(G)`
inter-reduced generators	`J.interreduced_basis()`
Gröbner bases	
Gröbner basis test	`J.basis_is_groebner()`
(reduced) Gröbner basis	`J.groebner_basis()`
change of order towards `lex`	`J.transformed_basis()`
change of order $R_1 \to R_2$	`J.transformed_basis('fglm', other_ring=R2)`

TABLE 9.6 – Gröbner bases.

Division by a Gröbner Basis. Division by a Gröbner basis G of a polynomial from $\langle G \rangle$ cannot end on a non-zero element of $\langle G \rangle$. This is an immediate consequence of the definition: indeed, such an element would be above the staircase associated to $\langle G \rangle$, thus still divisible by G. Hence every element from $\langle G \rangle$ reduces to zero in the division by G. In particular, a Gröbner basis of an ideal J generates J.

Likewise, the division of a polynomial $p \notin J$ by a Gröbner basis of J can only end on a polynomial "under the staircase", moreover two distinct polynomials "under the staircase" belong to different equivalence classes modulo J (since their difference is still "under the staircase"). The division by a Gröbner basis therefore provides a normal form for the elements of the quotient $K[\boldsymbol{x}]/J$, and this holds independently of the order in which we perform the elementary reductions. The normal form of an equivalence class $p + J$ is its unique representative under the staircase, or zero. This is the normal form computed by the operations in the quotient algebra presented in §9.2.3. To continue the example of Figure 9.7, the reduction

```
sage: p = (x + y)^5
sage: J_lex.reduce(p)
17/2*y^3 - 12*y^2 + 4*y - 49/2
```

decomposes into a Gröbner basis computation, followed by a division:

```
sage: p.reduce(J_lex.groebner_basis())
17/2*y^3 - 12*y^2 + 4*y - 49/2
```

The result of a projection onto the quotient is essentially the same:

```
sage: R_lex.quo(J_lex)(p)
17/2*ybar^3 - 12*ybar^2 + 4*ybar - 49/2
```

Naturally, changing the monomial order yields another normal form:

```
sage: R_drl.quo(J_drl)(p)
5*ybar^2 + 17*xbar + 4*ybar + 1
```

The monomials appearing in the normal form correspond to the points under the staircase.

Hence, the ideal J is zero-dimensional if and only if the number of points under its staircase is finite, and this number of points is the dimension of the quotient $K[\boldsymbol{x}]/J$. In this case, the basis returned by the method `normal_basis` described in §9.2.5 is simply the set of monomials under the staircase for the associated monomial order:

```
sage: J_lex.normal_basis()
[y^3, y^2, y, 1]
sage: J_invlex.normal_basis()
[x*y, y, x, 1]
sage: J_drl.normal_basis()
[y^2, y, x, 1]
```

Let us notice that the number of monomials under the staircase is independent of the monomial order.

Dimension. We are now equipped to give a general definition of the dimension of an ideal: J has dimension d when the number of points under the staircase, corresponding to monomials of total degree at most t, is of order t^d when $t \to \infty$. For example, the ideal $\langle 16\,x^2y^2 - 1\rangle$ (Figure 9.6) has dimension 1:

```
sage: ideal(16*x^2*y^2-1).dimension()
1
```

Indeed, the number of monomials m under the staircase such that $\deg_x m + \deg_y m \leq t$ equals $4t - 2$ for $t \geq 3$. Likewise, the two ideals of Figure 9.5 have respectively dimension 1 and 2. We can show that the dimension does not depend on the monomial order, and corresponds — degeneracies excepted — to the "geometric" dimension of the associated variety.

Reduced Bases. A finite set of polynomials containing a Gröbner basis is itself a Gröbner basis, therefore a non-zero ideal has an infinite number of Gröbner bases. A Gröbner basis $G = \{g_1, \ldots, g_s\}$ is called *reduced* when

- the leading coefficients of g_i are all 1 (and $0 \notin G$);
- and no term of g_i is reducible by the rest of the basis $G \setminus \{g_i\}$ in the sense of the rules (9.10).

With a fixed monomial order, each ideal has a unique reduced Gröbner basis. For example, the reduced Gröbner basis of the ideal $\langle 1\rangle$ is the singleton $\{1\}$, whatever the monomial order. The reduced Gröbner bases therefore provide a normal form for all ideals of $K[\boldsymbol{x}]$.

A reduced Gröbner basis is minimal in the sense that, if we remove any element, what remains is no longer a system of generators of the ideal. Concretely, it contains exactly one polynomial per "corner" of the staircase. It can be computed from any Gröbner basis G by replacing each element $g \in G$ by its remainder for the division by $G \setminus \{g\}$, and so on while possible. This is what the `interreduced_basis` method does. The polynomials reducing to zero are erased.

Elimination. The lexicographic monomial orders have the following fundamental property: *if G is a Gröbner basis for the lexicographic order of $J \subset K[x_1, \ldots, x_n]$, then the $G \cap K[x_{k+1}, \ldots, x_n]$ are Gröbner bases for the elimination ideals*[8] $J \cap K[x_{k+1}, \ldots, x_n]$. A lexicographic Gröbner basis splits into blocks, the last one of which depends only on x_n, the penultimate on x_n and x_{n-1}, and so on[9]:

```
sage: R.<t,x,y,z> = PolynomialRing(QQ, order='lex')
sage: J = R.ideal(t+x+y+z-1, t^2-x^2-y^2-z^2-1, t-x*y)
sage: [u.polynomial(u.variable(0)) for u in J.groebner_basis()]
[t + x + y + z - 1,
(y + 1)*x + y + z - 1,
(z - 2)*x + y*z - 2*y - 2*z + 1,
(z - 2)*y^2 + (-2*z + 1)*y - z^2 + z - 1]
```

In this example, the last polynomial of the basis only depends on y and z. It is preceded by a block of two polynomials in x, y and z, and the first polynomial contains all variables. The successive elimination ideals can be seen immediately.

We have seen however (§9.2.5) that the elimination ideals do not provide a perfect description of the ideal. Here, the block of polynomials in z only is empty, thus any value of z, except maybe a finite number, appears as last coordinate of a solution. We are tempted to express the possible values of y for each z thanks to the last equation. We get two values, except for $z = 2$, for which only $y = -1$ works. Only when we get to the preceding equation do we notice that the choice $z = 2$ is contradictory. Inversely, again from the last equation, $y = -1$ implies $z = 2$, and is thus excluded. It finally occurs that none of the leading terms of the polynomials (written in their respective main variable, as in the above Sage output) vanishes for $z \neq 2$.

Exercise 38 (Trigonometric relations)**.** Write $(\sin\theta)^6$ as a polynomial in $u(\theta) = \sin\theta + \cos\theta$ and $v(\theta) = \sin(2\theta) + \cos(2\theta)$.

9.3.5 Computations

We refer the reader interested in algorithms computing Gröbner bases to the reference [CLO07] mentioned in introduction. In addition, the module `sage.rings.polynomial.toy_buchberger` from Sage offers a "pedagogical" implementation of Buchberger's algorithm and various related algorithms, which closely follows their theoretical description.

Let us however keep in mind that computing a Gröbner basis is expensive both in terms of time and memory, and even very expensive in some unlucky cases. Besides, the `groebner_basis` method has several options[10] which allow

[8] For a given k, this is true more generally for any order such that $i \leq k < j \implies x_i > x_j$. Such an order is called a "block order" (see also Table 9.5).

[9] We thus get a "triangular form" of the system formed by the ideal generators, however in a weaker sense with respect to §9.2.5: we cannot say much *a priori* about the number of polynomials in each block or their leading terms.

[10] For more details, see the help page of this method, as well as those of internal methods of the ideal, whose name starts with `_groebner_basis`.

Change of order

The most interesting Gröbner bases are not the easiest to compute: often, the `degrevlex` order is the cheapest, but more useful information can be read on a lexicographic Gröbner basis. Besides, we sometimes need Gröbner bases of the same ideal for different monomial orders.

This motivates the introduction, in addition to general algorithms computing Gröbner bases, of algorithms of "change of order". These algorithms compute a Gröbner basis for a given monomial order from a Gröbner basis of the same ideal for a different order. They are often more efficient than algorithms computing directly a basis for the target order. Thus, a strategy which often wins to compute a lexicographic Gröbner basis is the following: first compute a basis for the `degrevlex` order, then apply an algorithm of change of order. Sage does it automatically in some cases.

The `transformed_basis` method allows us to compute "by hand" Gröbner bases by change of order, when the ideal is zero-dimensional, or when the target order is `lex`. If needed, it first computes a Gröbner basis for the monomial order attached to the polynomial ring.

the expert user to manually choose a Gröbner basis algorithm according to the characteristics of the problem to solve.

Let us consider the ideals $C_n(K) \subset K[x_0, \ldots, x_{n-1}]$ defined by:

$$\begin{aligned} C_2(K) &= \langle x_0 + x_1, x_0x_1 - 1\rangle \\ C_3(K) &= \langle x_0 + x_1 + x_2, x_0x_1 + x_0x_2 + x_1x_2, x_0x_1x_2 - 1\rangle \\ &\vdots \\ C_n(K) &= \Big\langle \sum_{i\in\mathbb{Z}/n\mathbb{Z}} \prod_{j=0}^{k} x_{i+j} \Big\rangle_{k=0}^{n-2} + \langle x_0 \cdots x_{n-1} - 1\rangle, \end{aligned} \tag{9.11}$$

and accessible in Sage by commands like:

```
sage: from sage.rings.ideal import Cyclic
sage: Cyclic(QQ['x,y,z'])
Ideal (x + y + z, x*y + x*z + y*z, x*y*z - 1) of
Multivariate Polynomial Ring in x, y, z over Rational Field
```

They are classical test problems to evaluate the efficiency of tools for solving polynomial systems. On a computer where Sage computes the reduced Gröbner basis of $C_6(\mathbb{Q})$ in less than a second:

```
sage: def C(R, n): return Cyclic(PolynomialRing(R, 'x', n))
```

```
sage: %time len(C(QQ, 6).groebner_basis())
CPU times: user 136 ms, sys: 0 ns, total: 136 ms
Wall time: 147 ms
45
```

the computation of that of $C_7(\mathbb{Q})$ does not terminate after a dozen of hours, and uses more than 3 Gb of memory.

Failing to compute the Gröbner basis over the rational numbers, let us try to replace $\mathbb{Q}$ by a finite field $\mathbb{F}_p$. The idea, classical in computer algebra, is to limit the cost of operations on coefficients: those on elements of $\mathbb{F}_p$ take a constant cost, whereas the number of digits of rational numbers tends to increase quite rapidly during computations. We choose p small enough such that computations in $\mathbb{F}_p$ can be done directly with machine integers. It must not however be too small, so that the Gröbner basis on $\mathbb{F}_p$ can share a large part of the structure of that on $\mathbb{Q}$.

For example, with a convenient p, the Gröbner basis of $C_6(\mathbb{F}_p)$ has the same number of elements as that of $C_6(\mathbb{Q})$:

```
sage: p = previous_prime(2^30)
sage: len(C(GF(p), 6).groebner_basis())
45
```

By increasing the size of the system to solve, we see that the influence of the coefficient field on the computing time is far from negligible: the cases $n = 7$ and $n = 8$ become easy to solve.

```
sage: %time len(C(GF(p), 7).groebner_basis())
CPU times: user 1.44 s, sys: 4 ms, total: 1.44 s
Wall time: 1.46 s
209
sage: %time len(C(GF(p), 8).groebner_basis())
CPU times: user 40.7 s, sys: 24 ms, total: 40.7 s
Wall time: 40.9 s
372
```

These examples illustrate also another important phenomenon: the output of a Gröbner basis computation might be much larger than the input. For example, the last computation above shows that any Gröbner basis, reduced or not, of $C_8(\mathbb{F}_p)$ (with this value of p) for the `degrevlex` order counts at least 372 elements, whereas C_8 is generated by only 8 polynomials.