LCM: Lectures on Computational Mathematics

Przemysław Koprowski

Septempber 10, 2015 – May 21, 2022

These notes were compiled on
May 21, 2022.
Since I update the notes quite frequently,
it is possible that a newer version is already available. Please check on:
http://www.pkoprowski.eu/lcm

Contents

Contents **3**

1 Fundamental algorithms **9**
1.1 Integers and univariate polynomials 9
1.2 Floating point numbers 33
1.3 Euclidean algorithm 38
1.4 Rational numbers and rational functions 57
1.5 Multivariate polynomials 58
1.6 Transcendental constants 58
1.A Scaled Bernstein polynomials 58

2 Modular arithmetic **79**
2.1 Basic modular arithmetic 79
2.2 Chinese remainder theorem 82
2.3 Modular exponentiation 87
2.4 Modular square roots 92
2.5 Applications of modular arithmetic 109

3 Matrices **115**
3.1 Matrix arithmetic 115
3.2 Gauss elimination 118
3.3 Matrix decomposition 126
3.4 Determinant 126
3.5 Systems of linear equations 147
3.6 Eigenvalues and eigenvectors 147
3.7 Short vectors 155
3.8 Special matrices 168

4 Reconstruction and interpolation **175**
4.1 Polynomial interpolation . . . 175
4.2 Runge's phenomenon and Chebyshev nodes . . . 187
4.3 Piecewise polynomial interpolation . . . 193
4.4 Fourier transform and fast multiplication . . . 194

5 Factorization and irreducibility tests **205**
5.1 Primality tests . . . 206
5.2 Integer factorization . . . 215
5.3 Square-free decomposition of polynomials . . . 216
5.4 Applications of square-free decomposition . . . 227
5.5 Polynomial factorization over finite fields . . . 232
5.6 Polynomial factorization over rationals . . . 237

6 Univariate polynomial equations **239**
6.1 Generalities . . . 239
6.2 Explicit formulæ for low degree polynomials . . . 242
6.3 Bounds on roots . . . 246
6.4 Distances between roots . . . 264
6.5 Root counting and isolating . . . 268
6.6 Root approximation . . . 286
6.7 Higher degree solvable polynomials . . . 305
6.8 Strategies for solving univariate polynomial equations . . . 305
6.9 Thom's lemma . . . 305

7 Integration **309**
7.1 Symbolic integration of rational functions . . . 309
7.2 Classical numeric integration schemes . . . 312
7.3 Monte Carlo integration . . . 312

8 Classical elimination theory **313**
8.1 Labatie–Kalkbrener triangulation . . . 315
8.2 Resultant . . . 327
8.3 Applications of resultants . . . 358
8.4 Subresultants . . . 375

9 Gröbner bases **381**
9.1 Algebraic prerequisites . . . 381
9.2 Monomial orders . . . 383
9.3 Gauss elimination revisited . . . 385
9.4 Multivariate polynomial reduction . . . 386
9.5 Gröbner bases . . . 389
9.6 Buchberger's criterion . . . 390
9.7 Minimal and reduced Gröbner bases . . . 392

9.8 Elimination of variables 394

10 Zero-dimensional polynomial systems **399**

A Fundamental concepts from algebra **401**

Bibliography **403**

Introduction

> Witchcraft to the ignorant,...
> simple science to the learned.
>
> "The Sorcerer of Rhiannon",
> L. Brackett

Computational mathematics is the least common multiple of mathematics and algorithms. This is a 'work-in-progress' version of lecture notes to "Introduction to computational mathematics" and "Computational mathematics" courses.

Terry Pratchett in his novel "Interesting times" says that "Magic isn't like maths". Assuming symmetry of the relation, this implies that mathematics is not like magic, either. Even if they both begin with 'ma'. They both provide their adepts with some tools to perform certain actions: spells in the case of magic and theorem in mathematics. The difference is that in mathematics one knows why and how a given tool works. One may read a proof of the theorem and verify its correctness. A mathematician can check how assumptions of the theorem are used and what would happen if they were omitted. On the contrary, in magic (as far as I understand) a wizard casts a spell to obtain the desired result but he does not know why and how the spell works, which ingredients of the spell are important and what would happen if he said 'dabraabraca' instead of 'abracadabra'.

Nowadays, there are many mathematically oriented computer programs (co-called *Computer Algebra Systems* or CASs for short). Some of them are paid. Some are free. Unfortunately, many users treat them like oracles. Without understanding the principles of how these systems works, using them is more magic than mathematics. Issuing a command like

```
(x^6 - 5*x^4 + 4*x^2 - x).roots()
```

is not very different from casting a spell, when one doesn't understand what a system can or cannot do with an unsolvable polynomial (and what it really means that a

polynomial is unsolvable). Thus, the aim of these notes is to explain mathematics that operates behind the scene in computer algebra systems. To show that there is really no magic in them.

1

Fundamental algorithms

A journey of a thousand miles
begins with a single step.

"Tao Te Ching", Laozi

This chapter is devoted to introducing representations of the most elementary mathematical objects, together with accompanying fundamental algorithm, like standard arithmetic operations. These fundamentals seem to be often neglected, but they are indispensable for developing "more serious" algorithms, as they form the basic building blocks of the latter ones. Moreover, these basic arithmetic operations, although shyly hidden inside the big algorithms, are executed over and over again. Thus, their speed have a major impact on the overall performance of their bigger brothers.

For foundations of mathematics the most fundamental objects are sets. In the realm of computational mathematics the elementary objects are integers (for symbolic computations) and floating-point numbers (for numerical methods). Hence, these two types of objects will be are main subject in the next sections. Moreover, since univariate polynomials are very like integers (and algorithms designed for polynomials are usually a little bit simpler) we discuss integers and polynomials together.

1.1 Integers and univariate polynomials

Integers and polynomials exhibit lots of similarities from algebraic and algorithmic points of view. On the algebraic side, the ring of integers and the ring of univariate polynomials (over a field) are both Euclidean domains. From the algorithmic point of view, integers and polynomials are just finite sequences of digits and coefficients, respectively. An integer $n \in \mathbb{Z}$ is represented by a signed, finite sequence of digits $\pm(n_k n_{k-1} \dots n_0)_\beta$ such that

$$n = \pm\big(n_k \beta^k + \cdots + n_1 \beta + n_0\big),$$

where the integer $\beta \geq 2$ is a fixed *base* of the representation and every *digit* n_i is an element of the set $\{0, \ldots, \beta - 1\}$. For example, let n be a number *one hundred ninety-seven thousand six hundred forty-one*, then at base *ten*, the number n is represented by a 6-tuple $(197641)_{10}$, since

$$n = 1 \cdot 10^5 + 9 \cdot 10^4 + 7 \cdot 10^3 + 6 \cdot 10^2 + 4 \cdot 10^1 + 1.$$

On the other hand, the same number is represented as $(110000010000001001)_2$ at base 2 and as $(349)_{256}$ at base 256. The choice of the base is quite arbitrary. As human beings we are used to the base 10, but the only reason for it is that we have a total of ten fingers on both hands. For computers a more natural choice is a power of 2. The base 2 is used mostly at the lowest, hardware level (in this case there are also some other representations in use, like for instance two's-complement notation, which we will not discuss in these notes). While at higher levels of abstraction, say in computer algebra systems, the base β is commonly chosen in such a way that a single digit fits into a single memory slot. So for example, at base 256 the digits are stored as bytes, at base 65 536 as 16-bit words... and so on. Here we will not place any restrictions on the base nor on the *length* of an integer, understood as the number of digits. We will always assume that we are able to store as many digits as we need.

Like integers, univariate polynomials are represented as finite sequences. Let R be a (fixed) ring of coefficients. Assume that we have means to represent elements of R and to operate on them. The simplest and most common approach is to represent a polynomial $f \in R[x]$ as a sequence of its coefficients $(f_n, \ldots, f_1, f_0)$, where $f = f_n x^n + \cdots + f_1 x + f_0$. This is called the *power basis representation* of f, since $f_0, \ldots, f_n$ are nothing else but the coordinates of f with respect to the basis $\{1, x, \ldots, x^k, \ldots\}$ of the free R-module $R[x]$. This is not the only possible representation of a polynomial and later we will discuss scaled Bernstein-basis representation (see Appendix 1.A).

Let $f = f_n x^n + \cdots + f_1 x + f_0$ be a nonzero polynomial. The maximal index k such that $f_k \neq 0$ is called the *degree* of f and denoted $\deg f$. The degree of the zero polynomial is defined to be $-\infty$. It will be often convenient to allow the coefficients of f to be defined for **all** integer indices, with the convention that $f_j \equiv 0$ whenever $j < 0$ or $j > \deg f$. Formally speaking, we view polynomials as infinite sequences indexed by elements of $\mathbb{Z}$ with only finitely many nonzero entries. The set $\operatorname{supp} f := \{k \in \mathbb{Z} \,|\, f_k \neq 0\}$ of the indices of nonzero terms is called the *support* of f. In this book we will sometimes (see e.g. Section 6.3 of Chapter 6) encounter the notion of a reciprocal polynomial, that we now introduce.

Definition 1.1.1. Let $f = f_n x^n + \cdots + f_1 x + f_0$ be a polynomial. A polynomial $f_0 x^n + \cdots + f_{n-1} x^{n-1} + f_n$, obtained from f by reversing the order of coefficients, is called *reciprocal* polynomial polynomial of f and denoted $\operatorname{rev} f$.

Observation 1.1.2. *Let f be a polynomial, then its reciprocal polynomial equals*

$$\operatorname{rev}(f) = x^n \cdot f\Big(\frac{1}{x}\Big).$$

In what follows we assume that basic arithmetic operations on single-digit integers and coefficients of a polynomial are already available in our computing environment. How these function are internally implemented by the environment/hardware is not important for us. In case of integers, they can be as simple as hard-coded tables (remember how we all had to memorize multiplication table at primary school) or hardware circuits. Using this low-level operations we will define arithmetic operations on integers and polynomials of arbitrary sizes.

Addition and subtraction

The first arithmetic operation we discuss is addition. We begin with addition of polynomials since—as mentioned in the introduction—they are slightly easier to deal with. There are no *carry* to worry about. Since we represent polynomials as finite sequences of coordinates with respect to the power basis, the sum and difference are computed coefficient-wisely. If the two polynomials differ in degree we simply treat the "missing" coefficients as zeros. Hence, if $f = f_m x^m + \cdots + f_1 x + f_0$ and $g = g_n x^n + \cdots + g_1 x + g_0$, then their sum $h = f + g$ has coefficients $h = h_k x^k + \cdots + h_1 x + h_0$, where $h_i = f_i + g_i$, with the convention that the exceeding coefficients are null, i.e.: $f_i = 0$ for $i > m$ and $g_i = 0$ for $i > n$. More precisely

$$h_i = \begin{cases} f_i + g_i, & \text{if } i \leq \min\{m, n\}, \\ f_i, & \text{if } n < i \leq m, \\ g_i, & \text{if } m < i \leq n. \end{cases}$$

The degree $\deg h$ of the sum satisfies $h \leq \max\{m, n\}$ and the equality holds if $m \neq n$.

Algorithm 1.1.3. *Given two univariate polynomials $f = f_m x^m + \cdots + f_1 x + f_0$ and $g = g_n x^n + \cdots + g_1 x + g_0$, this algorithm computes their sum (respectively difference) h.*

1. *Let l be the minimum and k the maximum of the degrees of f and g.*
2. *For every $i \in \{0, \ldots, l\}$, set $h_i := f_i + g_i$ (respectively $h := f_i - g_i$).*
3. *If $\deg f > \deg g$, then set $h_i := f_i$ for $i \in \{l + 1, \ldots, m\}$.*
4. *If $\deg f < \deg g$, then set $h_i := g_i$ (respectively $h := -g_i$) for $i \in \{l + 1, \ldots, n\}$.*
5. *Return $h = h_k x^k + \cdots + h_1 x + h_0$.*

The above algorithm is so trivial and well known that we take the liberty to omit a rigorous proof of its correctness. Addition and subtraction of integers is very similar to their polynomial counterparts. There are only two differences. First, when dealing with integers we must take care of signs of the numbers. Secondly, if a sum of two digits exceeds $\beta - 1$, then a *carry* appears.

Algorithm 1.1.4. *Let $a, b \in \mathbb{N}$ bet two non-negative integers represented as sequences of digits (with respect to an arbitrary base β):*

$$a = (a_m a_{m-1} \ldots a_0)_\beta = a_m \beta^m + \cdots + a_1 \beta + a_0$$
$$b = (b_n b_{n-1} \ldots b_0)_\beta = b_n \beta^n + \cdots + b_1 \beta + b_0.$$

This algorithm computes their sum $c = a + b$.

1. *Set* $k := \max\{m, n\}$.
2. *Initialize a carry* $h := 0$.
3. *For every* $i \in \{0, \ldots, k\}$ *do:*
   a) *Compute the sum* $a_1 + b_i + h = d_0 + d_1\beta$, *where* $d_0, d_1 \in \{0, \ldots, \beta - 1\}$, *with the convention that the exceeding digits are null, by definition* (*i.e.* $a_i \equiv 0$ *for* $i > m$ *and* $b_i \equiv 0$ *for* $i > n$).
   b) *Set the next digit* $c_i := d_0$ *and update the carry setting* $h := d_1$.
4. *If the carry* h *is nonzero, set* $c_{k+1} := h$ *and output* $c = c_{k+1}\beta^{k+1} + c_k\beta^k + \cdots + c_1\beta + c_0$;
5. *Otherwise return* $c = c_k\beta^k + \cdots + c_1\beta + c_0$.

Let us only remark that the sum of three digits in step 3a is well defined since the left-hand-side $a_i + b_i + h$ is at most $(\beta - 1) + (\beta - 1) + 1 \leq \beta^2$ for every input data, as the carry h never exceeds 1. The subtraction of natural numbers $a - b$ for $a \geq b$ is analogous to the addition.

Algorithm 1.1.5. *Let* $a = a_m\beta^m + \cdots + a_1\beta + a_0$ *and* $b = b_n\beta^n + \cdots + b_1\beta + b_0$ *be two non-negative integers represented as sequences of digits* (*with respect to an arbitrary base* β). *Assume that* $a \geq b$. *This algorithm computes their difference* $c = a - b$.

1. *Compute the difference* $a_0 - b_0 = sd$, *where* s *is the sign equal either* $+$ *or* $-$ *and* $d \in \{0, \ldots, \beta - 1\}$. *If* $s = +$, *then set the first digit* $c_0 := d$ *and initialize a carry* $h := 0$, *otherwise take* $c_0 := \beta - d$ *and* $h := 1$.
2. *For all* $0 < i \leq m$ *do:*
   a) *Compute the difference* $a_i - b_i - h = sd$, *with a convention that* $\beta_i \equiv 0$ *for* $i > n$.
   b) *If* $s = +$, *then set the next digit* $c_i := d$ *and update the carry setting* $h := 0$, *otherwise take* $c_i := \beta - d$ *and* $h := 1$.
3. *Prune the trailing zeros setting* $k := \min\{i \geq 1 \,|\, c_j = 0 \text{ for all } j \geq i\}$.
4. *Output the result* $c = c_k\beta^k + \cdots + c_1\beta + c_0$.

Addition of (signed) integers can be implemented using the two above algorithms and the formula:

$$a + b = \begin{cases} \operatorname{sgn}(a) \cdot (|a| + |b|), & \text{if } \operatorname{sgn}(a) = \operatorname{sgn}(b), \\ \operatorname{sgn}(a) \cdot (|a| - |b|), & \text{if } \operatorname{sgn}(a) \neq \operatorname{sgn}(b) \text{ and } |a| \geq |b|, \\ \operatorname{sgn}(b) \cdot (|b| - |a|), & \text{if } \operatorname{sgn}(a) \neq \operatorname{sgn}(b) \text{ and } |a| < |b|. \end{cases}$$

Finally, the subtraction of arbitrary integers is expressed in terms of the addition:

$$a - b = a + (-b),$$

where the negation $b \mapsto (-b)$, boils down to the change of the sign of the sequence of digits: $\pm(b_n \ldots b_0)_\beta \mapsto \mp(b_n \ldots b_0)_\beta$. The only thing left is to decide, which of the two

natural numbers is bigger. This is easily achieved by the following ordering relation on the set of finite sequences. Let $(a_m, \dots, a_0)$ and $(b_n, \dots, b_0)$ be two finite sequences with $a_i, b_i \in \{0, \dots, \beta - 1\}$ and $a_m \neq 0 \neq b_n$. Define a relation $\prec$ as follows

$$(a_m, \dots, a_0) \prec (b_n, \dots, b_0) \iff \begin{cases} m < n \\ \text{or} \\ m = n \text{ and } a_m < b_m. \end{cases}$$

Observation 1.1.6. *Let $a, b \in \mathbb{N}$ bet two non-negative integers represented as sequences of digits (with respect to an arbitrary base β):*

$$a = (a_m a_{m-1} \dots a_0)_\beta = a_m \beta^m + \dots + a_1 \beta + a_0$$
$$b = (b_n b_{n-1} \dots b_0)_\beta = b_n \beta^n + \dots + b_1 \beta + b_0.$$

Assume that a_m and b_n are non zero, then

$$a < b \iff (a_m, \dots, a_0) \prec (b_n, \dots, b_0).$$

Multiplication

The well known formula for polynomial multiplication is the following one. Let $f = f_m x^m + \dots + f_1 x + f_0$ and $g = g_n x^n + \dots + g_1 x + g_0$ be two polynomials, then their product $h = f \cdot g$ has the form $h_k x^k + \dots + h_1 x + h_0$, where $k = m + n$ and

$$h_i := \sum_{j=0}^{i} f_j g_{i-j} = \sum_{j=\max\{0, i-n\}}^{\min\{i,m\}} f_j g_{i-j}, \qquad \text{for} \quad i \in \{0, \dots, k\}. \tag{1.1}$$

This method of computing a product is known as the *long multiplication*. For the sake of completeness let us write it down in form of a pseudo-code.

Algorithm 1.1.7. *Given two polynomials $f = f_m x^m + \dots + f_1 x + f_0$ and $g = g_n x^n + \dots + g_1 x + g_0$, this algorithm computes their product $h = f \cdot g$.*

1. *If any of the two input polynomials is zero, then output $h := 0$ and terminate.*
2. *Set the degree of the output $k := \deg f + \deg g$.*
3. *For every $i \in \{0, \dots, k\}$ set*
$$h_i := \sum_{j=\max\{0, i-n\}}^{\min\{i,m\}} f_j g_{i-j}.$$
4. *Return $h = h_k x^k + \dots + h_1 x + h_0$.*

Time complexity analysis. Let f and g be as above. When computing their product using the above method, we need $m \cdot n$ products and up to $1/4 \cdot \big((m+n)^2 + 2(m+n)\big)$ additions of the coefficients. Provided that the multiplication of coefficients is the dominating operation, the time complexity of long multiplication of polynomials is

$$O\big(D^2 \cdot \text{complexity of arithmetic operations on coefficients}\big),$$

where $D := \max\{\deg f, \deg g\}$.

Multiplication of integers is basically the same as multiplication of polynomials, except that again we must handle the carry and sign.

Algorithm 1.1.8. *Let $a = \pm(a_m\beta^m + \cdots + a_1\beta + a_0)$ and $b = \pm(b_n\beta^n + \cdots + b_1\beta + b_0)$ be two integers. This algorithm computes their product $c := a \cdot b$.*

1. *If any of the two integers is zero, then output $c := 0$ and terminate.*
2. *Set the sign s of the result to be $+$ if the signs of a and b coincide, and to $-$ otherwise.*
3. *Set $k := m + n$ and initialize the carry $h := 0$.*
4. *For every $i \in \{0, \ldots, k\}$ do:*
   a) *Compute $d := h + \sum_{j=\max\{0,i-n\}}^{\min\{i,m\}} a_j b_{i-j}$.*
   b) *Write d as $d = d_0 + d_1\beta$, where $d_0, d_1 \in \{0, \ldots, \beta - 1\}$.*
   c) *Set the next digit $c_i := d_0$ and update the carry setting $h := d_1$.*
5. *If $h \neq 0$ then set $c_{k+1} := d_1$ and output $c = s(c_{k+1}\beta^{k+1} + c_k\beta^k + \cdots + c_1\beta + c_0)$.*
6. *Otherwise output $c = s(c_k\beta^k + \cdots + c_1\beta + c_0)$.*

It is clear that—like for polynomials—the time complexity of long multiplication of integers is quadratic in number of their digits. A faster multiplication algorithm was developed in early 1960s by a Russian mathematician A.A. Karatsuba[1]. Again, let f, g be two univariate polynomials. Take $d := 1/2 \cdot \lceil \max\{\deg f, \deg g\} \rceil$. We can split both polynomials into their "lower" and "upper" parts, as follows

$$f = f_u \cdot x^d + f_l \qquad \text{and} \qquad g = g_u \cdot x^d + g_l.$$

If we compute the product of f and g in a straight-forward way, we have

$$f \cdot g = (f_u x^d + f_l) \cdot (g_u x^d + g_l) = f_u g_u x^{2d} + (f_u g_l + f_l g_u) x^d + f_l g_l.$$

This formula computes four polynomial products: $f_u g_u$, $f_u g_l$, $f_l g_u$ and $f_l g_l$. However, with a cost of some extra additions and subtractions, we may get rid of one of these multiplications. To this end write

$$f \cdot g = f_u g_u x^{2d} + \big((f_u + f_l)(g_u + g_l) - f_u g_u - f_l g_l\big) x^d + f_l g_l.$$

Here we have only three products to compute: $f_u g_u$, $f_l g_l$ and $(f_u + f_l)(g_u + g_l)$. This observation leads us directly to the following *divide et impera* algorithm.

[1] Anatolii Alexeyevich Karatsuba (born on January 31, 1937, died on September 28, 2008) was a Russian mathematician.

Algorithm 1.1.9 (Karatsuba, 1962). *Let $f = f_m x^m + \cdots + f_1 x + f_0$ and $g = g_n x^n + \cdots + g_1 x + g_0$ be two univariate polynomials and let $d := {}^1\!/_2 \cdot \lceil \max\{\deg f, \deg g\} \rceil$. This algorithm computes the product $h = f \cdot g$.*

1. *If any of the two polynomials is zero, then output* 0 *and terminate.*
2. *If any of the two polynomials is constant, then multiply all the coefficients of the other polynomial by this constant, output the result and terminate.*
3. *Split both polynomials into their "upper" and "lower parts" taking*

$$f_u := f_m x^{m-d} + \cdots + f_{d+1} x + f_d, \qquad f_l := f_{d-1} x^{d-1} + \cdots + f_1 x + f_0,$$
$$g_u := g_n x^{n-d} + \cdots + g_{d+1} x + g_d, \qquad g_l := g_{d-1} x^{d-1} + \cdots + g_1 x + g_0.$$

4. *Recursively compute the products:*

$$h_u := f_u g_u, \qquad h_c := (f_u + f_l)(g_u + g_l), \qquad h_l := f_l g_l.$$

5. *Combine the partial results setting*

$$h = h_u \cdot x^{2d} + (h_c - h_u - h_l) \cdot x^d + h_l.$$

6. *Return h.*

Time complexity analysis. Every step of Karatsuba algorithm reduces the degrees of the polynomials involved by half. It follows that the total number of multiplications of coefficients in this method is $3^{\lceil \lg D \rceil} \leq 3 \cdot D^{\lg 3} \simeq 3 \cdot D^{1.585}$, where D is the maximum of the degrees of f and g. Therefore the time complexity of integer multiplication with Karatsuba algorithm is

$$O(D^{\log_2 3} \cdot \text{complexity of arithmetic operations on coefficients}),$$

provided that the dominating operation is the multiplication of the coefficients.

Example 1. Take two polynomials

$$f = x^3 + 4x^2 + 6x + 3, \qquad g = x^3 + 3x^2 + 2x + 1.$$

The diagram in Figure 1.1 illustrates consecutive steps of Karatsuba multiplication of these two polynomials.

To be completely honest, we must admit that Karatsuba originally invented his algorithm to multiply integers rather than polynomials. We leave it as an easy exercise for the reader to adapt Algorithm 1.1.9 to integers. As it is evident from the complexity analysis, Karatsuba multiplication is asymptotically much faster than the long multiplication. Yet it is still not the fastest known multiplication algorithm. In Section 4.4 we discus an even faster algorithm.

Exercise 1. Multiplication of a polynomial by a monomial $f_k x^k$ is extremely simple. Optimize Algorithm 1.1.9 taking this into account.

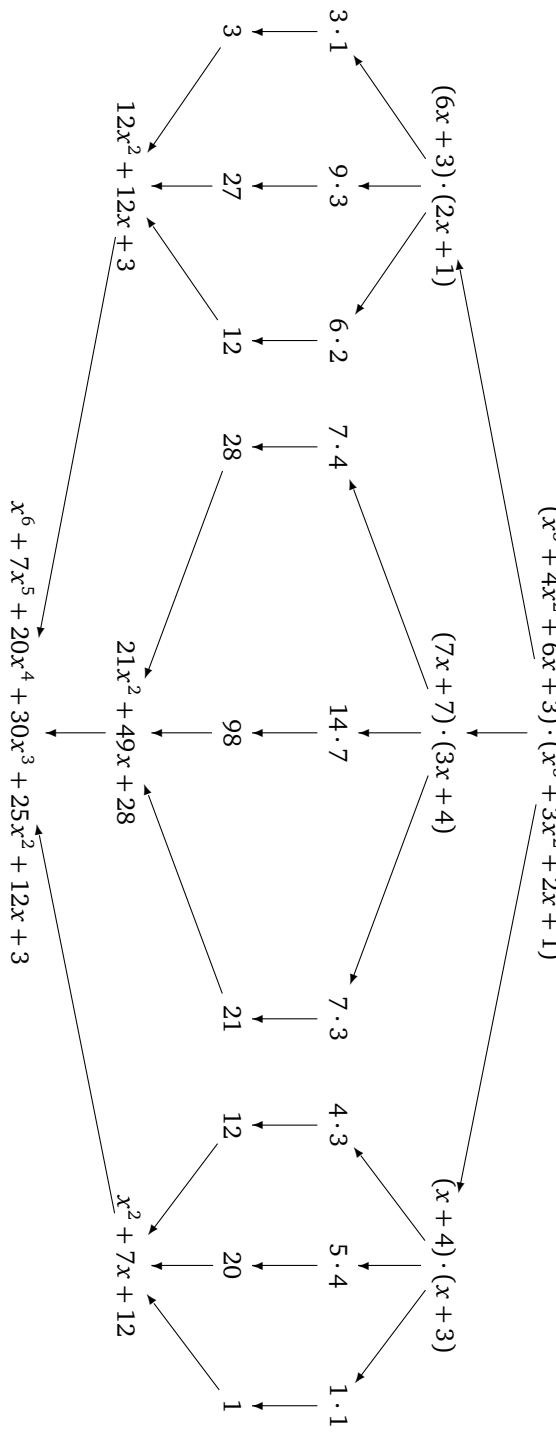


Figure 1.1: Karatsuba multiplication of $x^3 + 4x^2 + 6x + 3$ and $x^3 + 3x^2 + 2x + 1$.

Exponentiation

Once we have dealt with multiplication, it is time to concentrate at exponentiation. A general method is the same for every input type. Hence, although we present this algorithm in the section devoted to integers and polynomials it can be used with any type for which multiplication is defined. The multiplication does not even have to be commutative for this algorithm to work correctly. However, for some types of objects, there are far more efficient solutions than the one discussed below.

Let a be an element of a ring R, in which we can compute products. The natural exponent a^n is just a product n copies of a. Therefore the simplest algorithm to compute the n-th power is to multiply the intermediate result by a in a loop.

1. *Initialize* $r := a$.
2. *For* $i \in \{1, \ldots, n-1\}$ *update* r *setting* $r := r \cdot a$.
3. *Output* r.

This method requires $n-1$ multiplications. We may significantly reduce the number of multiplications using repeated squaring. Write n as a binary number $(d_k d_{k-1} \ldots d_0)_2$ so that

$$n = 2^k \cdot d_k + 2^{k-1} \cdot d_{k-1} + \cdots + 2 \cdot d_1 + d_0,$$

where $d_i \in \{0, 1\}$ for $i \in \{0, \ldots, k\}$. It follows that the power a^n can be expressed as a product

$$a^n = a^{d_0} \cdot \left(a^2\right)^{d_1} \cdots \left(a^{2^k}\right)^{d_k}.$$

If a digit d_i is null, then the corresponding power of a does not contribute to the overall result and may be omitted. Thus, we can find the n-th power of a by computing successive powers a^{2^i} and taking a product of those that correspond to nonzero digits in binary expansion of the exponent.

Algorithm 1.1.10. *Given an element a of a ring R and a natural exponent $n \geq 1$, this algorithm computes a^n.*

1. *Let $(d_k \ldots d_0)_2$ be the binary representation of the exponent n;*
2. *Initialize r to be the unit element of R (i.e. $r = 1$ in R) and set $b := a$;*
3. *For each $i \in \{0, \ldots, k\}$ do:*
   a) *If $d_i = 1$, then update r setting $r := r \cdot b$.*
   b) *Replace b by its square b^2.*
4. *Output r.*

Time complexity analysis. This algorithm needs at most $2 \cdot \lceil \lg(n) \rceil$ multiplication in the ring R.

Division and pseudo-division

The last of the four basic arithmetic operations is division. In general a quotient of two integers may no longer be an integer. Take for example $7/2$. It is a rational number but not an integer. Likewise a quotient of two polynomials is a rational function, but not necessarily a polynomial. It is where the division with a reminder comes into play.

We claim that in a ring $K[x]$ of polynomial in variable x **over a field** K the quotient and remainder of polynomial division are well defined and unique. (The same is true for integers.) Take two polynomials f and g with coefficients in some field. Denote their degrees respectively by $m := \deg f$, $n := \deg g$ and assume that $g \neq 0$. If $m < n$, then we can write

$$f = 0 \cdot g + r, \qquad \text{where} \quad r = f,\ \deg r < \deg g. \tag{1.2}$$

Suppose now that $m \geq n$. Let $c_f := \operatorname{lc}(f)$ and $c_g := \operatorname{lc}(g)$ be the leading coefficients of the two polynomials. Take

$$r := f - \frac{c_f}{c_g} \cdot g \cdot x^{m-n}. \tag{1.3}$$

It is clear that the leading term of f gets canceled and so $\deg r < \deg f$. Now, if $\deg r < \deg g$ then we are in situation of (1.2). Otherwise we may repeat the above reduction using r in place of f. This procedure produces a sequence of polynomials with decreasing degrees. Hence it must terminate after finitely many steps.

Proposition 1.1.11. *Let $f, g \in K[x]$ be two polynomials with coefficients in some field K. If g is nonzero, then there are unique polynomials $q, r \in K[x]$ such that*

$$f = q \cdot g + r \qquad \textit{and} \qquad \deg r < \deg g. \tag{1.4}$$

Proof. The existence follows from the preceding discussion. As for the uniqueness, suppose that there are two pairs (q_1, r_1) and (q_2, r_2) satisfying the assertion. Then

$$q_1 \cdot g + r_1 = f = q_2 \cdot g + r_2,$$

thus $r_1 - r_2$ is divisible by g, but $\deg(r_1 - r_2) \leq \max\{\deg r_1, \deg r_2\} < \deg g$. Hence $r_1 - r_2 = 0$ and it follows that $q_1 \cdot g = q_2 \cdot g$ so also $q_1 = q_2$. □

The polynomials q and r of Proposition 1.1.11 are called respectively the *quotient* and *remainder* of the division of f by g. In what follows we will write $(f \bmod g)$ for the remainder. In other words, we think of the computation of the remainder as of a binary operator. The earlier discussion provides a method for computing q and r. Let us rewrite is in form of an algorithm.

Algorithm 1.1.12 (Polynomial long division)**.** *Let $f = f_m x^m + \cdots + f_1 x + f_0$ and $g = g_n x^n + \cdots + g_1 x + g_0$ be two polynomials with coefficients in a field K. Assume that $g \neq 0$. This algorithm computes the quotient q and the remained r of f modulo g.*

1. *Let $c := 1/g_n$ be the inverse of the leading coefficient of g.*

2. *Initialize the quotient* $q := 0$ *and remainder* $r := f$.
3. *As long as the degree of* r *is greater than or equal to the degree of* g *do:*
   a) *Compute the next coefficient of the quotient setting* $d := c \cdot \mathrm{lc}(r)$.
   b) *Update the quotient setting* $q := d + x \cdot q$ (*i.e. prepend* d *at the beginning of the list of coefficients of* q).
   c) *Update the remainder setting* $r := r - d \cdot g \cdot x^{\deg r - n}$.
4. *Output the quotient* q *and remainder* r.

Time complexity analysis. We should count the number of arithmetic operations used by the algorithm. Each iteration of the main loop reduces the degree of r, hence the loop will be executed at most $m - n + 1$ times. Step (3a) uses exactly one scalar multiplication. Step (3b) does not involve any arithmetic operations and step (3c) multiplies a polynomial by a scalar and then subtracts two polynomials, hence uses up to n scalar multiplications and the same number of subtractions. Thus, the total number of arithmetic operations in the algorithm does not exceed:

- $2 \cdot (m - n + 1) \cdot (n + 1)$ multiplications,
- $(m - n + 1) \cdot n$ subtractions.

Consequently, we see that the time complexity of Algorithm 1.1.12 is

$$O(m \cdot n \cdot \text{complexity of arithmetic operations on coefficients}).$$

Exercise 2. The reduction of the remainder in step (3c) always cancels the leading term of r, nevertheless it may possibly kill some more terms, too. If these additionally removed terms directly succeed the leading term, then next few iterations of the loop can be omitted. They would just prepend zero to the list of coefficients of q and leave r untouched. Improve Algorithm 1.1.12 so that this empty iterations are not executed (the quotient q must still be correctly computed).

An integer counterpart of Proposition 1.1.11 is the following one:

Proposition 1.1.13. *If* a, b *are two integers with* $b \neq 0$, *then there are unique integers* q, r *such that*

$$a = q \cdot b + r \qquad \text{and} \qquad 0 \leq r < |b|.$$

The arguments are fully analogous to the polynomial case, except that in place of the degree we now use absolute value. We let the reader write down a rigorous proof. The corresponding algorithm resembles Algorithm 1.1.12 but is significantly more complicated. For details we refer the reader to a throughout exposition in Knuth's book [42, Algorithm D, Section 4.3.1].

The assumption of Proposition 1.1.11 that K is a field is indispensable as the assertion no longer holds if one replaces K with an arbitrary ring. Yet, there are many situations, when we need to divide polynomials over rings which are not fields. As we

will later see, the rings that are very important include: the integers $\mathbb{Z}$, the polynomial rings over the integers $\mathbb{Z}[x]$ and $\mathbb{Z}[x_1,\ldots,x_k]$ and polynomial rings over fields $K[x]$, $K[x_1,\ldots,.x_k]$. Neither of them is a field, though. Anyway, if we look carefully at Algorithm 1.1.12, we may notice that there is only one place where coefficients are divided, it is step (3a). Furthermore it is always the leading coefficient of g that we are dividing by. One consequence of this fact is:

Observation 1.1.14. *If the polynomial g is monic (which means that $\mathrm{lc}(g)=1$), then Algorithm 1.1.12 works correctly without the assumption that K is a field.*

Now suppose that g is not monic. We can still remedy the problem of the division being possibly undefined, by pre-multiplying the polynomial f by an appropriate power of $\mathrm{lc}(g)$, so that all the divisions that occur will be well defined.

Proposition 1.1.15. *Let R be an integral domain, $f,g\in R[x]$ be two polynomials and assume that g is nonzero. Then there are unique polynomials $q,r\in R[x]$ such that*

$$\mathrm{lc}(g)^d\cdot f = q\cdot g + r, \tag{1.5}$$

where $\deg r<\deg g$ and $d=\max\{0,\deg f-\deg g+1\}$.

Proof. Proceed by induction on the degree of f. For $\deg f<\deg g$, the trivial solution is $q=0$ and $f=r$. Assume that $\deg f\geq\deg g>0$ and the assertion holds for all polynomials of degree smaller than $\deg f$. Denote the degree of f by m, the degree of g by n and take a difference

$$h := \mathrm{lc}(g)\cdot f - \mathrm{lc}(f)\cdot g\cdot x^{m-n}. \tag{1.6}$$

It is clear that the leading terms cancel, hence $\deg h<m$ and we may apply the inductive hypothesis to it. There are unique polynomials $q_1,r_1\in R[x]$ such that

$$\mathrm{lc}(g)^{d_1}\cdot h = q_1\cdot g + r_1, \tag{1.7}$$

with $d_1=\max\{0,\deg h-n+1\}$ and $\deg r_1<n$. We claim that $d>d_1$. Indeed, the number d_1 equals either 0 or $\deg h-n+1$. In the first case, $d_1<1\leq m-n+1=d$, since $m\geq n$, by assumption. On the other hand, if $d_1=\deg h-n+1$, the claim follows from the fact that $\deg h<m$.

Combining (1.6) with (1.7) and rearranging the terms we write

$$\mathrm{lc}(g)^{d_1+1}\cdot f = \left(q_1+\mathrm{lc}(g)^{d_1}\cdot\mathrm{lc}(f)\cdot x^{m-n}\right)\cdot g + r_1.$$

Multiply both sides by $\mathrm{lc}(g)^{d-d_1-1}$. By the previous paragraph, it is a non-negative power of an element of R, hence an element of R itself. This yields:

$$\mathrm{lc}(g)^d\cdot f = \left(q_1\cdot\mathrm{lc}(g)^{d-d_1-1}+\mathrm{lc}(g)^{d-1}\cdot\mathrm{lc}(f)\cdot x^{m-n}\right)\cdot g + \mathrm{lc}(g)^{d-d_1-1}\cdot r_1 = q\cdot g + r.$$

It remains to prove the uniqueness of q and r. Suppose there are q_1, q_2 and r_1, r_2 such that $\deg r_1, \deg r_2 < \deg g$ and

$$q_1 \cdot g + r_1 = \mathrm{lc}(f)^d \cdot f = q_2 \cdot g + r_2.$$

It follows that $r_1 - r_2$ is divisible by g, but this is possible only when $r_1 - r_2 = 0$ due to the fact that $\deg g > \deg(r_1 - r_2)$. Thus also $q_1 \cdot g = q_2 \cdot g$ and so $q_1 = q_2$, as claimed. □

Definition 1.1.16. The polynomials q and r in the above propositions are called respectively: the *pseudo-quotient* and *pseudo-remainder* of f modulo g.

Algorithm 1.1.17 (Polynomial pseudo-division)**.** *Let f and g be two polynomials with coefficients in an integral domain R. Denote $m := \deg f$, $n := \deg g$ and assume that g is nonzero. This algorithm computes the pseudo-quotient q and pseudo-remained r of f modulo g.*

1. *If the degree of g is strictly greater that the degree of f, then output $q = 0$, $r = f$ and terminate.*
2. *Let $c := \mathrm{lc}(g)$ be the leading coefficient of g.*
3. *Initialize the pseudo-quotient $q := 0$ and pseudo-remainder $r := f$.*
4. *As long as the degree of r is is greater than or equal to the degree of g do:*
   a) *Compute the next coefficient of the pseudo-quotient $d := \mathrm{lc}(r) \cdot c^{m-n}$.*
   b) *Update the pseudo-quotient setting $q := d + x \cdot q$,*
   c) *Update the pseudo-remainder setting $r := c \cdot r - \mathrm{lc}(r) \cdot g \cdot x^{\deg r - n}$.*
5. *Output q and r.*

It is evident that the asymptotic behavior of pseudo-division (in terms of number of arithmetic operations on coefficients) is the same as of polynomial division (Algorithm 1.1.12), although the exact number of operations is slightly greater. The following result is a useful consequence of Proposition 1.1.15. It is sometimes called "Little Bézout Theorem", however it was proved by R. Descartes[2] already in 1637, hence long before É. Bézout[3] was even born.

Proposition 1.1.18. *Let f be a polynomial over an integral domain R and $\xi \in R$. Then ξ is a root of f if an only if $(x - \xi)$ divides f in $R[x]$.*

Proof. Take any $\xi \in R$, not necessarily a root of f. Proposition 1.1.15 assert that there are unique polynomials q and r such that

$$f = q \cdot (x - \xi) + r$$

and $\deg r < \deg(x - \xi) = 1$. This means that r must be a constant. Evaluating both sides at ξ we get $f(\xi) = 0 + r(\xi) = r$. In particular $f(\xi) = 0$ if and only if $r = 0$. Where the latter condition means that $x - \xi$ divides f. □

[2] René Descartes (born 31 March 1596, died 11 February 1650) was a French philosopher and mathematician.

[3] Étienne Bézout (born 31 March 1730, died 27 September 1783) was a French mathematician.

Corollary 1.1.19. *Over an integral domain, a polynomial of degree $n > 0$ has no more than n roots.*

Binomial coefficients

Binomial coefficients appear in many algorithm, hence we should learn how to compute them. In Section 4.4 of Chapter 4 we will discuss an effective method to compute binomial coefficients in bulk using fast Fourier transform. But before we deal with advanced techniques, we should learn elementary ones. This is the goal of this section.

In school we usually learn the following definition of a binomial coefficient:

$$\binom{n}{k} := \frac{n!}{(n-k)!\,k!}, \tag{1.8}$$

where n, k are two non-negative integers and $k \leq n$. In addition, for $k > n$ we set $\binom{n}{k} := 0$. This definition has a natural interpretation in combinatorics, since $\binom{n}{k}$ is the numbers of subsets of size k of a set of size n. Unfortunately, formula (1.8) is horrible when it comes to actually computing a binomial coefficient. Imagine we want to calculate $\binom{100}{98}$. The answer is 4 950. But if we try to compute the numerator and denominator separately, we would have to deal with numbers having over 150 decimal digits. A better way to compute the binomial coefficient is to use the following recursive formula

$$\binom{n}{k} = \begin{cases} \frac{n}{k} \cdot \binom{n-1}{k-1}, & \text{if } n \geq k > 0, \\ 1, & \text{if } n \geq 0 \text{ and } k = 0, \\ 0, & \text{otherwise.} \end{cases} \tag{1.9}$$

Additionally we have the following well known identity to help:

Observation 1.1.20. $\binom{n}{k} = \binom{n}{n-k}$.

The proof is trivial. Combining these two facts we obtain the first method for computing binomial coefficients.

Algorithm 1.1.21. *Given two non-negative integers n, k, this algorithm computes the binomial coefficient $\binom{n}{k}$.*

1. *If $n < k$ then output 0 and terminate.*
2. *Set $k := \min\{k, n-k\}$.*
3. *Initialize the result $r := 1 = \binom{n-k}{0}$.*
4. *For $i \in \{1, \ldots, k\}$ update the result setting $r := \dfrac{(n-k+i)\cdot r}{i}$.*
5. *Output r.*

The correctness of this algorithm is obvious in view of the previous two results. In order to compute the binomial coefficient $\binom{n}{k}$ this algorithm uses $\min\{k, n-k\}$ integer multiplications and divisions, Hence its asymptotic complexity is $O\big(2^d \cdot M(d)\big)$, where

$d = \lg n$ is the size (i.e. number of binary digits) of n and $M(d)$ denotes the complexity of multiplication of two integers of size d. For our example $\binom{100}{98}$ the algorithm uses just two products and two divisions:

$$\binom{100}{98} = \binom{100}{2} = \frac{100 \cdot \binom{99}{1}}{2} = \frac{100 \cdot \frac{99 \cdot \binom{98}{0}}{1}}{2} = 4950.$$

Probably the best known identity for binomial coefficients is the one leading to Pascal triangle construction.

Observation 1.1.22. $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$.

Proof. Starting from the right hand side we have

$$\begin{aligned}\binom{n-1}{k-1} + \binom{n-1}{k} &= \frac{(n-1)!}{(n-k)!\,(k-1)!} + \frac{(n-1)!}{(n-k-1)!\,k!} \\ &= (n-1)! \cdot \frac{k + (n-k)}{(n-k)!\,k!} = \frac{n!}{(n-k)!\,k!} = \binom{n}{k}.\end{aligned}$$

□

This observation let us compute binomial coefficients using only additions without any multiplications.

Algorithm 1.1.23. *Given a non-negative integer n, this algorithm computes all binomial coefficients $\binom{k}{j}$ for $0 \le j \le k \le n$.*

1. *Set $\binom{0}{0} = 1$.*
2. *For every $k \le n$ do:*
   a) *Initialize $\binom{k}{0} := 1$ and $\binom{k}{k} := 1$.*
   b) *For $j \le k/2$ compute $\binom{k}{j} := \binom{k-1}{j-1} + \binom{k-1}{j}$ and set $\binom{k}{k-j} := \binom{k}{j}$.*

The best way to visualize the algorithm is in form of a triangular diagram, known as a *Pascal triangle*. Here we present a Pascal triangle of height 5:

$$\begin{array}{ccccccccc}
 & & & & \binom{0}{0}=1 & & & & \\
 & & & \swarrow & & \searrow & & & \\
 & & \binom{1}{0}=1 & & & & \binom{1}{1}=1 & & \\
 & \swarrow & & \searrow & & \swarrow & & \searrow & \\
 \binom{2}{0}=1 & & \binom{2}{1}=2 & & & & \binom{2}{2}=1 & & \\
 \swarrow & & \searrow & & \swarrow & & \searrow & & \swarrow \\
 \binom{3}{0}=1 & & \binom{3}{1}=3 & & \binom{3}{2}=3 & & \binom{3}{3}=1 & & \\
 \swarrow & & \searrow & & \swarrow & & \searrow & & \swarrow \\
 \binom{4}{0}=1 & & \binom{4}{1}=4 & & \binom{4}{2}=6 & & \binom{4}{3}=4 & & \binom{4}{4}=1
\end{array}$$

A much faster classical algorithm for computing the binomial coefficient is based on the following 19th century result.

Theorem 1.1.24 (Kummer, 1852)**.** *Let p be a prime number and n, k be two non-negative integers. Then the maximal power with which p divides the binomial coefficient $\binom{n}{k}$ equals the number m is of carries when k is added to $n-k$ at base p, i.e.*

$$p^m \mid \binom{n}{k} \qquad \textit{but} \qquad p^{m-1} \nmid \binom{n}{k}.$$

We omit the proof referring the reader to the original paper of Kummer [48]. Instead we present a short example.

Example 2. Take $n = 100$, $k = 70$ and $p = 5$. At base 5 we have $k = (240)_5$ and $n-k = (110)_5$. Hence, in order to add k to $n-k$ at base 5 we need only one carry. Consequently $5 \mid \binom{100}{70}$ but $5^2 \nmid \binom{100}{70}$. Indeed

$$\binom{100}{70} = 2^4 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11 \cdot 19 \cdot 31 \cdot 37 \cdot 41 \cdot 43 \cdot 47 \cdot 71 \cdot 73 \cdot 79 \cdot 83 \cdot 89 \cdot 97.$$

The application of the theorem is clear. We take all the primes not greater than n. For each prime we find the corresponding exponent m using Kummer's theorem. Then to get the sought binomial coefficient, we just multiply all of them. The only missing part is how to find the primes. We will deal with this problem in Chapter 5.

Algorithm 1.1.25. *Let n, k be two non-negative integers. Given a list $\mathscr{P}$ of primes not exceeding n, this algorithm computes the binomial coefficient $\binom{n}{k}$.*

1. *If $k > n$ then output 0 and terminate.*
2. *If either $k = 0$ or $k = n$ then output 1 and terminate.*
3. *Initialize the result $r := 1$.*
4. *For every prime $p \leq n$ do:*
   a) *Add k to $n-k$ at base p using Algorithm 1.1.4 and count the number m of carries.*
   b) *Update the result setting $r := r \cdot p^m$.*
5. *Output $\binom{n}{k} = r$.*

Remark 1. We presented the algorithm in a bare-bone form for the sake of clarity. Nevertheless, there are two obvious optimizations to it. Let $p \leq n$ be a prime. If $p > \max\{k, n-k\}$, then k and $n-k$ are single digit numbers at base p. Hence there will be exactly one carry. Thus we may immediately set $m = 1$ in this case without converting the two numbers to base p.

Secondly, if $p > \sqrt{n}$ then the result of the addition $(n-k)+k = n$ is a two-digit number at base p. Therefore, there is at most one possible carry. It occurs if and only if both the reminders $(k \bmod p)$ and $\big((n-k) \bmod p\big)$ are greater than $(n \bmod p)$. We leave it to the reader to improve the pseudo-code accordingly.

Finally we present some upper and lower bounds for the size of a binomial coefficient. Let us begin with a crude but extremely simple one. Fix a set X of size n. Then X has 2^n subsets in total and $\binom{n}{k}$ of them has size k. Thus we have:

Observation 1.1.26. $\binom{n}{k} \leq 2^n$.

As simple as the above observation is, it is not very useful when a greater precision is needed. Fortunately there are better estimates. Recall that $(1 + 1/k)^k$ is increasing and convergent to the base of the natural logarithm $e \approx 2.71828\ldots$. We begin with an estimate for the factorial.

Lemma 1.1.27. $(k/e)^k \leq k! \leq k^k$.

Proof. The right inequality is trivial. In order to prove the left one, we proceed by induction on k. The assertion is certainly true for $k = 1$. Assume that it holds for $k-1$, hence $(k-1)! \geq (k-1/e)^{k-1}$. Multiplying both sides and using the fact that $e \geq (1+1/k)^k$ we have:

$$\begin{aligned} k! \geq \frac{1}{e^{k-1}} \cdot k \cdot (k-1)^{k-1} &\geq \frac{1}{e^k} \cdot k \cdot \left(1 + \frac{1}{k}\right)^k \cdot (k-1)^{k-1} \\ &= \frac{1}{e^k} \cdot \frac{(k^2-1)^k}{k^k - k^{k-1}} = \frac{k^k}{e^k} \cdot \frac{(k^2-1)^k}{k^{2k} - k^{2k-1}} \\ &= \left(\frac{k}{e}\right)^k \cdot \frac{(k^{2k} - k^{2k-1}) + \sum_{j=2}^{k} \binom{k}{j} (-1)^j \cdot k^{2k-2j}}{k^{2k} - k^{2k-1}}. \end{aligned}$$

Grouping the terms of the sum in the numerator into pairs we can easily show that the sum is non-negative and this proves the assertion. □

Proposition 1.1.28. $\left(\frac{n}{k}\right)^k \leq \binom{n}{k} \leq \left(\frac{ne}{k}\right)^k$.

Proof. First we prove the lower bound. By (1.9) we obtain

$$\binom{n}{k} = \prod_{j=0}^{k-1} \frac{n-j}{k-j} \leq \prod_{j=0}^{k-1} \frac{n}{k} = \left(\frac{n}{k}\right)^k.$$

The other inequality follows from the preceding lemma, indeed we have

$$\binom{n}{k} = \frac{n \cdot (n-1) \cdots (k+1)}{k!} \leq \frac{n^k}{k!} \leq \left(\frac{ne}{k}\right)^k. \qquad \square$$

Polynomial evaluation and differentiation

So far our discussion has been based on similarities between integers and polynomials. But there are operations that are meaningful only to later ones. These include evaluation and composition as well as differentiation. First let us deal with the evaluation

of a polynomial. Let $f = f_n x^n + \cdots + f_1 x + f_0$ be a polynomial with coefficients in a ring P. Further let α be an element of some ring R containing P. Evaluating f at α in a totally naïve way means to compute

$$f_n \cdot \underbrace{\alpha \cdots \alpha}_{n \text{ times}} + \cdots + f_1 \cdot \alpha + f_0.$$

This formula requires $n + \cdots + 2 + 1 = {}^1\!/_2 \cdot n \cdot (n+1)$ multiplications and n additions. Hence that a naïve evaluation of polynomials has complexity $O(n^2)$. Rearranging the terms, one reduces this easily to a linear time. Write f as

$$f = f_0 + x \cdot \big(f_1 + x \cdot \big(\cdots + x \cdot (f_{m-1} + x \cdot f_n) \cdots \big)\big).$$

This way we need only n multiplications and the same number of additions.

Algorithm 1.1.29 (Horner scheme)**.** *Given a polynomial $f = f_n x^n + \cdots + f_1 x + f_0$ and an argument α, this algorithm computes $f(\alpha)$.*

1. *Initialize the result $r := f_n$.*
2. *For $i = n-1, n-2, \ldots, 0$ update the result, setting $r := r \cdot \alpha + f_i$.*
3. *Output r.*

Remark 2. In the preceding discussion we have assumed that the argument α belongs to a ring R, which is potentially bigger than the ring of coefficients P. In particular, if $R = P[x]$ and $\alpha = g$ is a polynomial, then the result is the composition $f \circ g$.

The second operation used with polynomials, that has little sense for integers is computation of a derivative. (There is a notion of an *arithmetic derivative* but we will not delve into this subject.) As we will see in Chapter 5, the fact that there is a well defined and nicely behaving derivative of every polynomial, allows for simple algorithms computing radicals and square-free factors.

Differentiation of polynomials is straightforward due to its additivity and the well known rule $(f_k x^k)' = k f_k x^{k-1}$.

Algorithm 1.1.30. *Given a polynomial $f = f_n x^n + \cdots + f_1 x + f_0$, this algorithm computes its derivative.*

1. *For each $0 \le i \le n-1$, take $h_i := (i+1) \cdot f_{i+1}$.*
2. *Output $h = h_0 + h_1 x + \cdots + h_{n-1} x^{n-1}$.*

Time complexity analysis. The cost of the above algorithm is clearly linear. In fact, in order to compute a derivative of a polynomial of degree n, this algorithm uses n multiplications of coefficients by positive integers and no other operations.

In certain applications one needs to simultaneously evaluate a polynomial f and some of its derivatives at the same argument α. Of course, this can be achieved by repeated applications of the previous two algorithms, but there are better methods. As before let $f = f_n x^n + \cdots + f_1 x + f_0$ be a polynomial with coefficients in a ring P and α

be an element of a ring R containing P. We will compute $f(\alpha)$, ${}^{f'(\alpha)}/_{1!}$, ${}^{f''(\alpha)}/_{2!}\ldots$ up to ${}^{f^{(m)}(\alpha)}/_{m!}$ for some $m \leq n$. Padding f with zeros if necessary, we may assume that there is a known divisor q of $n+1$. In applications, q will be equal to either $n+1$ or $\sqrt{n+1}$. Denote $s(j) := \big((n-j) \bmod q\big)$ and recursively define:

$$T_{-1,i} := f_{n-i-1} \cdot x^{s(i+1)} \qquad \text{for } i \in \{0,\ldots,n-1\}, \tag{a}$$

$$T_{j,j} := f_n \cdot x^{s(0)} \qquad \text{for } j \in \{0,\ldots,m\}, \tag{b}$$

$$T_{j,i} := T_{j-1,i-1} + T_{j,i-1} \cdot x^{s(i-j)-s(i-j-1)+1} \qquad \text{for } j \in \{0,\ldots,m\}, i \in \{j+1,\ldots,n\}. \tag{c}$$

Using the congruence $n \equiv -1 \pmod q$ we may rewrite the last formula as

$$T_{j,i} := \begin{cases} T_{j-1,i-1} + T_{j,i-1} & \text{if } i \not\equiv j \pmod q, \\ T_{j-1,i-1} + x^q \cdot T_{j,i-1} & \text{if } i \equiv j \pmod q. \end{cases} \tag{c'}$$

We need to extend the binomial symbol $\binom{k}{j}$ to negative integers[4]. We set $\binom{-1}{-1} = 1$ and $\binom{k}{-1} = 0$ for all $k \geq 0$. The next lemma is rather technical but very important.

Lemma 1.1.31. *For all valid indices i and j we have*

$$T_{j,i} = x^{s(i-j)} \cdot \sum_{k=j}^{i} \binom{k}{j} \cdot f_{n-i+k} \cdot x^{k-j}.$$

Proof. First, we show that the above identity holds when $i = j$. Indeed, the right hand side equals

$$x^{s(0)} \cdot \sum_{k=j}^{j} \binom{k}{j} \cdot f_{n-j+k} \cdot x^{k-j} = x^{s(0)} \cdot f_n = T_{j,j}.$$

Next, assume that $j = -1$. By our convention for binomial coefficients the right hand side takes a form

$$x^{s(i+1)} \cdot \sum_{i=-1}^{i} \binom{k}{-1} \cdot f_{n-i+k} \cdot x^{k+1} = x^{s(i+1)} \cdot 1 \cdot f_{n-i+1} \cdot x^0$$

and this is $T_{-1,i}$ by (a). In order to conclude the proof, assume that the assertion holds for $T_{j-1,i-1}$ and $T_{j,i-1}$. We will use (c) to show that it holds for $T_{j,i}$. We have

$$\begin{aligned} T_{j,i} &= T_{j-1,i-1} + T_{j,i-1} \cdot x^{s(i-j)-s(i-j-1)+1} \\ &= x^{s(i-j)} \cdot \sum_{k=j-1}^{i-1} \binom{k}{j-1} \cdot f_{n-i+k+1} \cdot x^{k+1-j} \\ &\qquad + x^{s(i-j)-s(i-j-1)+1} \cdot x^{s(i-j-1)} \cdot \sum_{k=j}^{i-1} \binom{k}{j} \cdot f_{n-i+k+1} \cdot x^{k-j} \end{aligned}$$

[4]The reader should be warned that there are a number of different and pairwise incompatible methods to extend the domain of the binomial coefficient beyond the natural numbers. The definition used here is not the only one possible.

Change indexation setting $l := k+1$ and use the fact that $\binom{j-1}{j} = 0$. This yields

$$= x^{s(i-j)} \cdot \sum_{l=j}^{i} \left(\binom{l-1}{j-1} + \binom{l-1}{j} \right) \cdot f_{n-i-l} \cdot x^{l-j}.$$

The assertion follows now from Observation 1.1.22. □

Theorem 1.1.32 (Shaw–Traub, 1974)**.**

$$\frac{f^{(j)}}{j!} = \frac{T_{j,n}}{x^{s(n-j)}}. \tag{1.10}$$

Proof. Fix $k \geq j$ and compute j-th derivative of $f_k x^k$. We have

$$\left(f_k x^k\right)^{(j)} = k \cdot (k-1) \cdots (k-j+1) \cdot f_k \cdot x^{k-j} = \binom{k}{j} \cdot j! \cdot f_k \cdot x^{k-j}.$$

Now, in order to conclude the proof we take sums of both sides with k running over $\{0, \ldots, n\}$. □

Theorem 1.1.32 gives raise to a method of evaluating several derivatives of f at once. We will present two representatives scenarios when one uses this result. First we show how to simultaneously evaluate f and f' at the same argument α. This is useful for Newton–Raphson iterations (see Algorithm 6.6.12) and its variations (e.g. Hensel lifting).

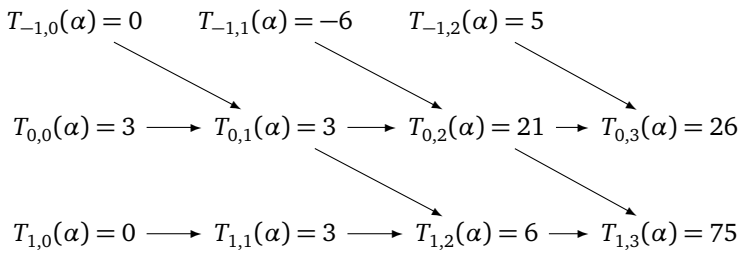
Algorithm 1.1.33. *Given a polynomial $f = f_n x^n + \cdots + f_1 x + f_0$ and an argument α, this algorithm computes $f(\alpha)$ and $f'(\alpha)$.*

1. *Set $q := \lceil \sqrt{n+1} \rceil$ and $N := q^2 - 1$.*
2. *Compute consecutive powers $\alpha^1, \alpha^2, \ldots, \alpha^q$ of α and store then in an auxiliary array.*
3. *For $i \in \{0, \ldots, N-1\}$ initialize $T_{-1,i}(\alpha) := f_{N-i-1} \cdot \alpha^{s(i+1)}$, with the standard convention that $f_k \equiv 0$ for $k > n$.*
4. *Set $T_{0,0}(\alpha)$ and $T_{1,1}(\alpha)$ to $f_N \cdot \alpha^{s(0)}$.*
5. *Using (c') with $x = \alpha$ compute $T_{j,i}(\alpha)$ for $j \in \{0,1\}$ and $i \in \{j+1, \ldots, N\}$.*
6. *Output $f(\alpha) = {}^{T_{0,N}(\alpha)}/_{\alpha^{s(N)}}$ and $f'(\alpha) = {}^{T_{1,N}(\alpha)}/_{\alpha^{s(N-1)}}$.*

Time complexity analysis. For simplicity assume that $n+1$ is a perfect square (hence $N = n$) and that multiplication and division in the ring R occupy the same amount of time. Then Algorithm 1.1.33 uses $2\sqrt{n+1} + n + 2$ multiplications/divisions, $2n+2$ additions and one square root computation. For comparison, in order to compute $f(\alpha)$ and $f'(\alpha)$ using Algorithm 1.1.30 together with Horner scheme we would need $(2n-1)$ multiplications in R plus $(n-1)$ multiplications of coefficients by integers.

Remark 3. There is a common variation of the above algorithm that uses $q = 2 \cdot \lceil \sqrt{n/2} \rceil$. This reduces the number of operations when $n+1$ is not a perfect square.

Example 3. Take $f = x^3 - 2x + 5$. We will evaluate f and f' at $\alpha = 3$. In our case $q = 2$ and the following diagram shows the successive steps of Algorithm 1.1.33:



Consequently $f(\alpha) = {}^{T_{0,3}(\alpha)}/_{\alpha^0} = 26$ and $f'(\alpha) = {}^{T_{1,3}(\alpha)}/_{\alpha^1} = 25$.

A second typical use of Shaw–Traub theorem is to compute all derivatives of a polynomial f in order to "shift" it using Taylor expansion. We will see some applications of such shift in Chapter 6. The application include depressing of a polynomial (Observation 6.1.8) and VAS root isolation (Algorithm 6.5.23). Let $f = f_n x^n + \cdots + f_1 x + f_0$ be a given polynomial. Denote $g := f(x + \alpha)$ for some scalar α. Then g is again a polynomial and our task is to find its coefficients. Writing g explicitly we have

$$g_j = \sum_{i=j}^{n} \binom{i}{j} \cdot f_i \cdot \alpha^{i-j}$$

for $j \in \{0, \ldots, n\}$. This formula is rather expensive to evaluate. Fortunately, we can do better if we use Taylor expansion. We have

$$g = \sum_{j=0}^{n} \frac{f^{(j)}(\alpha)}{j!} \cdot x^j.$$

and so Theorem 1.1.32 yields $g_j = {}^{T_{j,n}(\alpha)}/_{\alpha^{s(n-j)}}$. Taking $q = n+1$ we obtain the following method for shifting polynomials:

Algorithm 1.1.34. *Given a polynomial $f = f_n x^n + \cdots + f_1 x + f_0$ and a scalar α, this algorithm constructs the polynomial $g = f(x + \alpha)$.*

1. *Compute the consecutive powers $\alpha^1, \alpha^2, \ldots, \alpha^n$ of α and store them in an auxiliary array.*
2. *Initialize $T_{-1,i}(\alpha) := f_{n-i-1}\alpha^{n-i}$ for $i \in \{0, \ldots, n-1\}$ and $T_{j,j}(\alpha) := f_n \alpha^n$ for $j = \{0, \ldots, n\}$.*
3. *For every $j \in \{0, \ldots, n\}$ and every $i \in \{j+1, \ldots, n\}$ set*
$$T_{j,i}(\alpha) := T_{j-i,i-1}(\alpha) + T_{j,i-1}(\alpha).$$
4. *Output $g = \sum_{j=0}^{n} \frac{T_{j,n}(\alpha)}{\alpha^{n-j}} \cdot x^j$.*

Time complexity analysis. This algorithm requires $2n-1$ multiplication, n divisions and $n(n+1)/2$ additions.

Exercise 3. Find a general formula (depending on n, m and q) for a number of additions, multiplication and divisions needed to compute first m derivatives of f using Theorem 1.1.32.

We will now present an alternative method of shifting a polynomial. The following algorithm utilizes a divide et impera approach, where the polynomial is recursively split in half at every step. The correctness of the algorithm is obvious.

Algorithm 1.1.35. *Let $f = f_n x^n + \cdots + f_1 x + f_0$ be a polynomial and $N = 2^m$ be a power of 2 greater than or equal to $n+1$. Given a scalar α and precomputed polynomials $(x+\alpha)^{2^i}$ for $i < m$, this algorithm constructs the polynomial $g = f(x+\alpha)$.*

1. *If f is constant then return $g = f$.*
2. *Split f into its lower and upper parts $f = f_u \cdot x^{N/2} + f_l$, where*

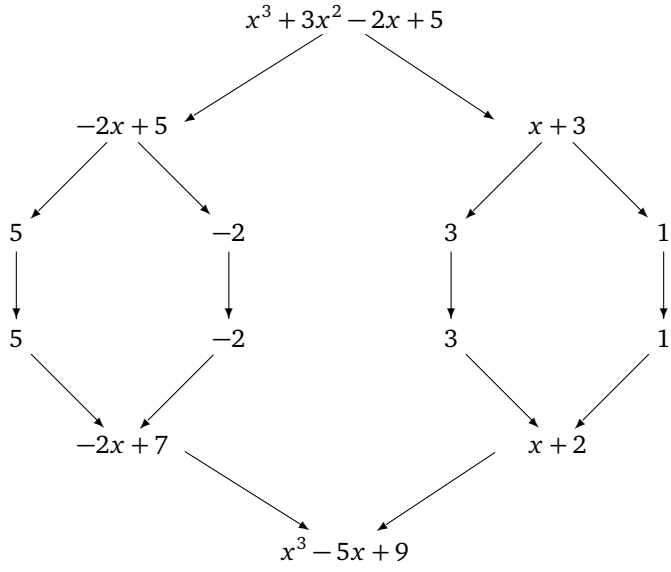
$$f_u = f_n x^{n-N/2} + \cdots + f_{N/2+1} + f_{N/2}, \qquad f_l = f_{N/2-1} x^{N/2-1} + \cdots + f_1 x + f_0.$$

3. *Execute this algorithm recursively to construct polynomials*

$$g_u = f_u(x+\alpha) \qquad \text{and} \qquad g_l = f_l(x+\alpha).$$

4. *Output $g = (x+\alpha)^{N/2} \cdot g_u + g_l$.*

Example 4. Take $f = x^3 + 3x^2 - 2x + 5$ and $\alpha = -1$. The following diagram present how Algorithm 1.1.35 works:



Time complexity analysis. The precise number of arithmetic operations used by Algorithm 1.1.35 is difficult to calculate. Hence we will only analyze its asymptotic complexity. Let M be the complexity of polynomial multiplication. Hence $M(d) = O(d^2)$ for long multiplication and $M(d) \approx O(d^{1.585})$ in case of Karatsuba method. Later, in Chapter 4, we will learn an even faster method for which $M(d) = O(d \cdot \lg d)$. Let n be the degree of the polynomial we are shifting and denote $m := \lceil \lg(n+1) \rceil$. Algorithm 1.1.35 uses precomputed polynomials $(x+\alpha)^{2^i}$ for $i < m$. They can be constructed by successive squaring in time $\sum_{i=0}^{m-2} M(2^i)$. Now look at the algorithm itself. At level i of the recursion tree there are 2^i nodes and at each node the algorithm performs one multiplication of polynomials of degree 2^{m-i-1}. Consequently the asymptotic time complexity of the algorithm is

$$\sum_{i=0}^{m-2} M(2^i) + \sum_{i=0}^{m} 2^i \cdot M(2^{m-i-1}).$$

With fast polynomial multiplication, when $M(d) = O(d \cdot \lg d)$, this formula simplifies to $O(n)$. Hence, Algorithms 1.1.34 and 1.1.35 have the same asymptotic complexity. Nevertheless, for high-degree polynomials the later one is reported to have superior performance in practice (c.f. [88]).

Integer square root

The next algorithm is specific to integers. Let $n = m^2$ be a perfect square. A typical way to find m is to approximate by Heron's method (see Section 6.6 of Chapter 6) with precision < 0.5, the square root $\sqrt{n}$ using floating point numbers. Once we have $\tilde{x} \in \mathbb{R}$ such that $|\tilde{x} - \sqrt{n}| < 0.5$ we round $\tilde{x}$ to the nearest integer m and check whether $m^2 = n$. This is the most common procedure of computing integer square roots. However, it is not the only one. There are other methods that have their own merits. The one described below computes the root digit by digit and uses only integer arithmetic. Hence it is convenient for hand computations or for extremely large integers.

First we present a sub-procedure of a rather technical nature.

Algorithm 1.1.36. *Let $m = (m_k \dots m_0)_\beta$, $n = (n_l \dots n_0)_\beta$ be two integers at base β such that $(\beta-1) \cdot (2m\beta + \beta - 1) \le n$. This algorithm finds the biggest single-digit integer $d \in \{0, \dots, \beta-1\}$ satisfying the condition $d \cdot (2m\beta + d) \le n$.*

1. *Start with a rough estimate $d := \lfloor n/(2m\beta+1) \rfloor$.*
2. *While $(d+1) \cdot (2m\beta + d + 1) \le n$, increment d by one.*
3. *While $d \cdot (2m\beta + d) > n$, decrement d by one.*
4. *Output d.*

The correctness of this algorithm is obvious. We now present the main algorithm that computes the integer square root. The algorithm to some extend resembles the procedure of long division. Like in long division, the result is constructed digit after digit, starting from the most significant one. The idea is first to group the digits of n

into pairs. Then for every such pair find the largest single-digit integer d such that $d \cdot (2m\beta + d)$ does not exceed the number $(c_k \dots c_0 n_j n_{j-1})_\beta$, where $c = (c_k \dots c_0)_\beta$ is a carry. More formally we write the algorithm as follows:

Algorithm 1.1.37 (Integer square root). *Let $n = (n_k \dots n_0)_\beta$ be a positive integer. Padding n with a leading zero if necessary, assume that $k+1$ is even. If n is a perfect square, then this algorithm computes $m \in \mathbb{N}$ such that $m^2 = n$.*

1. *Initialize the result $m := 0$, carry $c := 0$ and digit index $j := k$.*
2. *As long as $j > 0$ do:*
   a) *Append next two to digits to the carry setting $c := c\beta^2 + (n_j n_{j-1})_\beta$.*
   b) *Using Algorithm 1.1.36 find d such that $d \cdot (2m\beta + d) \leq c$.*
   c) *Update the carry setting $c := c - d \cdot (2m\beta + d)$.*
   d) *Append the computed digit d to the result setting $m := m\beta + d$.*
   e) *Decrease the index setting $j := j - 2$.*
3. *If the carry is nonzero then n is not a perfect square and we should raise an error.*
4. *Otherwise output m.*

Proof of correctness. Let $m = (m_l \dots m_0)_\beta$ be the integer square root of n and $\tilde{m} = (m_l \dots m_{i+1} 0 \dots 0)_\beta = m_l \beta^l + \dots + m_{i+1}\beta^{i+1}$ be the part computed after $l - i$ iterations of the algorithm. We have

$$n = m^2 = \tilde{m}^2 \beta^{2i+2} + m_i \cdot (2\tilde{m}\beta + m_i) \cdot \beta^{2i+1} + \cdots,$$

where the part represented by dots is strictly smaller than β^{2i+1}. It follows that m_i is the largest single-digit integer d subject to the condition

$$d \cdot (2\tilde{m}\beta + d) \leq \left(n - \tilde{m}\beta^{2i+2}\right) \cdot \beta^{-(2i+1)}.$$

Group the digits of n into pairs:

$$n = (n_k\beta + n_{k-1}) \cdot \beta^{k-1} + \dots + (n_{2i+2}\beta + n_{2i+1}) \cdot \beta^{2i+1} + \cdots.$$

Again, the tail replaced by dots is strictly smaller than β^{2i+1}, hence can be ignored. Consequently m_i is the largest single-digit integer d such that

$$d \cdot (2\tilde{m}\beta + d) \leq \left((N - k\beta + n_{k-1}) \cdot \beta^{k-2i-2} + \dots + (n_j\beta + n_{j-1})\right) - \tilde{m}\beta.$$

Now, the righ hand side of the above inequality is nothing else but the carry c computed in the previous iteration. This proves the correctness of the algorithm. □

Example 5. The following table summarizes the steps of computing the square root of 83521 (at base 10):

m	c	$(n_j n_{j-1})_{10}$	$100c + (n_j n_{j-1})_{10}$	d	$d \cdot (20m + d)$
0	0	8	8	2	4
2	4	35	435	8	384
28	51	21	5121	9	5121
289					

Consequently $\sqrt{83521} = 289$.

Remark 4. Although we have presented Algorithm 1.1.37 in context of the integer square root, it can be easily adapted to work with fixed-point rationals. We leave it as an exercise to the reader to modify the algorithm accordingly.

In step (2b) of the algorithm we find the biggest single-digit number d satisfying the inequality $d \cdot (2m\beta + d) \leq c$. For an arbitrary base β we use Algorithm 1.1.36 which is just an exhaustive search starting from some initial guess. However, if we deal with binary numbers, we have only two digits, hence just two possibilities: either $4m + 1 \leq c$ and so $d = 1$ or $4m + 1 > c$ and $d = 0$. This observation let us write a version of Algorithm 1.1.37 optimized for integers in binary representation.

Algorithm 1.1.38. *Given a nonzero integer $n = (n_k \dots n_0)_2$ in binary representation, this algorithm finds a square root of n, when n is a perfect square or raises an error if n is not a square of an integer.*

1. *If necessary, pad n with a leading zero so the number of digits is even.*
2. *Initialize the result $m := 0$, carry $c := 0$ and digit index $j := k$.*
3. *As long as $j > 0$ do:*
   a) *Append next two to digits to the carry setting $c := 4c + (n_j n_{j-1})_2$.*
   b) *If $4m + 1 \leq c$, then set $d := 1$ and $c := c - 4m - 1$, otherwise set $d := 0$ and do not change the carry.*
   c) *Append the computed digit to the result setting $m := 2m + d$.*
   d) *Decrease the index setting $j := j - 2$.*
4. *If the carry is nonzero then n is not a perfect square and we should raise an error.*
5. *Otherwise output m.*

1.2 Floating point numbers

This section needs a **major** update

The exact representation of integers (discussed in the previous section) and rational numbers (that will be discussed in Section 1.4) has the obvious advantage of being—as said—exact. The cost we pay for it is the size of objects that can grow enormously during calculation making some computations totally impractical. Hence, another possibility is to work with approximate numbers. Those however intrinsically suffer from rounding errors. That me must learn how to deal with.

If we are given a real number x and its approximation $\tilde{x}$, then the *absolute error* of this approximation is $|x - \tilde{x}|$. Unfortunately, the absolute error does not tell us much. For example, imagine that we are measuring a size of an animal. Is a 1 cm error acceptable or not? The answer depends on what kind of animals we are dealing with. When gauging the height of an elephant, 1 cm scale would be probably precise enough, but when taking a measure of a louse, it would be ridiculously inexact. Thus,

it is usually more reasonable to take the ratio of the absolute error to the actual value of x.

Definition 1.2.1. The fraction $\frac{|x-\tilde{x}|}{|x|}$ is called the *relative error* of approximating x by $\tilde{x}$.

This notion leads naturally to a so called *scientific notation* and further to floating point numbers. The basic idea goes back to ancient Babylonians. If we have limited space and can write only a certain (fixed) number of digits, it is natural to present x (in fact its approximation) by writing down only few most significant digits times the order of magnitude. This way, separating the order of magnitude from actual digits, we may keep the relative error under control. For instance, the speed of light in vacuum may be expressed as

$$c \approx 2.99\,792 \times 10^8 \ {}^{\mathrm{m}}/_{\mathrm{s}}.$$

with the relative error not exceeding 10^{-5}. Likewise we can write the electron rest mass as

$$m_e \approx 9.10\,938 \times 10^{-31} \text{ kg}$$

and keep the relative error still under 10^{-5}.

Definition 1.2.2. Let $\beta, p > 1$ be two fixed integers. An expression of the form

$$\pm d_0.d_1 \dots d_{p-1} \times \beta^e$$

is called a *floating point number* with p digits: $d_0, \dots, d_{p-1} \in \{0, \dots, \beta - 1\}$ and an exponents $e \in \{e_{\min}, \dots, e_{\max}\}$.

As the notation suggests, $\pm d_0.d_1 \dots d_{p-1} \times \beta^e$ represents a real number

$$\pm\big(d_0 \cdot \beta^e + d_1 \cdot \beta^{e-1} + \dots + d_{p-1} \cdot \beta^{e-p+1}\big).$$

The number β is called the *base* of the floating point number and if we drop the sign and order of magnitude, the number $d_0.d_1 \dots d_{p-1}$ is the *significand*.

Suppose we are approximating a real number x by the nearest floating point number $\tilde{x}$. Assume that x does not exceed the range representable by the given floating point format (i.e. $\beta^{e_{\min}} = 1.0 \dots 0 \times \beta^{e_{\min}} \le |x| \le (\beta-1).(\beta-1)\dots(\beta-1) \times \beta^{e_{\max}} < \beta^{e_{\max}+1}$). Let $\tilde{x} = d_0.d_1 \dots d_{p-1} \times \beta^e$. We want to estimate the relative error of this approximation. As we know, the order of magnitude does not influence the relative error, hence without loss of generality we may assume that x is positive, $x \in [1, \beta)$ and $e = 0$. Since $\tilde{x}$ is the closest possible floating point number, the absolute error is bounded by $0.0 \dots 0({}^{\beta}/_2) = {}^{\beta}/_2 \cdot \beta^{-p}$. It follows that the relative error does not exceed

$$\frac{{}^{\beta}/_2 \cdot \beta^{-p}}{x}.$$

Now, $1 \le x < \beta$ and so the relative error is bounded by

$$\varepsilon := \frac{\beta}{2} \cdot \beta^{-p}.$$

Definition 1.2.3. The number ε derived above is called the *machine epsilon* of the floating point representation in base β with p digits.

Of course, once we fix a base, number of digits and range of allowed exponents, then there are only finitely many numbers that we can exactly represent in a floating point format. For example, $1/10$ is exactly representable as a floating point number 1.00×10^{-1} in base 10 with three digits: 1, 0, 0. On the other hand, it cannot be precisely represented in base 2, no matter how how many digits we use. With just three digits, the best we can get is

$$1.10 \times 2^{-4} = 1 \cdot 2^{-4} + 1 \cdot 2^{-5} = \frac{3}{32}$$

while with 12 digits

$$1.10011001100 \times 2^{-4} = 1 \cdot 2^{-4} + 1 \cdot 2^{-5} + 1 \cdot 2^{-8} + 1 \cdot 2^{-9} + 1 \cdot 2^{-12} + 1 \cdot 2^{-13} = \frac{819}{8192}.$$

The relative error in the first case equals $1/16$ and in the second case $1/4096$.

Even with the fixed base and number of digits, the floating point representation is not unique. In the above example, the number $1/10$ can be represented as either 1.00×10^{-1} or 0.10×10^{0} or 0.01×10^{1}.

Definition 1.2.4. A floating point number $\pm d_0.d_1 \cdots d_{p-1} \times \beta^e$ is said to be *normalized* if $d_0 \neq 0$.

In general we normalize a number (see Algorithm 1.2.5, below) by shifting its digits so that only the first non-zero digit protrudes before the radix point. Unfortunately, we cannot do it with zero since all its digits are null. But the advantages of using normalized floating point numbers are so great, that we cheat a bit and say that the normalized floating point representation of zero is $1.0\ldots0 \times \beta^{e_{\min}-1}$, where $e_{\min}$ is the lowest possible exponent for nonzero numbers.

Example 6. Consider a floating point representation in base 2 with three digits and the range of exponents $\{-1, \ldots, 2\}$. The following table gathers all possible normalized positive numbers:

	1.00	1.01	1.10	1.11
-1	½	⅝	¾	⅞
0	1	5/4	3/2	7/4
1	2	5/2	3	7/2
2	4	5	6	7

The machine epsilon of this representation equals $\varepsilon = 2/2 \cdot 2^{-3} = 1/8$.

As said, the process of normalization of a floating point number boils down to shifting its radix point.

Algorithm 1.2.5. *Assume that parameters β and p, meaning respectively the base and the requested number of digits, are fixed. If $x = d_0 \dots d_k.d_{k+1} \dots d_m \times \beta^e$ is a floating point number which is not normalized, then one can compute its normalized representation as follows:*

1. *if all the digits are null (i.e. $x = 0$), then return $1.0 \dots 0 \times \beta^{e_{\min}-1}$ and quit;*
2. *let l be the index of the first non-zero digit of x;*
3. *shift the radix point, placing it just after d_l and update the exponent setting $e := e + k - l$;*
4. *check if $e_{\min} \leq e \leq e_{\max}$, if not, then throw an exception;*
5. *drop all leading zeros;*
6. *if the number of digits (i.e. $m - l + 1$) is less or equal p then append trailing zeros (if needed) and return $d_l.d_{l+1} \dots d_m 0 \dots 0 \times \beta^e$;*
7. *set $j := p + l$ and $c := 0$;*
8. *as long as $j > l$ repeat the following steps:*
   a) *set $y := c.d_j \dots d_m$ and reset $c := 0$;*
   b) *if $y < 1/2$, then return $d_l.d_{l+1} \dots d_{j-1} 0 \dots 0 \times \beta^e$ and quit;*
   c) *if $y = 1/2$ (i.e. $c = 0$, $d_j = \beta/2$ and $d_{j+1} = \dots = d_m = 0$), then set*

$$d_{j-1} := 2 \cdot \left\lfloor \frac{d_{j-1} + 1}{2} \right\rfloor \qquad \text{and} \qquad d_i := 0 \quad \text{for } i \geq j;$$

   d) *if $y > 1/2$, then set*

$$d_{j-1} := d_{j-1} + 1 \qquad \text{and} \qquad d_i := 0 \quad \text{for } i \geq j;$$

   e) *did any of the two previous steps cause d_{j-1} to become β, then set $d_{j-1} := 0$ and $c := 1$,*
   f) *decrease j by 1;*
9. *return to step 2*

Remark 5. The step 9 is executed only if the algorithm was not terminated inside the loop in step 8. This means that the rounding process pushed the carry c all the way up to the first digit causing an overflow. This can be observed when rounding for example 9.9999 to three digits. We will obtain 10.00 which is not normalized and so we need to shift the decimal point again.

We can now discuss arithmetic of floating point numbers. Although the basic arithmetic is usually available in hardware or in programming libraries (nowadays, the latter are used mostly, when we need higher precision than offered by hardware, see e.g. MPFR library [27]), it is still important to know how floating point numbers are constructed and what kind of challenges and pitfalls can be encountered when dealing with them.

Since round-off errors are built into the very nature of the floating point representation, we shall denote the four basic arithmetic operations by $\oplus$, $\ominus$, $\odot$ and $\oslash$ in order to distinguish them from their exact counterparts. Observe that even if two numbers are exactly representable in a floating point format, we cannot in general expect their sum (respectively difference, product or quotient) to be expressible, too. Take for instance $x = 1 = 1.00 \times 2^0$ and $y = 5/4 = 1.01 \times 2^0$. They are both exactly representable in base 2 with 3 digits (see Example 6), but their sum is not. Thus, given two floating point numbers x, y, we want $x \oplus y$ to be a floating point number $\tilde{z}$ closest to the their exact sum (the same applies of course to subtraction, multiplication and division).

Addition and subtraction of floating point numbers is nearly the same as of integers, except that we have to agree the exponents. If we are doomed to loose some digits, it is obviously better to "sacrifice" digits from the number with a smaller absolute value. Hence, we shift its digits to match the radix point of the other number. Let us consider an example. Take $x = 1.234 \times 10^1$ and $y = 5.678 \times 10^{-1}$. Compute their sum

$$\begin{array}{r} 1.234|00 \times 10^1 \\ +\,0.056|78 \times 10^1 \\ \hline 1.290|78 \times 10^1 \end{array}$$

and so the result is 1.291×10^1 with the relative error $\approx 1.7 \times 10^{-4} < \varepsilon$. Before we formalize the algorithm, let us consider one more example. This time, take $x = 1.234 \times 10^3$ and $y = 5.678 \times 10^{-2}$, then

$$\begin{array}{r} 1.234|00000 \times 10^3 \\ +\,0.000|05678 \times 10^3 \\ \hline 1.234|05678 \times 10^3 \end{array}$$

Hence $x \oplus y = x$, the second argument is to small to have any impact on the result. We might have realized it without any calculations just looking at the difference of the exponents.

Algorithm 1.2.6. *Given two normalized floating point numbers $x = \pm d_0.d_1 \dots d_{p-1} \times \beta^e$ and $y = \pm d'_0.d'_1 \dots d'_{p-1} \times \beta^{e'}$, this algorithm computes their sums $x \oplus y$ (respectively their difference $x \ominus y$) and return it in a normalized form.*

1. *Check whether $e \geq e'$, if it is not then swap the roles of x and y;*
2. *if $e - e' > p$, then return x and quit;*
3. *move the radix point of y by $e - e'$ digits to the left;*
4. *add (respectively subtract) the numbers $x/\beta^e = d_0.d_1 \dots d_{p-1}0 \dots 0$ and $y/\beta^e = 0.0 \dots 0d'_0 \dots d'_{p-1}$ and store the result in $z = \delta_{-1}\delta_0.\delta_1 \dots \delta_{p+e-e'-1}$;*
5. *normalize z using Algorithm 1.2.5 and return the result.*

Remark 6. Addition (subtraction) of the fixed point numbers in step 4 is the same as addition (subtraction) of integers $x \cdot \beta^{p-e'-1} + y \cdot \beta^{p-e'-1}$ and can be computed using Algorithm 1.1.4.

Observe that the algorithm assumes that the intermediate value of z is temporally stored with a greater number of digits than offered by the input/output format. In fact, we require the intermediate value to be stored with up to $2p$ digits. This extra figures are known as *guard digits*. This aspect is important as a negligence here can lead to disastrous errors. Compute for example $1.00 \times 10^1 - 9.99 \times 10^0$. The exact difference is $^1/_{100} = 1.00 \times 10^{-2}$, but if we tried to calculate it without the guard digits, then the result would be

$$\begin{array}{rl} 1.00 & \times 10^1 \\ -0.99\cancel{9} & \times 10^1 \\ \hline 0.01 & \times 10^1 = 1.00 \times 10^{-1}. \end{array}$$

The computed value is 10 times (!) greater than the correct result, the relative error reached 9 (that is nearly $2\,000\varepsilon$).

The multiplication and division of floating point numbers are also very easy.

Algorithm 1.2.7. *Given two normalized floating point numbers $x = \pm d_0.d_1 \dots d_{p-1} \times \beta^e$ and $y = \pm d'_0.d'_1 \dots d'_{p-1} \times \beta^{e'}$, this algorithm computes their product $x \odot y$ and returns it in a normalized form.*

1. *If any of the two numbers is null, then return the (normalized) zero $1.0 \dots 0 \times \beta^{e_{\min}-1}$ and quit;*
2. *compute the exponent of the result setting $\epsilon := e + e'$;*
3. *multiply the fixed point numbers $\pm d_0.d_1 \dots d_{p-1}$ and $\pm d'_0.d'_1 \dots d'_{p-1}$ and store the result in $z = \pm \delta_{-1}\delta_0.\delta_1 \dots \delta_{2p-2}$;*
4. *normalize the number $z \times \beta^\epsilon$ and return the result.*

Remark 7. The fixed point multiplication in step 3 is equivalent to computing a product of two integers, namely

$$\left(d_0 \cdot \beta^{p-1} + d_1 \cdot \beta^{p-2} + \dots + d_{p-1}\right) \cdot \left(d'_0 \cdot \beta^{p-1} + d'_1 \cdot \beta^{p-2} + \dots + d'_{p-1}\right)$$

and placing the radix point so that $2 \cdot (p-1)$ digits stay behind it.

Example 7. Compute the product of $x = 7.43 \times 10^1$ and $y = 2.74 \times 10^3$ in base 10 with 3 digits. We have $\epsilon = 1 + 3 = 4$ and $z = 7.43 \cdot 2.74 = 20.3582$. After normalization the result is 2.04×10^5.

Inverse of a floating point number, division

1.3 Euclidean algorithm

Basic Euclidean algorithm

One of the fundamental operations needed in numerous algorithms is computation of the greatest common divisor of two elements of some ring, be it the ring of integers or the ring of polynomials. If R is a ring in which every element has a unique factorization

into prime (irreducible) elements, the *greatest common divisor* (gcd for short) of any two elements $a, b \in R$ is an element $g \in R$ such that g divides both a, b and if c is another element dividing a, b, then g divides c. Unfortunately, in general, such element is not unique. If g is a greater common divisor of a, b and $u \in R$ is any invertible element (i.e. $uv = 1$ for some $v \in R$), then ug is another gcd. For instance $\gcd(12, -30)$ is either 6 or -6, since -1 is invertible in $\mathbb{Z}$. For the integers this non-uniqueness is not a serious problem. It is a common convention to always select a non-negative gcd. Problems start when we consider other rings, for which there is no canonical way of choosing gcd of two elements. Take for example $a = 13i$ and $b = 4 - 7i$ in a ring $\mathbb{Z}[i] = \{x + yi \,|\, x, y \in \mathbb{Z}\}$. Their greatest common divisor is any of the following fours elements

$$2 + 3i, \quad 3 - 2i, \quad -2 - 3i, \quad -3 + 2i.$$

For this reason, abusing a notation slightly, by $\gcd(a, b)$ we shall understand (depending on the context) either an element of the ring R being the greatest common divisor of a, b or the set of all such elements. In these few cases, when we have to distinguish between these two meanings, we will refer to the former one as to a *representative* of gcd. Consequently, whenever we are dealing with greatest common divisors, the equality relation (unless specified otherwise) is to be understood as an equality of sets. Equivalently, for actual representatives, it is an equality up to a multiplication by an invertible element.

A classical approach is to compute gcd by the *Euclidean algorithm*, which D. Knuth calls "granddaddy of all algorithms" (see [42]). Basically it computes the greatest common divisor of two elements a and b of a ring R replacing $\gcd(a, b)$ by $\gcd(b, r)$, where r is the remainder of a modulo b. Thus, this methods assumes the remainder of a division to be well defined (and computable) in the ring R. In a more formal language, we require R to be an *Euclidean domain*, where this notion is defined as follows.

Definition 1.3.1. An integral ring R is called an *Euclidean domain*, if there is a function (called a *norm*) $\nu : R \to \mathbb{N}_0$ satisfying the following conditions:

- $\nu(a) = 0$ if and only if $a = 0$;
- $\nu(a \cdot b) = \nu(a) \cdot \nu(b)$ for all $a, b \in R$;
- if $a, b \in R$ and $b \neq 0$, then there are unique elements $q, r \in R$ such that

$$a = b \cdot q + r \qquad \text{and} \qquad \nu(r) < \nu(b).$$

The two standard examples of Euclidean domains are: the ring $\mathbb{Z}$ of integers (see Proposition 1.1.13) and the ring $K[x]$ of polynomials over a field (see Proposition 1.1.11). In the former case ν is just the absolute value and in the later case the norm is now given by the formula $\nu(f) := 2^{\deg f}$. These are by far not all the Euclidean domains that exist. Another example is the ring of *Gaussian integers* $\mathbb{Z}[i] = \{x + yi \,|\, x, y \in \mathbb{Z}\}$, here $\nu(x + yi) := x^2 + y^2$. In fact it is that there are infinitely many Euclidean domains.

If we have two elements of an Euclidean domain, then their greatest common divisor can be computed by a recursive procedure.

Proposition 1.3.2. *Let R be an Euclidean domain. If $a, b \in R$, then*

$$\gcd(a,b) = \begin{cases} a, & \text{if } b = 0 \\ \gcd(b,r), & \text{if } b \neq 0 \text{ and } r \text{ is the remainder of } a \text{ modulo } b. \end{cases} \tag{1.11}$$

Proof. The formula holds trivially when $b = 0$. Assume that b is nonzero and let r be the remainder of a modulo b. Hence there is $q \in R$ such that

$$a = b \cdot q + r \qquad \text{and} \quad \nu(r) < \nu(b). \tag{1.12}$$

Now, $\gcd(a,b)$ divides both a and b so it must also divide r. It follows that $\gcd(a,b)$ divides the greatest common divisor of b and r. Conversely, $\gcd(b,r)$ divides the right hand side of (1.12), hence it divides a, too. Every element dividing a and b must divide their greatest common divisor. Thus, $\gcd(b,r)$ divides $\gcd(a,b)$. This proves that they are equal. □

Let us rewrite the previous proposition in form of a pseudo-code.

Algorithm 1.3.3 (Euclidean algorithm)**.** *Let R be an Euclidean domain in which remainders are computable. Given two elements $a, b \in R$, this algorithm computes their greatest common divisor.*

1. *As long as b is non-zero proceed as follows:*
   a) *Compute the remainder $r := a \bmod b$ of a modulo b.*
   b) *Replace a by b and b by r.*
2. *Output a.*

Time complexity analysis. This algorithm works in every Euclidean domain. Yet to analyze its complexity we need to specify the ring R. We show the cost of the algorithm in case when R is a polynomial ring. Hence let $a = f$ and $b = g$ be two polynomials of degree m and n, respectively. The algorithm consists of a single loop and at each iteration the polynomial b is replaced by the remainder a mod b, hence a polynomial of a lower degree. Construct a sequence

$$a_0 := a, \quad a_1 := b, \quad a_2 := \big(a_0 \bmod ba_1\big), \ \ldots, \ a_{i+1} := \big(a_{i-1} \bmod a_i\big), \ \ldots$$

and denote $n_i := \deg a_i$. The sequence of the degrees is strictly decreasing. Consequently, the number if iteration of the loop (denote it k) is bounded from above by the degree of the polynomial b. Each iteration consists of just a single division. Therefore, if we use long multiplication (Algorithm 1.1.12) then the whole gcd computation will use no more than

$$2 \cdot \sum_{i=0}^{k} (n_{i-1} - n_i + 1)(n_i + 1) \tag{1.13}$$

multiplications and up to n inversions of elements of the field K. If we assume in addition that every division reduces the degree by one, then (1.13) simplifies to

$$2 \cdot (m-n+1)(n+1) + 4 \cdot \sum_{i=1}^{n} (n-i+1) = 2 \cdot (mn+m+n+1).$$

We leave it as an exercise for the reader to check that this (degree reduction by one in each iteration) is actually the worst case. Consequently we obtain that the asymptotic complexity of the gcd computation (using long division) equals $O(mn)$, hence is quadratic.

Remark 8. With a similar analysis one can show that for integers the complexity of Algorithm 1.3.3 is quadratic in number of digits.

Remark 9. In $\mathbb{Z}$ it is customary to select the positive representative of the greatest common divisor. Hence implementing Algorithm 1.3.3 for integers one would output $|a|$ rather than a in step (2).

One of the fundamental properties of the greatest common divisor is that it can be expressed as a linear combination of its arguments.

Theorem 1.3.4 (Bézout's identity)**.** *Let R be an Euclidean domain and $a, b \in R$. Then there exist elements $\alpha, \beta \in R$ such that*

$$\gcd(a,b) = \alpha \cdot a + \beta \cdot b. \tag{1.14}$$

Proof. Denote $a_0 := a$ and $a_1 := b$. Further, for $i \geq 1$, let a_{i+1} be the remainder modulo a_i of a_{i-1} and let q_i be the corresponding quotient so we have

$$a_{i-1} = q_i \cdot a_i + a_{i+1}. \tag{1.15}$$

The norms $\nu(a_0), \nu(a_1), \ldots$ of these element form a strictly decreasing sequence non-negative integers. Thus the two sequences are finite. In other words, there is an index k such that a_k divides a_{k-1}. Applying repeatedly (1.11) we have

$$\gcd(a,b) = \gcd(a_0,a_1) = \gcd(a_1,a_2) = \cdots = \gcd(a_{k-1},a_k) = \gcd(a_k,0) = a_k.$$

Now, for every $i > 1$ replace a_i by $a_{i-2} - q_{i-1}a_{i-1}$ to get

$$\begin{aligned}\gcd(a,b) = a_k &= a_{k-2} - q_{k-1} \cdot a_{k-1} \\ &= a_{k-2} - q_{k-1} \cdot (a_{k-3} - q_{k-2} \cdot a_{k-2}) = (-1) \cdot a_{k-3} + (1 + q_{k-2}q_{k-1}) \cdot a_{k-2} \\ &= \cdots = \alpha \cdot a_0 + \beta \cdot a_1.\end{aligned}$$

This shows the assertion. □

The elements α and β in the above theorem are called the *Bézout coefficients*. The following consequence of Bézout's identity is sometimes useful.

Corollary 1.3.5. *Let R be an Euclidean domain with a norm ν and $a, b \in R$. Then a, b have a non-trivial common factor if and only if there are $\alpha, \beta \in R$ such that*

$$\alpha \cdot a + \beta \cdot b = 0 \qquad \text{and} \qquad \nu(\alpha) < \nu(b), \quad \nu(\beta) < \nu(a).$$

Proof. Assume that $g := \gcd(a, b)$ is not a unit. Write $a = a' \cdot g$ and $b = b' \cdot g$ for some $a', b' \in R$. Clearly $\nu(a') = \nu(a) - \nu(g) < \nu(a)$. Likewise $\nu(b') = \nu(b) - \nu(a) < \nu(b)$. It suffices to take $\alpha := b'$ and $\beta := -a'$.

Conversely, suppose that such two elements α, β exist and yet $\gcd(a, b) = 1$. Bźeout's identity asserts that there are $\alpha', \beta' \in R$ such that

$$1 = \alpha' a + \beta' b.$$

Multiply both sides by α and use the identity $\alpha a + \beta b = 0$ to get

$$\alpha = \alpha\alpha' a + \alpha\beta' b = -\beta\alpha' b + \alpha\beta' b = b \cdot (\alpha\beta' - \beta\alpha').$$

This implies that the norm of α is greater or equal the norm of b, which contradicts our assumption. □

The Bézout coefficients can be computed by the so called *extended* Euclidean algorithm, which is nothing else but a formalization of the proof of Theorem 1.3.4 presented above.

Algorithm 1.3.6 (Extended Euclidean algorithm)**.** *Let R be an Euclidean domain in which remainders are computable. Given two elements $a, b \in R$, this algorithm computes their greatest common divisor together with Bézout coefficients α, β.*

1. *Initialize $\alpha := 1$, $\alpha' := 0$ and $\beta := 0$, $\beta' := 1$.*
2. *As long as $b \neq 0$ proceed as follows:*
   a) *Compute the quotient q and remainder r of the division of a by b.*
   b) *Replace a and b with b and r, respectively.*
   c) *Replace α and α' with α' and $\alpha - q \cdot \alpha'$, respectively.*
   d) *Replace β and β' with β' and $\beta - q \cdot \beta'$, respectively.*
3. *Output the greatest common divisor $\gcd(a, b) = a$ and the Bézout's coefficients α, β.*

Example 8. Take two integers $m = 3256$ and $n = 2331$. The following table gathers all the steps of the extended Euclidean algorithm

m	n	q	r	α	α'	β	β'
3256	2331	1	925	1	0	0	1
2331	925	2	481	0	1	1	−1
925	481	1	444	1	−2	−1	3
481	444	1	37	−2	3	3	−4
444	37	12	0	3	−5	−4	7
37	**0**			**−5**		**7**	

It follows that $\gcd(m, n) = 37$ and the corresponding Bézout coefficients are $\alpha = -5$ and $\beta = 7$.

Polynomial gcd via pseudo-division

One problem with the basic Euclidean algorithm, described in the previous section, is that it works only in Euclidean domains like $\mathbb{Z}$ or $K[x]$. (Another problem is that in some domains, like $\mathbb{Q}[x]$, it suffers from explosive growth of coefficients in intermediate steps. We will return to this issue later.) On the other hand, the greatest common divisors exist in a much wider class of rings. In practice we often need to compute greatest common divisors of multivariate polynomials or polynomials over $\mathbb{Z}$, where neither $K[x_1,\ldots,x_n]$ nor $\mathbb{Z}[x]$ is an Euclidean domain. Thus, our next aim is to adapt the Euclidean algorithm to work in these rings, by using pseudo-division in place of division. To this end we need to introduce a notion of *primitive* polynomials. Recall from algebra course that a *unique factorization domain* (ufd in short) is is an integral are ring such that element of it can be uniquely (up to a multiplication by a unit) written as a product of primes. All the rings mentioned above are ufds.

Definition 1.3.7. Let R be a unique factorization domain and $f \in R[x]$ be a nonzero polynomial. The greatest common divisor of the coefficients of f is called the *content* of f and denoted $\mathrm{cont}(f)$.

Writing the definition symbolically, if $f = f_n x^n + \cdots + f_1 x + f_0$, then $\mathrm{cont}(f) = \gcd(f_n,\ldots,f_0)$.

Definition 1.3.8. We say that the polynomial is *primitive*, when its coefficients are co-prime.

In other words, $f \in R[x]$ is primitive if and only if $\mathrm{cont}(f)$ is invertible in R. Let $f = f_n x^n + \cdots + f_1 x + f_0$ and $cont(f) = c$. Denote $g_i := {}^{f_i}/_{c}$ and $g := g_n x^n + \cdots + g_1 x + g_0$. Observe that $g_i \in R$ for every index i (since $c = \gcd(f_n,\ldots,f_0)$) and $g_0,\ldots,g_n$ are relatively prime. Consequently the polynomial g is primitive. This wat we have proved:

Observation 1.3.9. *For every nonzero polynomial f, with coefficients in a unique factorization domain, there is a unique primitive polynomial g such that*

$$f = \mathrm{cont}(f) \cdot g.$$

The polynomial g is called the *primitive part* of f and denoted $\mathrm{pp}(f)$. For the sake of completeness, we define also $\mathrm{cont}(0) := 1$ and $\mathrm{pp}(0) := 0$. One of the fundamental facts concerning primitive polynomials is known as *Gauss*[5] *lemma*.

Theorem 1.3.10 (Gauss lemma)**.** *If R is a unique factorization domain and $f, g \in R[x]$ are two primitive polynomials, then their product $f \cdot g$ is also a primitive polynomial.*

Proof. Let $f = f_m x^m + \cdots + f_1 x + f_0$ and $g = g_n x^n + \cdots + g_1 x + g_0$. Fix any prime element $p \in R$. Since f, g are primitive, not all the coefficients are divisible by p. Let j be the minimal index such that p does not divide f_j and k be the minimal index such

[5] Johann Carl Friedrich Gauß (born 30 April 1777, died 23 February 1855) was a German mathematician.

that p does not divide g_k. Consider the coefficient h_{j+k} of the product $h = f\,g$. The formula (1.1) for the coefficients of the product yields

$$h_{j+k} = \cdots + f_{j-1}g_{k+1} + f_j g_k + f_{j+1}g_{k-1} + \cdots .$$

It follows that p divides all the summands except $f_j g_k$ so it does not divide h_{j+k}. We conclude that h is primitive. □

One consequence of the Gauss lemma is that factorization of polynomials is basically the same over a unique factorization domain and its field of fractions. In particular gcd of two primitive polynomials is itself primitive. More precisely:

Corollary 1.3.11. *If R is a unique factorization domain and $f, g \in R[x]$, then*

$$\gcd(f,g) = \gcd\big(\operatorname{cont}(f), \operatorname{cont}(g)\big) \cdot \gcd\big(\operatorname{pp}(f), \operatorname{pp}(g)\big).$$

As always, the equality in the above formula holds up to a multiplication by a unit. Two polynomials $f, g \in R[x]$ are said to be *similar* (denoted $f \sim g$) if there are nonzero elements $\alpha, \beta \in R$ such that $\alpha \cdot f = \beta \cdot g$. Similarity is clearly an equivalence relation. Moreover we have:

Observation 1.3.12. *Two polynomials f, g are similar if and only if their primitive parts are equal.*

The next lemma shows that this relation is compatible with the pseudo-division.

Lemma 1.3.13. *Let $f_0, f_1, g_0, g_1 \in R[x]$. Assume that r_i is a pseudo-remainder of f_i modulo g_i for $i \in \{0,1\}$. If $f_0 \sim f_1$ and $g_0 \sim g_1$, then also $r_0 \sim r_1$.*

Proof. Denote the degrees of the polynomials by $m := \deg f_0 = \deg f_1$ and $n := \deg g_0 = \deg g_1$. If $m < n$ then $r_0 = f_0$ and $r_1 = f_1$. Consequently the pseudo-reminders are trivially similar by assumptions of the lemma. Thus, without loss of generality, we may assume that $m \geq n$. Let $d := m - n + 1$. There are unique polynomials q_0, q_1 such that

$$\operatorname{lc}(g_i)^d \cdot f_i = q_i \cdot g_1 + r_i, \qquad \text{for } i = 0, 1. \tag{1.16}$$

Since f_0 is similar to f_1 and g_0 is similar to g_1, there are nonzero elements $\alpha_0, \alpha_1, \beta_0, \beta_1$ such that

$$\alpha_0 \cdot f_0 = \alpha_1 \cdot f_1 \qquad \text{and} \qquad \beta_0 \cdot g_0 = \beta_1 \cdot g_1.$$

In particular $\beta_0 \cdot \operatorname{lc}(g_0) = \beta_1 \cdot \operatorname{lc}(g_1)$. It follows from (1.16) that

$$\begin{aligned}
\big(\beta_0 \cdot \operatorname{lc}(g_0)\big)^d \cdot \alpha_0 \cdot f_0 &= \big(\alpha_0 \cdot \beta_0^{d-1} \cdot q_0\big) \cdot \big(\beta_0 \cdot g_0\big) + \alpha_0 \cdot \beta_0^d \cdot r_0 \\
\big(\beta_1 \cdot \operatorname{lc}(g_1)\big)^d \cdot \alpha_1 \cdot f_1 &= \big(\alpha_1 \cdot \beta_1^{d-1} \cdot q_1\big) \cdot \big(\beta_1 \cdot g_1\big) + \alpha_1 \cdot \beta_1^d \cdot r_1 \\
&= \big(\alpha_1 \cdot \beta_1^{d-1} \cdot q_1\big) \cdot \big(\beta_0 \cdot g_0\big) + \alpha_1 \cdot \beta_1^d \cdot r_1.
\end{aligned}$$

Now, the left-hand-sides are equal and so must be the right-hand-sides. Therefore, both $\alpha_0\beta_0^d r_0$ and $\alpha_1\beta_1^d r_1$ are pseudo-remainders of $\alpha_0 f_0$ modulo $\beta_0 g_0$. But the pseudo-remainder is unique, which means that $\alpha_0\beta_0^d r_0 = \alpha_1\beta_1^d r_1$. □

We will now develop a method for computing the greatest common divisor of two polynomials that uses pseudo-division. Fix two polynomials f and g with coefficients in some unique factorization domain R, Define a (finite) sequence $(f_0, \dots, f_k)$ of polynomials as follows. Set $f_0 := f$ and $f_1 := g$. Next, compute the pseudo-remainder of f_0 modulo f_1 and denote it f_2. Continue this process setting $f_{i+1} := (f_i \bmod f_{i-1})$ till we reach zero. This means that we have

$$\begin{aligned} \operatorname{lc}(f_1)^{d_1} \cdot f_0 &= f_1 \cdot q_1 + f_2, \qquad && \deg f_2 < \deg f_1. \\ \operatorname{lc}(f_2)^{d_2} \cdot f_1 &= f_2 \cdot q_2 + f_3, && \deg f_3 < \deg f_2 \\ &\vdots \\ \operatorname{lc}(f_k)^{d_k} \cdot f_{k-1} &= f_k \cdot q_k + 0, \end{aligned}$$

for some polynomials (pseudo-quotients) $q_1, \dots, q_k$ and integers $d_1, \dots, d_k$. Observe that the corresponding sequence $\big(\operatorname{pp}(f_0), \dots \operatorname{pp}(f_k)\big)$ of the primitive parts of the polynomials $f_0, \dots, f_k$ will stay unchanged even if we replace a polynomial f_i by a polynomial similar to it at a certain step. As it is evident from Corollary 1.3.11, these are the primitive parts that matter when it comes to computing gcd. This facts justify the following definition.

Definition 1.3.14. Let $f, g \in R[x]$ be two polynomials with coefficients in a unique factorization domain R. A *polynomial pseudo-remainder sequence* (prs for short) of f and g is a sequence $(f_0, f_1, \dots, f_k, f_{k+1})$ of polynomials, where $f_0 = f$, $f_1 = g$, $f_{k+1} = 0$ and for every $i \geq 1$ the polynomial f_{i+1} is similar to the pseudo-remainder of f_{i-1} modulo f_i.

We are now in a position to present the main result of this point.

Theorem 1.3.15. *If $(f_0, f_1, \dots, f_k, 0)$ is a pseudo-remainder sequence of two polynomials f, g with coefficients in some unique factorization domain R, then*

$$\gcd(f, g) = \gcd\big(\operatorname{cont}(f), \operatorname{cont}(g)\big) \cdot \operatorname{pp}(f_k).$$

Proof. Corollary 1.3.11 assert that

$$\gcd(f, g) = \gcd\big(\operatorname{cont}(f), \operatorname{cont}(g)\big) \cdot \gcd\big(\operatorname{pp}(f_0, \operatorname{pp}(g)\big).$$

Therefore it suffices to show that $\gcd\big(\operatorname{pp}(f), \operatorname{pp}(g)\big)$ equals $\operatorname{pp}(f_k)$. Without loss of generality we may assume that f and g are primitive. Denote their greatest common divisor by h. Directly by definition h divides the pseudo-remainder of f_0 modulo f_1, which is similar to f_2. Inductively, we show that h divides all the polynomials f_j for $j \geq 0$ so in particular h divides f_k. And since gcd of two primitive polynomials is primitive, h divides $\operatorname{pp}(f_k)$.

On the other hand, $\operatorname{pp}(f_k)$ divides f_{k-1}, because the next element in the sequence is zero. Again by a simple induction, $\operatorname{pp}(f_k)$ divides all f_i for $i = k-1, \dots, 0$. It

follows that, $\operatorname{pp}(f_k)$ divides both $f = f_0$ and $g = f_1$. Consequently, $\operatorname{pp}(f_k)$ divides $h = \gcd(f, g)$. This shows that h and $\gcd(f, g)$ are equal (up to a multiplication by a unit). □

This theorem gives raise to a variant of the Euclidean algorithm presented below. It differs from the basic Euclidean algorithm in that it uses pseudo-division in place of division. Consequently it is suitable for polynomials over unique factorization domains.

Algorithm 1.3.16. *Let R be a unique factorization domain in which* gcd *is computable. Given two nonzero polynomials $f, g \in R[x]$, this algorithm computes their greatest common divisor.*

1. *Let $d := \gcd\big(\operatorname{cont}(f), \operatorname{cont}(g)\big)$ be the greatest common divisor of the contents of f and g.*
2. *Set a and b to be the primitive parts of f and g, respectively.*
3. *As long as b is non-zero do:*
   a) *Compute the pseudo-remainder r of $a \bmod b$.*
   b*) *Replace r by a suitable polynomial r^* similar to r.*
   c) *Replace a and b by b and r, respectively.*
4. *Output the result $\gcd(f, g) = d \cdot \operatorname{pp}(a)$.*

The correctness of the algorithm follows from the theorem preceding it. Still, there is step (3b*) that needs some clarification. Here we replace the pseudo-remainder r by a similar but somehow "simpler" polynomial r^*. The efficiency of the whole procedure strongly depends on this step.

A naïve approach is omit this step altogether, leaving the pseudo-remainder r unchanged. As simple as this approach is, it suffers from an explosive growth of the size of the coefficients in intermediate steps. For polynomials over $\mathbb{Z}$, the size of coefficients in intermediate steps grows exponentially with each step (for a detailed analysis see [42, §4.6.1]). We may observe this phenomenon in the following example:

Example 9. Take two polynomials over the integers:

$$f = x^6 + 2x^5 - 4x^4 + 11x^2 - 9x + 2,$$
$$g = 7x^5 - 11x^4 - 18x^3 + 48x^2 - 29x + 6.$$

Contents of the both polynomials equal 1. Compute the greatest common divisor using Algorithm 1.3.16 omitting step (3b*), i.e. without altering the pseudo-remainders. The generated pseudo-remainder sequence has a form:

$$f_2 = 205x^4 + 114x^3 - 458x^2 + 242x - 52,$$
$$f_3 = 248822x^3 + 271656x^2 - 405279x + 93394,$$
$$f_4 = -260529028550x^2 - 854841955800x - 667567797400,$$
$$f_5 = 505432618757499534152860025 00x + 10108652375149990683057 2005000,$$
$$f_6 = 0.$$

Therefore, the greatest common divisor of f and g equals

$$\gcd(f,g) = \mathrm{pp}(f_5) = x+2.$$

Notice the enormous growth of the coefficients of the intermediate steps, even when the input and output polynomials have really small coefficients.

The other extreme is to reduce the sizes of the elements as much as possible, substituting the primitive part for each pseudo-remainder. Such a pseudo-remainder sequence, in which every element is primitive, is called a *primitive pseudo-remainder sequence* (pprs for short).

Example 10. Take the same two polynomials as in the previous example. Compute their gcd substituting the primitive part for every pseudo-remainder. This time the generated pseudo-remainder sequence consists of:

$$\begin{aligned} f_2 &= 205x^4 + 114x^3 - 458x^2 + 242x - 52, \\ f_3 &= 5078x^3 + 5544x^2 - 8271x + 1906, \\ f_4 &= -1291x^2 - 4236x - 3308, \\ f_5 &= x+2, \\ f_6 &= 0. \end{aligned}$$

Again $\gcd(f,g) = \mathrm{pp}(x+2) = x+2$. This time the coefficient are evidently much shorter. In fact they are as short as possible.

Unfortunately, the costs of "pruning" the coefficients substituting $\mathrm{pp}(r)$ for r every time, can easily become extremely high. At each step we have to compute the content of the intermediate polynomial, hence the gcd of its coefficients. Imagine that we are working with multivariate polynomials viewed as univariate polynomials in one variable over a multivariate polynomial ring in the remaining variables. Computing the greatest common divisor of two polynomials leads to finding lots of gcds of coefficients which are polynomials in one less variable. This in turn results in computing even more gcds of polynomials in yet one less variable and so on. It does not need many variables for this procedure to become prohibitively time consuming.

Our aim is to keep the coefficients reasonably short without computing the greatest common divisors of the coefficients at each step. We can achieve it dividing the pseudo-remainder r_i by a certain scalar $\alpha_i \in R$ which is **a priori** known to divide the content of r_i. The following solution is due to G.E. Collins and the resulting prs is called the *subresultant pseudo-remainder sequence*. Let r_i be the pseudo-remainder of f_{i-1} modulo f_i. Define f_{i+1} to be

$$f_{i+1} := \frac{(-1)^{\delta_{i-1}+1}}{\mathrm{lc}(f_{i-1}) \cdot \alpha_{i-1}^{\delta_{i-1}}} \cdot r_i, \tag{1.17}$$

where $\delta_{i-1} = \deg f_{i-1} - \deg f_i$ and α_i is defined recursively by the formula:

$$\alpha_1 := 1 \qquad \text{and} \qquad \alpha_i := \mathrm{lc}(f_i)^{\delta_{i-1}} \cdot \alpha_{i-1}^{1-\delta_{i-1}}.$$

missing

It can be proved that the division in (1.17) is performed in the ring R. However, the proof of this fact needs to be postponed till Chapter 8 (see), where we will have necessary tools at our disposal. In particular, we will discover that the polynomials $f_2, \ldots, f_k$ can be interpreted as determinants of certain matrices. Considering its importance it is worth to explicitly present a variant of Algorithm 1.3.16 that uses the subresultant pseudo-remainder sequence.

Algorithm 1.3.17. *Let $f, g \in R[x]$ be two polynomials with coefficients in a unique factorization domain R in which* gcd *is computable, then the following algorithm computes the greatest common divisor of f and g.*

1. *Let d be the greatest common divisor of the contents of f and g, i.e. $d := \gcd\big(\mathrm{cont}(f), \mathrm{cont}(g)\big)$.*
2. *Set a and b to be the primitive parts of f and g, respectively.*
3. *Initialize $\alpha := 1$.*
4. *As long as b is non-zero proceed as follows:*
   a) *Compute the pseudo-remainder r of $a \bmod b$ and set $\delta := \deg a - \deg b$.*
   b) *Replace a and b with b and $\frac{(-1)^{\delta+1} \cdot r}{\mathrm{lc}(a) \cdot \alpha^{\delta}}$, respectively.*
   c) *Update α setting $\alpha := \alpha^{1-\delta} \cdot \mathrm{lc}(a)^{\delta}$ (notice that a has already been substituted).*
5. *Return the product $d \cdot \mathrm{pp}(a)$.*

Remark 10. Notice that in this algorithm the contents of the polynomials (i.e. the greatest common divisors of the coefficients) need to be computed only in the first and the last steps, hence outside the loop.

Example 11. Consider once again the same two polynomials that we have encountered in the last two examples:

$$\begin{aligned} f &= x^6 + 2x^5 - 4x^4 + 11x^2 - 9x + 2, \\ g &= 7x^5 - 11x^4 - 18x^3 + 48x^2 - 29x + 6. \end{aligned}$$

Find $\gcd(f, g)$ using the above algorithm. This time the prs computed by the algorithm consists of the following polynomials:

$$\begin{aligned} f_2 &= 205x^4 + 114x^3 - 458x^2 + 242x - 52, \\ f_3 &= 5078x^3 + 5544x^2 - 8271x + 1906, \\ f_4 &= -2582x^2 - 8472x - 6616, \\ f_5 &= 3929x + 7858, \\ f_6 &= 0. \end{aligned}$$

Of course the result is the same since $\mathrm{pp}(3929x + 7858) = x + 2$. Observe that the coefficients of the intermediate polynomials remain relatively short, although not as short as in the previous example, yet still much shorter than in Example 9.

Exercise 4. The following two polynomials with integer coefficients appear in the context of pseudo-remainder sequences in [13, 24, 42, 91, 94]:

$$f = x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5,$$
$$g = 3x^6 + 5x^4 - 4x^2 - 9x + 21.$$

Therefore, they can be considered to be a de facto standard example of coefficients growth phenomenon. Compute their greatest common divisor using all three methods described above. Compare the sizes of coefficients of intermediate polynomials.

Binary and x-ary gcd algorithms

We will now present a variant of Euclidean algorithm, due to J. Stein, that uses only division by 2. The algorithm is based on the following straightforward observations:

Observation 1.3.18. *If m, n are two even integers, then*

$$\gcd(m,n) = 2 \cdot \gcd\Big(\frac{m}{2}, \frac{n}{2}\Big).$$

Observation 1.3.19. *If m is even and n is odd, then*

$$\gcd(m,n) = \gcd\Big(\frac{m}{2}, n\Big).$$

Observation 1.3.20. *If $m < n$ are two positive odd integers, then their difference is even and*

$$\gcd(m,n) = \gcd\Big(\frac{m-n}{2}, m\Big).$$

Let n be a positive integer represented at base 2 as $n \sim (n_k n_{k-1} \dots n_0)_2$. Denote $T_0(n) := \max\{j \mid n_0 = n_1 = \dots = n_j = 0\}$ the number of its trailing zeros. This number can be computed either by a simple loop or, in some architectures, it can be a build-in constant-time operation (e.g. `TZCNT` instruction on x86 processors).

Algorithm 1.3.21 (Stein, 1967)**.** *Given two non-negative integers $m, n \in \mathbb{Z}$ this algorithm returns their greatest common divisor.*

1. *If $m = 0$, then return n;*
2. *likewise, if $n = 0$, then return m;*
3. *set $t := \min\big\{T_0(m), T_0(n)\big\}$;*
4. *as long as m is nonzero repeat the following steps:*
   a) *prune the trailing zero of m and n setting:*

$$m := \frac{m}{2^{T_0(m)}}, \qquad n := \frac{n}{2^{T_0(n)}}$$

   b) *if $m > n$, then swap the roles of m, n;*

c) replace n by m and m by $(n-m)/2$.

5. *return* $2^t \cdot n$.

Recall that the greatest common divisor is a symmetric function, that is $\gcd(m,n) = \gcd(n,m)$. Hence Observations 1.3.18–1.3.20 cover all possible cases. Moreover, in all three cases the 1-norms of arguments to gcd are diminished. Thus, for every pair of integers the algorithm will terminate. This proves the correctness of Stein algorithm. The asymptotic complexity of this algorithm is the same as of the 'ordinary' Euclidean algorithm. The main advantage of this algorithm is that it uses only division and multiplication by powers of 2, which on a binary computer is likely to be much more efficient than an arbitrary division with a reminder. Like for human beings working with base 10 integers, it is much easier to divide by and multiply by powers of 10. It all boils down to appending/removing trailing zeros. On many architectures, and in some programming languages, there exist dedicated instructions for shifting bits (e.g. << and >> in C/C++) that can be used for this purpose.

Example 12. Let us compute the greatest common divisor of $m = 3816$ and $n = 26748$. In binary format $m \sim (111011101000)_2$ and $n \sim (110100001111100)_2$, hence $T_0(m) = 3$, $T_0(n) = 2$ and so $t = 2$. The following table gathers all the steps of computing $\gcd(m,n)$ with Algorithm 1.3.21.

	m	n
	3816	26748
$m \leftarrow \frac{m}{8}$	477	26748
$n \leftarrow \frac{n}{4}$	477	6687
$m \leftarrow \frac{n-m}{2}, n \leftarrow m$	3105	477
$m \leftrightarrow n$	477	3105
$m \leftarrow \frac{n-m}{2}, n \leftarrow m$	1314	477
$m \leftarrow \frac{m}{2}$	657	477
$m \leftrightarrow n$	477	657
$m \leftarrow \frac{n-m}{2}, n \leftarrow m$	90	477
$m \leftarrow \frac{m}{2}$	45	477
$m \leftarrow \frac{n-m}{2}, n \leftarrow m$	216	45
$m \leftarrow \frac{m}{8}$	27	45
$m \leftarrow \frac{n-m}{2}, n \leftarrow m$	9	27
$m \leftarrow \frac{n-m}{2}, n \leftarrow m$	9	9
$m \leftarrow \frac{n-m}{2}, n \leftarrow m$	0	9

It follows that $\gcd(3816, 26748) = 2^2 \cdot 9 = 36$.

It is easy to devise a polynomial counterpart of Stein algorithm. The only difference is that, in order to kill the constant coefficient, we canot just subtract polynomial, as we with integers (unless we are working with polynomials over a binary field $\mathbb{F}_2$). Instead we take a linear combination of the two polynomials.

Observation 1.3.22. *If f, g are two polynomials (with coefficients in a unique factorization domain), then*

$$\gcd(f,g) = \gcd\Big(\frac{f\cdot g_0 - g\cdot f_0}{x}, f\Big).$$

Similarly as we did for integers, for a nonzero polynomial $f = f_n x^n + \cdots + f_1 x + f_0$ we denote $T_0(f) := \max\{j \mid f_0 = f_1 = \cdots = f_j = 0\}$. The polynomial analog of Algorithm 1.3.21 take a form:

Algorithm 1.3.23 (x-ary gcd)**.** *Given two polynomials f, g with coefficients in a field K, this algorithm returns their greatest common divisor.*

1. *If $f = 0$, then return g;*
2. *if $g = 0$, then return f;*
3. *set $t := \min\{T_0(f), T_0(g)\}$;*
4. *as long as f is nonzero repeat the following steps:*
   a) *prune the trailing zeros of both polynomials setting:*
   $$f := \frac{f}{x^{T_0(f)}}, \qquad g := \frac{g}{x^{T_0(g)}}$$
   b) *if $\deg f > \deg g$, then swap the roles of f and g;*
   c) *replace g by f and f by $\frac{f\cdot g_0 - g\cdot f_0}{x}$.*
5. *return $x^t \cdot g$.*

Remark 11. We stated Algorithm 1.3.23 for polynomials over a field, but one should observe that it does not involve division with remainder. Consequently it is not limited to polynomials over fields but can be easily adapted to work with polynomials over unique factorization domains. In such a setting one needs to apply an analog of Theorem 1.3.15. We let the reader work out the details.

Remark 12. For polynomials over $\mathbb{Q}$ (or $\mathbb{Z}$) Algorithm 1.3.23 suffers from an explosive growth of coefficients even more severe than observed in Example 9 of the previous section. On the other hand, the algorithm excels when applied to polynomials over finite fields. This makes it a good companion for a modular gcd algorithm described in Section2.5 (see Algorithm 2.5.2).

Fast gcd algorithm

So far all algorithms presented in this section had quadratic asymptotic complexity. Now we are going to present an algorithm with substantially better asymptotic behavior. For clarity of exposition we concentrate on the polynomial case alone, but the reader should be informed that there is a version of this algorithm that works with integers. In fact the latter one was developed first. For further details see the bibliographic notesat the end of this section. missing

First we introduce some notation. Let $f, g \in K[x]$ be two polynomials with coefficients in a field and such that $g \neq 0$. Further let $f_0 = f$, $f_1 = g$, $f_2 = (f \bmod g)$,..., $f_{i+1} = (f_{i-1} \bmod f_i)$, ..., $f_k = (f_{k-2} \bmod f_{k-1})$, $f_{k+1} = 0$ be the sequence of polynomials constructed by successive steps of the basic Euclidean algorithm (Algorithm 1.3.3) and let $q_i := (f_{i-1} \operatorname{div} f_i)$ be the corresponding quotients for $i = 0, \ldots, k$. Therefore we have

$$\begin{aligned} f_0 &= f_1 \cdot q_1 + f_2, \\ f_1 &= f_2 \cdot q_2 + f_3, \\ &\vdots \\ f_{k-1} &= f_k \cdot q_k + 0. \end{aligned}$$

Set $Q_i := \left(\begin{smallmatrix} q_i & 1 \\ 1 & 0 \end{smallmatrix}\right)$. We may rewrite the above equalities in a matrix form

$$\begin{pmatrix} f_{i-1} \\ f_i \end{pmatrix} = Q_i \cdot \begin{pmatrix} f_i \\ f_{i+1} \end{pmatrix} \qquad \text{for} \quad i \in \{1, \ldots, k\}.$$

Consequently $\left(\begin{smallmatrix} f_0 \\ f_1 \end{smallmatrix}\right) = Q_1 Q_2 \cdots Q_i \cdot \left(\begin{smallmatrix} f_i \\ f_{i+1} \end{smallmatrix}\right)$ for every $i \leq k$. A matrix of a form $R_i := Q_1 \cdots Q_i$ will be called *regular*. For the sake of completeness we define R_0 to be the identity matrix and call it regular, too.

Observation 1.3.24. $Q_i^{-1} = \left(\begin{smallmatrix} 0 & 1 \\ 1 & -q_i \end{smallmatrix}\right)$.

Observation 1.3.25. $\det R_i = (-1)^i$.

Lemma 1.3.26. *If $R_i = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$ is regular and distinct form the identity matrix, then*

$$\deg a \geq \max\{\deg b, \deg c\} \geq \deg d.$$

Proof. We proceed by induction on i. The assertion is trivial for $R_1 = \left(\begin{smallmatrix} q_1 & 1 \\ 1 & 0 \end{smallmatrix}\right)$. Assume that the lemma is true for $R_{i-1} = \left(\begin{smallmatrix} a' & b' \\ c' & d' \end{smallmatrix}\right)$. Then

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} a' & b' \\ c' & d' \end{pmatrix} \cdot \begin{pmatrix} q_i & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} a'q_i + b' & a' \\ c'q_i + d' & c' \end{pmatrix}.$$

It is clear that $\deg(a'q_i + b') = \deg a' + \deg q_i \geq \deg a'$. Likewise

$$\deg(a'q_i + b') = \deg a' + \deg q_i \geq \deg c' + \deg q' = \deg(c'q_i + d').$$

Finally $\deg(c'q_i + d') = \deg c' + \deg q_i \geq \deg c'$. □

Lemma 1.3.27. *Let $f, g \in K[x]$ be two polynomials and $f_0, \ldots, f_k$ be defined as before. Finally let $a, b \in K[x]$ be two further polynomials. Then the following conditions are equivalent:*

1. *There is an index i such that $a = f_i$ and $b = f_{i+1}$.*
2. *There is a regular matrix R_i such that $\binom{f}{g} = R_i \cdot \binom{a}{b}$.*

Proof. The implication (1)$\Longrightarrow$(2) is trivial. Conversely assume (2). Observation 1.3.25 implies that R_i is invertible, hence

$$\begin{pmatrix} a \\ b \end{pmatrix} = R_i^{-1} \cdot \begin{pmatrix} f \\ g \end{pmatrix} = Q_i^{-1} \cdots Q_1^{-1} \cdot \begin{pmatrix} f_0 \\ f_1 \end{pmatrix} = \begin{pmatrix} f_i \\ f_{i+1} \end{pmatrix}.$$

This shows that (1) holds. □

The main idea behind the fast gcd algorithm is to observe that the quotient of two polynomials can be completely computed knowing only a few of their leading coefficients. All the remaining coefficients are necessary to compute nomen omen the remainder, but not the quotient. Let us show it on an example Take

$$f = 6x^7 - 8x^6 + 5x^5 - 4x^3 + 2x^2 + 4 \qquad \text{and} \qquad g = 2x^5 - 3x^3 + 4x^2 + 5x - 1$$

and compute the quotient $f \operatorname{div} g$ using long division. Below we replace the coefficients we do not use by empty squares:

$$\begin{array}{r|l} & 3x^2 - 4x \quad +1 \\ \hline 2x^5 - 3x^3 + \square x^2 + \square x + \square & 6x^7 - 8x^6 \quad + 5x^5 + \square x^4 + \square x^3 + \square x^2 + \square \\ & \underline{6x^7 \qquad - 9x^5 + \square x^4 + \square x^3 + \square x^2 + \square} \\ & \qquad - 8x^6 + 14x^5 + \square x^4 + \square x^3 + \square x^2 + \square \\ & \qquad \underline{- 8x^6 + 12x^5 + \square x^4 + \square x^3 + \square x^2 + \square} \\ & \qquad\qquad 2x^5 + \square x^4 + \square x^3 + \square x^2 + \square \\ & \qquad\qquad \underline{2x^5 + \square x^4 + \square x^3 + \square x^2 + \square} \\ & \qquad\qquad\qquad \square x^4 + \square x^3 + \square x^2 + \square \end{array}$$

We can now express this phenomenon more formally.

missing

We are now in a position to present the main ingredient of the fast gcd computation. The following procedure is often called *half-gcd* algorithm.

Algorithm 1.3.28 (Half-gcd). *Given two polynomials $f = f_m x^m + \cdots + f_1 x$ and $g = g_n x^n + \cdots + g_1 x$ with coefficients in a field K such that $\deg f > \deg g \geq 0$, this algorithm returns a regular matrix R_i such that the polynomials $\binom{f_i}{f_{i+1}} = R^{-1} \cdot \binom{f}{g}$ satisfy the condition*

$$\deg f_i \geq \left\lceil \frac{\deg f}{2} \right\rceil > \deg f_{i+1}. \tag{1.18}$$

1. *Set the threshold $t := \lceil \deg f / 2 \rceil$.*
2. *If $\deg g < t$, then output the identity matrix and terminate.*

3. *Take the "upper parts" of f and g:*

$$f_u = f \operatorname{div} x^t = f_m x^{m-t} + \cdots + f_{t+1}x + f_t$$
$$g_u = g \operatorname{div} x^t = g_n x^{n-t} + \cdots + g_{t+1} + g_t.$$

4. *Call this algorithm recursively with input f_u, g_u. Denote the result by Q.*
5. *Set $\left(\begin{smallmatrix} F \\ G \end{smallmatrix}\right) := Q^{-1} \cdot \left(\begin{smallmatrix} f \\ g \end{smallmatrix}\right)$.*
6. *If $\deg G < t$, then output Q and terminate.*
7. *Compute the quotient q and remainder r of F modulo F.*
8. *Denote $l := \deg G$ and $k := 2t - l$.*
9. *Take the "upper parts" of G and r setting*

$$G_u := G \operatorname{div} x^k \qquad \text{and} \qquad r_u := r \operatorname{div} x^k.$$

10. *Call this algorithm recursively with input G_u, r_u. Denote the result by S.*
11. *Output $R = Q \cdot \left(\begin{smallmatrix} q & 1 \\ 1 & 0 \end{smallmatrix}\right) \cdot S$.*

Before we prove the correctness of the algorithm, we need to present a rather technical lemma.

Lemma 1.3.29. *Let f, g be two polynomials with $\deg f > \deg g \geq 0$ and R be a regular matrix. For a threshold $t < \deg f$ denote the upper and lower part of f and g:*

$$f_u = f \operatorname{div} x^t, \qquad f_l = f \bmod x^t,$$
$$g_u = g \operatorname{div} x^t, \qquad g_l = g \bmod x^t.$$

Further set $F := F_u x^t + F_l$, $G := G_u x^t + G_l$, where

$$\begin{pmatrix} F_u & F_l \\ G_u & G_l \end{pmatrix} = R^{-1} \cdot \begin{pmatrix} f_u & f_l \\ g_u & g_l \end{pmatrix}.$$

If $\deg F_u \geq \max\left\{1 + \deg G_u, \lceil \deg f_u / 2 \rceil\right\}$, then

$$\deg F = t + \deg F_u \qquad \text{and} \qquad \deg G = t + \max\{\deg G_u, \deg f_u - \deg F_u - 1\}.$$

Proof. Let $R = \left(\begin{smallmatrix} p & q \\ r & s \end{smallmatrix}\right)$. By assumption $f_u = p \cdot F_u + q \cdot G_u$ and $\deg F_u > \deg G_u$. Moreover Lemma 1.3.26 asserts that $\deg p \geq \deg q$. It follows that $\deg f_u = \deg p + \deg F_u$. Now, again by assumption, $2 \deg F_u \geq \deg f_u$ and it follows that $\deg p \leq \deg F_u$.

The determinant of R is ± 1 by Observation 1.3.25 and so

$$R^{-1} = \pm \begin{pmatrix} s & -q \\ -r & p \end{pmatrix}.$$

Consequently $F_l = \pm(sf_l - qg_l)$, hence

$$\begin{aligned}\deg F_l &\le \max\{\deg s + \deg f_l, \deg q + \deg g_l\}\\ &< t + \max\{\deg s, \deg q\}\\ &\le t + \deg p\\ &\le t + \deg F_u.\end{aligned}$$

This implies that $\deg F = t + \deg F_u$.

We now prove the second assertion. We have $G_l = \pm(-rf_l + pg_l)$, hence Lemma 1.3.26 yields

$$\begin{aligned}\deg G_l &\le \max\big\{\deg(rf_l), \deg(pg_l)\big\}\\ &< t + \max\{\deg r, \deg p\}\\ &= t + \deg p\\ &= t + \deg f_u - \deg F_u.\end{aligned}$$

It follows that $\deg G = \deg(G_u \cdot x^t + G_l) \le \max\{t + \deg G_u, t + \deg f_u - \deg F_u - 1\}$ and this is precisely what the lemma says. □

Proof of correctness of Algorithm 1.3.28. The algorithm may terminate in three different ways: outputting the identity matrix in step (2), returning the matrix Q in step (6) or outputting the matrix R in step (11). The regularity of the returned matrix is trivial in all three cases. What we need to check is condition (1.18).

If the algorithm terminates in step (2), then it is obvious that (1.18) holds in this case since $f_i = f$, $f_{i+1} = g$ and $\deg g < t = \lceil \deg f/2 \rceil$. Suppose that the algorithm terminates in step (6). Let $Q = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$. As in Lemma 1.3.29 split the polynomials F, G into their upper and lower parts : $F = F^u \cdot x^t + F^l$ and $G = G^u \cdot x^t + G^l$. To avoid ambiguity of notation we use superscript here, as the upper part G^u of G is not the same as G_u in the algorithm. Consequently we have

$$\begin{pmatrix} F^u & F^l \\ G^u & G^l \end{pmatrix} := Q^{-1} \cdot \begin{pmatrix} f_u & f_l \\ g_u & g_l \end{pmatrix}.$$

The matrix Q, obtained by the recursive call, satisfies condition 1.18 with f, g replaced bu f_u, g_u:

$$\deg F^u \ge \left\lceil \frac{\deg f_u}{2} \right\rceil > \deg G^u.$$

The preceding lemma asserts that

$$\deg F = t + \deg F^u \ge t = \left\lceil \frac{\deg f}{2} \right\rceil.$$

Together with the condition $\deg G < t$ from step (6), this proves the correctness of the result returned in this step.

It remains to prove the correctness of the result returned in the last step. We did not terminate in step (6), hence $l = \deg G \geq t$. Now

$$\deg G_u = \deg G - k = l - (2t + l) = 2(l - t). \tag{1.19}$$

After calling the algorithm recursively in step (10) we obtain a matrix S such that

$$\deg F'_u \geq \left\lceil \frac{\deg G_u}{2} \right\rceil > \deg G'_u, \tag{1.20}$$

where $\binom{F'_u}{G'_u} = S^{-1} \cdot \binom{G_u}{r_u}$. Combining (1.19) and (1.20) we have

$$\deg F'_u \geq l - t > \deg G'_u.$$

Set $\binom{F'}{G'} := S^{-1} \cdot \binom{G}{r}$. Using the equalities $F = qG + r$ and $\binom{F}{G} = Q^{-1} \cdot \binom{f}{g}$ we can write

$$\begin{pmatrix} f \\ g \end{pmatrix} = Q \cdot \begin{pmatrix} F \\ G \end{pmatrix} = Q \cdot \begin{pmatrix} q & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} G \\ r \end{pmatrix} = Q \cdot \begin{pmatrix} q & 1 \\ 1 & 0 \end{pmatrix} \cdot S \cdot \begin{pmatrix} F' \\ G' \end{pmatrix}. \tag{1.21}$$

Lemma 1.3.29 shows that

$$\deg F' = k + \deg G_u = k + (\deg G - k) = \deg G \geq t$$

and

$$\begin{aligned} \deg G' &= k + \max\{\deg G'_u, \deg G_u - \deg F'_u - 1\} \\ &\leq k + \max\{l - t - 1, 2 \cdot (l - t) - (l - t) - 1\} \\ &= k + l - t - 1 = t - 1. \end{aligned}$$

Consequently $\deg F' \geq t > \deg G'$. Together with (1.21) this proves the correctness of the result. □

Complexity analysis

Damien Stehle and Paul Zimmermann, A binary recursive gcd algorithm, Proc. ANTS-VI, LNCS 3076 (2004), 411-–425
Möller, Niels. On Schönhage's algorithm and subquadratic integer GCD computation. Math. Comp. 77 (2008), no. 261, 589–607
K. Thull, C. Yap, A Unified Approach to HGCD Algorithms for polynomials and integers
other sub-quadratic gcd algorithm(s)?

Gcd of more than two arguments

Now, when we know how to compute the greatest common divisor of two elements, it is time to consider gcd of an arbitrary tuple. This is needed for example when calculating the content of a polynomial. One idea is to use the following identity (recall our convention that an equality of greatest common divisors is an equality of sets or equivalently an equality of elements but only up to a multiplication by a unit):

Proposition 1.3.30. *If $a_1, \dots, a_n$ are elements of some unique factorization domain R, then*

$$\gcd(a_1, \dots, a_n) = \gcd\big(a_1, \gcd(a_2, \dots, a_n)\big).$$

Proof. Denote $g_1 := \gcd(a_1, \dots, a_n)$ and $g_2 := \gcd\big(a_1, \gcd(a_2, \dots, a_n)\big)$. By definition, g_1 divides $a_2, \dots, a_n$ and so it divides their greatest common divisor. Hence, g_1 divides both a_1 and $\gcd(a_2, \dots, a_n)$ and it follows that it divides $\gcd\big(a_1, (a_2, \dots, a_n)\big)$ which equals g_2. Conversely, g_2 divides $\gcd(a_2, \dots, a_n)$, hence it divides every a_i for $i \in \{2, \dots, n\}$. Of course, it divides a_1, too. Consequently, dividing all elements $a_1, a_2, \dots, a_n$ it must divide their greatest common divisor. Therefore, g_1 and g_2 are equal up to a multiplication by a unit. □

Algorithm 1.3.31. *Given elements $a_1, \dots, a_n$ of a unique factorization domain, in which the greatest common divisor of any two elements is computable, this algorithm computes* $\gcd(a_1, \dots, a_n)$.

1. *Initialize $g := a_1$;*
2. *for every $j \in \{2, \dots, n\}$:*
   a) *compute* $\gcd(g, a_j)$*, if it is a unit, then return* 1 *and quit;*
   b) *Replace g with* $\gcd(g, a_j)$*;*
3. *return g.*

Least common multiple

The knowledge of the greatest common divisor allows us to compute also the *least common multiple* by the formula

$$\operatorname{lcm}(a, b) = \frac{a \cdot b}{\gcd(a, b)}.$$

Unfortunately, the formula is undefined $^0/_0$, when both a and b are zero. In this case, let us just define $\operatorname{lcm}(0, 0) := 0$.

1.4 Rational numbers and rational functions

This section should be extended

Representation of integers and polynomials, that we discussed above, immediately give rise to an exact representation of rational numbers and rational functions. A rational number (respectively rational function) is given by a pair of integers (resp. polynomials): a numerator and denominator, with well known arithmetical operations:

$$\frac{a}{b} \pm \frac{c}{d} = \frac{ad \pm bc}{bd}, \qquad \frac{a}{b} \cdot \frac{c}{d} = \frac{ac}{bd}, \qquad \left(\frac{a}{b}\right)^{-1} = \frac{b}{a},$$

for all a, b, c, d for which the above fractions make sense. Moreover gcd algorithm provides us with a method of canceling common terms from the numerators and denominators in order to obtain reduced fractions.

1.5 Multivariate polynomials

Missing

1.6 Transcendental constants

> You could trust numbers, except perhaps for pi
>
> "Making Money", Terry Pratchett

> All numbers are by their nature correct. Well, except for Pi, of course. I can't be doing with Pi. Gives me a headache just thinking about it, going on and on and on and on and on...
>
> "Anansi boys", Neil Gaiman

Evaluation of e and π Bellard/Bailey–Borwein–Plouffe formula, "Unbounded Spigot Algorithms for the Digits of Pi" by Jeremy Gibbons.

1.A Scaled Bernstein polynomials

As it was pointed out in Section 1.1, the power basis representation of polynomials is not the only one available, neither the only one actually used. There are other representations. One of them is presented in this appendix. Recall that the coefficients of a polynomial $f \in K[x]$ are nothing else but the coordinates of f with respect to the *power basis* $\mathscr{B} = \{1, x, x^2, \dots\}$ of $K[x]$ treated as a linear space over K. Here, for

simplicity, we consider polynomials over a field, but everything we do can be generalized to an arbitrary ring. For any $n \geq 0$ the set of polynomials whose degree do not exceed n we will denoted

$$K[x]_n := \{f \in K[x] \,|\, \deg f \leq n\}.$$

It is clear that $K[x]_n$ is a subspace of $K[x]$ of dimension $n+1$ and the restriction of the power basis $\mathscr{B} \cap K[x]_n = \{1, x, \ldots, x^n\}$ forms a basis of $K[x]_n$. We will construct a different basis of this space. Fix a degree n and write

$$1 = 1^n = \big((1-x)+x\big)^n = (1-x)^n + n(1-x)^{n-1}x + \cdots + \binom{n}{k}(1-x)^{n-k}x^k + \cdots + x^n.$$

Dropping the binomial coefficients we define

Definition 1.A.1. Polynomials $b_k^n := (1-x)^{n-k}x^k$ for $k \in \{0, \ldots, n\}$ are called the *scaled Bernstein polynomials* of degree n.

It is convenient to define in addition $b_k^n := 0$ for $k < 0$ or $k > n$.

Proposition 1.A.2. *For every $n \geq 0$, the set $\mathscr{B} := \{b_0^n, \ldots, b_n^n\}$ of the scaled Bernstein polynomials is a basis of $K[x]_n$.*

Proof. The power basis of $K[x]_n$ consists of the monomials $1, x, \ldots, x^n$. The scaled Bernstein polynomial b_k^n is divisible by x^k, hence its coordinates with respect to first k elements $1, \ldots, x^{k-1}$ of the power basis are all zero. If we write the coordinates (with respect to the power basis) of $b_0^n, \ldots, b_n^n$ as columns of a matrix, then the resulting matrix will be lower-triangular with a main diagonal consisting of ± 1. The matrix is, thus, invertible and so $b_0^n, \ldots, b_n^n$ are linearly independent. It follows that they form a basis of $K[x]_n$. □

Remark 13. The adjective "scaled" refers to the fact that "standard" Bernstein polynomials of degree n are

$$B_0^n = (1-x)^n = b_0^n, \ \ldots \ B_k^n = \binom{n}{k}(1-x)^{n-k}x^k = \binom{n}{k}b_k^n, \ \ldots \ B_n^n = x^n = b_n^n.$$

They also form a basis of $K[x]_n$, unfortunately the corresponding change-of-basis matrix, although invertible over $\mathbb{Q}$, is not invertible over $\mathbb{Z}$. Hence in order to express all polynomials of degree $\leq n$ in the basis of standard Bernstein polynomials we must use division. This rules them out in some applications, especially when we need to work in $\mathbb{Z}[x]$. Furthermore, the arithmetic of scaled Bernstein polynomials is simpler than the one of their standard counterparts.

Definition 1.A.3. Given a polynomial f of degree $\deg f \leq n$, the *scaled Bernstein-basis representation in degree n* of f is the finite sequence $(c_0, \ldots, c_n)$ of the coordinates of f with respect to the basis of scaled Bernstein polynomials $\{b_0^n, \ldots, b_n^n\}$. The elements $c_0, \ldots, c_n$ are called the *scaled Bernstein coefficients* of f (in degree n).

Example 13. Take a polynomial $f = 6x^3 - 10x^2 + 7x - 2$, the scaled Bernstein-basis coefficients of f in degree 3 are $(-2, 1, -2, 1)$ since

$$f = -2 \cdot b_0^3 + 1 \cdot b_1^3 - 2 \cdot b_2^3 + 1 \cdot b_3^3.$$

The specification of the degree is important in case of the Bernstein-basis representation since the same polynomial can be represented in degree 5 as

$$f = -2 \cdot b_0^5 - 3 \cdot b_1^5 - 2 \cdot b_2^5 - 2 \cdot b_3^5 + 0 \cdot b_4^5 + 1 \cdot b_5^5.$$

The scaled Bernstein-basis representation may seem awkward at first glance, but it has certain unbeatable advantages. For one it is far less prone to round-off errors, when coefficients of polynomials are floating point numbers (for details see).

missing

Before we discuss how to adapt fundamental polygonal algorithm to the new representation, let us notice what happens if we reverse the sequence of Bernstein coefficients of a polynomial f. Recall that in power basis representation we obtain (see Observation 1.1.2) a polynomial $x^{\deg f} \cdot f(1/x)$. For the scaled Bernstein-basis representation the result is different.

Observation 1.A.4. *Let $f = f_0 b_0^n + \cdots + f_n b_n^n$ be a polynomial represented with respect to the scaled Bernstein basis. Set $g = f_n b_0^n + \cdots + f_0 b_n^n$, then*

$$g = f(1-x).$$

This follows from the fact that $b_k^n(x) = b_{n-k}^n(1-x)$ for every degree n and every index k.

Degree elevation

Let $f = f_m x^m + \cdots + f_1 x + f_0$ be a polynomial in the power-basis representation. We can always harmlessly append leading zeros to the list of coefficients since trivially

$$f = f_m x^m + \cdots + f_1 x + f_0 = 0x^n + \cdots + 0x^{m+1} + f_m x^m + \cdots + f_1 x + f_0$$

for every $n > m$. This way we *elevate* the representation of f to degree n. It is no longer so trivial in the scaled Bernstein-basis representation. Yet the elevation is still quite simple. The key tool is the following result.

Lemma 1.A.5. *For every degree $n \geq 0$ and every index $k \in \mathbb{Z}$ except $k = -1$ and $k = n+1$ the following identity holds*

$$b_k^n = b_k^{n+1} + b_{k+1}^{n+1}.$$

Proof. For $k < -1$ and $k > n+1$ all three polynomials: b_k^n, b_k^{n+1} and b_{k+1}^{n+1} are null, by definition. Hence the identity is vacuously true. On the other hand, for $k \in \{0, \ldots, n\}$ we have

$$b_k^n = (1-x) \cdot b_k^n + x \cdot b_k^n = (1-x)^{n-k+1} x^k + (1-x)^{n-k} x^{k+1} = b_k^{n+1} + b_{k+1}^{n+1}. \qquad \square$$

It is easy to turn the lemma into an algorithm for the degree elevation in the scaled Bernstein representation.

Algorithm 1.A.6. *Let $f = f_0 b_0^m + \cdots + f_m b_m^m$ be a polynomial, specified by its scaled Bernstein coefficients in degree $m \geq \deg f$. Given a degree $n > m$, this algorithm computes the scaled Bernstein coefficients of f in degree n.*

1. *Set $k := n - m$ and construct a list of binomial coefficients (a row a the Pascal triangle)*
$$\binom{k}{0}, \binom{k}{2}, \ldots, \binom{k}{k}.$$
2. *Compute the scaled Bernstein coefficients in degree n of f by the formula*
$$F_j := \sum_{i=0}^{k} \binom{k}{i} \cdot f_{i+j-k}, \qquad \text{for} \quad j \in \{0, \ldots, n\},$$
with the convention that f_l is null whenever $l < 0$ or $l > n$.
3. *Output $(F_0, \ldots, F_n)$.*

Example 14. Let us now return to the polynomial $f = -2b_0^3 + 1b_1^3 - 2b_2^3 + 1b_3^3$ from Example 13 and see how we elevated its degree from 3 to 5. The list of binomial coefficients, computed in the first step of the algorithm is just the third raw of the standard Pascal triangle,
$$\left(\binom{2}{0}, \binom{2}{1}, \binom{2}{2}\right) = (1, 2, 1).$$
The elevated coefficients computed using the previous algorithm are:
$$\begin{aligned} F_0 &= 1 \cdot 0 + 2 \cdot 0 + 1 \cdot c_0 = -2, & F_1 &= 1 \cdot 0 + 2 \cdot f_0 + 1 \cdot f_1 = -3, \\ F_2 &= 1 \cdot f_0 + 2 \cdot f_1 + 1 \cdot f_2 = -2, & F_3 &= 1 \cdot f_1 + 2 \cdot f_2 + 1 \cdot f_3 = -2, \\ F_4 &= 1 \cdot f_2 + 2 \cdot f_3 + 1 \cdot 0 = 0, & F_5 &= 1 \cdot f_3 + 2 \cdot 0 + 1 \cdot 0 = 1. \end{aligned}$$
Consequently $f = -2b_0^5 - 3b_1^5 - 2b_2^5 - 2b_3^5 + 0b_4^5 + 1b_5^5$.

Remark 14. Observe that elevating a degree of a polynomial (in scaled Bernstein-basis representation) by one, we add the list of its coefficients to its copy shifted by a single slot, where both list are padded with zeros to match the lengths. Iterating this operation one elevates a polynomial representation to any degree. In case of the polynomial f from the previous example, we first add its list of coefficients to its shifted copy to get degree four representation:
$$(-2, 1, -2, 1, 0) + (0, -2, 1, -2, 1) = (-2, -1, -1, -1, 1).$$
We then repeat the same procedure to get the sought degree five representation:
$$(-2, -1, -1, -1, 1, 0) + (0, -2, -1, -1, -1, 1) = (-2, -3, -2, -2, 0, 1).$$
This step-by-step elevation approach may be preferable when implementing the degree elevation on some architectures.

A process opposite to degree elevation is called de-elevation. If the degree of a polynomial is lower than the degree of its representation, it may be desirable to de-elevate it. Here we restrict the discussion to the reduction of the degree by one, since this is exactly what is need for polynomial division in the scaled Bernstein-basis representation (see Algorithm 1.A.18). Let $F = F_0 b_0^n + \cdots + F_n b_n^n$ and assume that $\deg F < n$. Hence there are coefficients $f_0, \ldots, f_{n-1}$ such that $f_0 b_0^{n-1} + \cdots + f_{n-1} b_{n-1}^{n-1} = F$. As explained in the above remark, the "elevated" coefficients $(F_0, \ldots, F_n)$ are obtained by adding a vector $(f_0, \ldots, f_{n-1})$ to its shifted copy:

$$\begin{pmatrix} f_0 \\ \vdots \\ f_{n-1} \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ f_0 \\ \vdots \\ f_{n-1} \end{pmatrix} = \begin{pmatrix} F_0 \\ F_1 \\ \vdots \\ F_n \end{pmatrix}.$$

It is clear that one can sequentially recover $f_0, \ldots f_{n-1}$ taking

$$\begin{aligned} f_0 &= F_0, \\ f_1 &= F_1 - f_0, \\ &\vdots \\ f_{n-1} &= F_{n-1} - f_{n-1} = F_n. \end{aligned}$$

Let us write this procedure in form of an algorithm.

Algorithm 1.A.7. *Let $F = F_0 b_0^n + \cdots + F_n b_n^n$ be a polynomial. This algorithm returns its scaled Bernstein coefficients in degree $n-1$ or throws an exception if it cannot be done (i.e. if the degree of F is n).*

1. *Set $f_0 := F_0$.*
2. *For $j = 1, \ldots, n-1$ set $f_j := F_j - f_{j-1}$.*
3. *If $f_{n-1} \neq F_n$ then report an error.*
4. *Output $(f_0, \ldots, f_{n-1})$.*

Representation conversion

The next thing that we discuss is how to convert polynomials from the power-basis representation to the scaled Bernstein-basis representation, et vice versa. Such a conversion is known to be numerically unstable (for details see), yet in some cases is unavoidable. For instance, even if a program internally uses exclusively the scaled Bernstein-basis representation, its input and output may be required to be in human readable format and most people expect polynomials to be presented in "traditional form", this means in the power-basis representation.

missing

Take a scaled Bernstein polynomial b_k^n and expand it using standard binomial formula

$$b_k^n = (1-x)^{n-k}\cdot x^k = \left(\sum_{i=0}^{n-k}(-1)^i\cdot\binom{n-k}{i}\cdot x^i\right)\cdot x^k$$
$$= \sum_{i=0}^{n-k}(-1)^i\cdot\binom{n-k}{i}\cdot x^{i+k} = \sum_{j=k}^{n}(-1)^{j-k}\cdot\binom{n-k}{j-k}\cdot x^j.$$

Applying this identity to every term of a polynomial we may convert it from the scaled Bernstein basis to the power basis.

Observation 1.A.8. *If $f = c_0 b_0^n + \cdots + c_n b_n^n$ is a polynomial in the scaled Bernstein-basis representation, then its power-basis coefficients $f_0, \ldots, f_n$ are computed by the formula*

$$f_j = \sum_{k=0}^{j}(-1)^{j-k}\cdot\binom{n-k}{j-k}\cdot c_k,$$

for $j \in \{0, \ldots, n\}$.

The above observation translates directly to an algorithm.

Algorithm 1.A.9. *Let $f = c_0 b_0^n + \cdots + c_n b_n^n$ be a polynomial specified by its scaled Bernstein-basis coefficients. This algorithm computes its power-basis representation.*

1. *Initialize the list of power-basis coefficients, setting $f_n = \cdots = f_0 = 0$.*
2. *For every index $k \in \{n, n-1, \ldots, 0\}$ proceed as follows:*
   a) *Compute the $(n-k)$th row of the Pascal triangle $\binom{d-k}{0}, \ldots, \binom{d-k}{d-k}$.*
   b) *For every $j \in \{k, \ldots, n\}$ update the coefficient f_j setting*

$$f_j := f_j + (-1)^{j-k}\cdot\binom{d-k}{j-k}\cdot c_k.$$

3. *Prune leading zeros $f_n = \cdots = f_{k+1} = 0$, if any.*
4. *Output the sequence of coefficients of f in power-basis representation: $f_k, \ldots, f_0$.*

Time complexity analysis. If we construct the Pascal triangle row-by-row using Observation (1.1.22), then step (2a) uses only $\max\{0, 2\cdot(n-k-1)\}$ integer additions and no multiplications. Step (2b), on the other hand, uses one multiplication and one addition. All in all, Algorithm 1.A.9 uses n^2 additions and n multiplications.

To obtain the opposite conversion we make a simple observation that $x^k = b_k^k$. Elevating the degree, we may represent b_k^k in degree n as

$$b_k^k = \sum_{j=k}^{n}\binom{n-k}{j-k}b_j^n.$$

Summing over all the monomials of $f = f_n x^n + \cdots + f_1 x + f_0 = f_n b_n^n + \cdots + f_1 b_1^1 + f_0 b_0^0$, we derive its scaled Bernstein-basis representation.

Observation 1.A.10. *If $f = f_n x^n + \cdots + f_1 x + f_0$ is a polynomial in the power-basis representation, then $f = c_0 b_0^n + \cdots + c_n b_n^n$, where*

$$c_j = \sum_{k=0}^{j} \binom{n-k}{j-k} f_k.$$

Again, this observation can be directly converted into an algorithm.

Algorithm 1.A.11. *Let $f = f_n x^n + \cdots + f_1 x + f_0$ be a polynomial specified by its power-basis coefficients. This algorithm computes its scaled Bernstein-basis representation in degree n.*

1. *Initialize the list of scaled Bernstein-basis coefficients, setting $c_0 = \cdots = c_n = 0$.*
2. *For every index $k \in \{n, n-1, \ldots, 0\}$ proceed as follows:*
   a) *Compute the $(d-k)$th row of the Pascal triangle: $\binom{n-k}{0}, \ldots, \binom{n-k}{n-k}$.*
   b) *For every $j \in \{k, \ldots, n\}$ update the coefficient c_j setting $c_j := c_j + \binom{n-k}{j-k} \cdot f_k$.*
3. *Output the sequence of coefficients of f in the scaled Bernstein-basis representation: $c_0, \ldots, c_n$.*

Time complexity analysis. This algorithm uses the same number of arithmetic operations as Algorithm 1.A.9, hence n^2 additions and n multiplications.

Example 15. We can now explain how we calculated scaled Bernstein coefficient of $f = 6x^3 - 10x^2 + 7x - 2$ in Example 13. The following table summarizes successive steps of Algorithm 1.A.11:

k	f_k	c_0	c_1	c_2	c_3
—	—	0	0	0	0
3	6				$6\binom{0}{0} = 6$
2	-10			$-10\binom{1}{0} = -10$	$6 - 10\binom{1}{1} = -4$
1	7		$7\binom{2}{0} = 7$	$-10 + 7\binom{2}{1} = 4$	$-4 + 7\binom{2}{2} = 3$
0	-2	$-2\binom{3}{0} = -2$	$7 - 2\binom{3}{1} = 1$	$4 - 2\binom{3}{2} = -2$	$3 - 2\binom{3}{3} = 1$

Remark 15. One may wonder if there is a correspondence between a polynomial f having certain coordinates with respect to the power basis and a polynomial g having exactly the same coordinates but with respect to the scaled Bernstein basis. The answer is very simple. Take a polynomial $f = f_n x^n + \cdots + f_1 x + f_0$ and construct a *homogeneous* bivariate polynomial

$$F(x, y) := y^n \cdot f\left(\frac{x}{y}\right) = f_n x^n + \cdots + f_1 y^{n-1} x + f_0 y^n.$$

Then

$$g = F(x, 1-x) = (1-x)^{\deg f} \cdot f\Big(\frac{x}{1-x}\Big).$$

Conversely, if $g = g_0 b_0^n + g_1 b_1^n + \cdots + g_n b_n^n$ is a polynomial in the scaled Bernstein-basis representation in degree n and $f = g_n x^n + \cdots + g_1 x + g_0$ is a corresponding polynomial in the power-basis representation, then inverting the previous formula we have

$$f = (1+x)^n \cdot g\Big(\frac{x}{1+x}\Big).$$

Addition and subtraction

The scaled Bernstein coefficients of a polynomial are just its coordinates with respect to the scaled Bernstein basis of a given degree. Hence addition and subtraction of two polynomials can be done coefficient-wisely provided that their representation degrees coincide (the important one is the degree of the representation, not the true degree of a polynomial). Otherwise, if two polynomials are represented in different degrees, we must elevate the lower degree to match the higher one. Only then we can add/subtract the polynomials coefficient-wisely, exactly as in the power-basis case.

Algorithm 1.A.12. *Given two polynomials $f = f_0 b_0^m + \cdots + f_m b_m^m$ and $g = g_0 b_0^n + \cdots + g_n b_n^n$ in the scaled Bernstein-basis representation, this algorithm computes their sum* (*respectively difference*).

1. *If the degrees m and n differ, then elevate the lower degree to match the higher one. Denote the common degree of both representations by d.*
2. *Let $f_0, \dots, f_d$ and $g_0, \dots, g_d$ be the coefficients of f and g, respectively.*
3. *For every $i \in \{0, \dots, d\}$, set $h_i := f_i + g_i$ (respectively $h := f_i - g_i$).*
4. *Output $h = h_0 b_0^d + h_1 b_1^d + \cdots + h_k b_d^d$.*

Multiplication

Surprisingly the long-multiplication formula, that we used for polynomials in the power-basis representation, can be used also for polynomials in the scaled Bernstein-basis representation. In particular, Algorithm 1.1.7 can be directly applied to polynomials in any of the two representations.

Proposition 1.A.13. *If $f = f_0 b_0^m + \cdots + f_m b_0^m$ and $g = g_0 b_0^n + \cdots + g_n b_0^n$, then the scaled Bernstein coefficients of the product $h = f \cdot g$ are $h_0, \dots, h_k$, where $k = m + n$ and*

$$h_i := \sum_{j=\max\{0, i-n\}}^{\min\{i,m\}} f_j g_{i-j}, \qquad \text{for} \quad i \in \{0, \dots, k\}.$$

Proof. Write

$$h = \Big(\sum_{i=0}^{m} f_i b_i^m\Big)\Big(\sum_{j=0}^{n} g_j b_j^n\Big) = \sum_{\substack{0 \le i \le m \\ 0 \le j \le n}} f_i g_j b_i^m b_j^n = \sum_{\substack{0 \le i \le m \\ 0 \le j \le n}} f_i g_j (1-x)^{(m+n)-(i+j)} x^{i+j}.$$

Gathering the common terms we prove the assertion. □

In the first section we discovered that (asymptotically) we may substantially reduce the cost of multiplication using divide-and-conquer approach due to Karatsuba. This idea can be adapted to the scaled Bernstein-basis representation, as well. To this end we need the following simple, yet important, identity.

Observation 1.A.14. *For every non-negative integers k, l, m, n we have*

$$b_k^m \cdot b_l^n = (1-x)^{m-k+n-l} \cdot x^{k+l} = b_{k+l}^{m+n}.$$

Take a polynomial $f = f_0 b_0^n + \cdots + f_n b_n^n$ in the scaled Bernstein-basis representation and let $d := \lceil n/2 \rceil$. Subdivide f into its lower and upper parts f_l and f_u as follows. If n is odd, then just split the list of the coefficients in half, exactly as in the power-basis case:

$$\begin{aligned} f &= f_0 b_0^{2d-1} + \cdots + f_{d-1} b_{d-1}^{2d-1} + f_d b_d^{2d-1} + \cdots + f_{2d-1} b_{2d-1}^{2d-1} \\ &= \left(f_0 b_0^{d-1} + \cdots + f_{d-1} b_{d-1}^{d-1}\right) \cdot b_0^d + \left(f_d b_0^{d-1} + \cdots + f_{2d-1} b_{d-1}^{d-1}\right) \cdot b_d^d \\ &= f_l b_0^d + f_u b_d^d. \end{aligned}$$

On the other hand, if n is even, then $n = 2d$ and splitting the list we pad the **lower part** with a single zero:

$$\begin{aligned} f &= f_0 b_0^{2d} + \cdots + f_{d-1} b_{d-1}^{2d} + f_d b_d^{2d} + \cdots + f_{2d} b_{2d}^{2d} \\ &= \left(f_0 b_0^d + \cdots + f_{d-1} b_{d-1}^d + 0 b_d^d\right) \cdot b_0^d + \left(f_d b_0^d + \cdots + f_{2d} b_d^d\right) \cdot b_d^d \\ &= f_l \cdot b_0^d + f_u \cdot b_d^d. \end{aligned}$$

Now, given two polynomials f and g with the same representation degree, we multiply them using Karatsuba's formula, as follows:

$$\begin{aligned} f \cdot g &= \left(f_l b_0^d + f_u b_d^d\right) \cdot \left(g_l b_0^d + g_u b_d^d\right) \\ &= f_l b_0^d g_l b_0^d + \left((f_l + f_u)(g_l + g_u) - f_l g_l - g_u g_u\right) b_0^d b_d^d + f_u b_d^d g_u b_d^d \\ &= (f_l g_l) b_0^{2d} + \left((f_l + f_u)(g_l + g_u) - f_l g_l - g_u g_u\right) b_d^{2d} + (f_u g_u) b_{2d}^{2d}. \end{aligned}$$

The algorithm is now clear and nearly identical to Algorithm 1.1.9, yet for the sake of completeness we write it explicitly.

Algorithm 1.A.15. *Let $f = f_0 b_0^m + \cdots + f_m b_m^m$ and $g = g_0 b_0^n + \cdots + g_n b_n^n$ be two polynomials in the scaled Bernstein-basis representation. This algorithm computes the product $h := f \cdot g$.*

1. *If any of the two polynomials is zero, then output* 0 *and terminate.*
2. *If $m = 0$, then output $(f_0 g_0) b_0^n + \cdots + (f_0 g_n) b_n^n$ and terminate.*
3. *Analogously, if $n = 0$, then output $(f_0 g_0) b_0^m + \cdots + (f_m g_0) b_m^m$ and terminate.*

4. *If $m \neq n$, then elevate the lower degree to match the higher one.*
5. *Denote the common degree of the representation of f and g by D and set $d := \lceil D/2 \rceil$.*
6. *Split both polynomials into their lower and upper parts, setting:*
   a) *if $D = 2d - 1$:*

$$f_l := f_0 b_0^{d-1} + \cdots + f_{d-1} b_{d-1}^{d-1}, \qquad f_u := f_d b_0^{d-1} + \cdots + f_{2d-1} b_{d-1}^{d-1},$$
$$g_l := g_0 b_0^{d-1} + \cdots + g_{d-1} b_{d-1}^{d-1}, \qquad g_u := g_d b_0^{d-1} + \cdots + g_{2d-1} b_{d-1}^{d-1};$$

   b) *otherwise, if $D = 2d$:*

$$f_l := f_0 b_0^{d} + \cdots + f_{d-1} b_{d-1}^{d} + 0 b_d^d, \qquad f_u := f_d b_0^{d} + \cdots + f_{2d} b_d^d,$$
$$g_l := g_0 b_0^{d} + \cdots + g_{d-1} b_{d-1}^{d} + 0 b_d^d, \qquad g_u := g_d b_0^{d} + \cdots + g_{2d} b_d^d.$$

7. *Recursively compute the products:*

$$h_l := f_l g_l, \qquad h_c := (f_l + f_u)(g_l + g_u), \qquad h_u := f_u g_u.$$

8. *Re-combine the partial results setting*

$$h = h_l \cdot b_0^{2d} + (h_c - h_l - h_u) \cdot b_d^{2d} + h_u \cdot b_{2d}^{2d}.$$

9. *Output h.*

Remark 16. The degree elevation is executed at most once during the recursion, since the polynomials constructed in step (6) always have equal degrees.

Time complexity analysis. The time complexity of the algorithm is the same as of Algorithm 1.1.9, that is

$$O(D^{\log_2 3} \cdot \text{complexity of arithmetic operations on coefficients}).$$

Exercise 5. Let f, g be two polynomials (in the scaled Bernstein-basis representation) whose degrees differ by one and the higher one is odd. Prove that in this case the degree elevation can be omitted.

Division and quasi-division

The last of the four basic arithmetic operations is the division. Recall (see Proposition 1.1.11) that in a ring of univariate polynomials over a field we have a well defines division with remainders. The key to compute the quotient and remainder of two polynomials given in the scaled Bernstein forms is the following observation, which can be derived by analyzing the procedure of converting polynomial representation from the scaled Bernstein basis to the power basis (see Algorithm 1.A.9).

Observation 1.A.16. *Let $f = f_0 b_0^m + \cdots + f_m b_m^m$ be a polynomial in the scaled Bernstein-basis form. Then the coefficient of f at x^m in the power-basis representation is*

$$\sum_{j=0}^{m} (-1)^{m-j} \cdot f_j.$$

Corollary 1.A.17. *Let $f = f_0 b_0^m + \cdots + f_m b_m^m$ be as above, then the degree of the polynomial*

$$r := f - \left(\sum_{j=0}^{m} (-1)^{m-j} \cdot f_j \right) \cdot b_m^m \tag{1.22}$$

is strictly less than m.

The degree in the above corollary is the degree of the polynomial r not its representation. Now the idea of dividing two polynomials in the scaled Bernstein-basis form is to adapt Algorithm 1.1.12 so that it uses (1.22) in place of (1.3) and de-elevates the degree of the remainder after each step.

Algorithm 1.A.18 (Polynomial division in the scaled Bernstein-basis form)**.** *Let $f = f_0 b_0^m + \cdots + f_m b_m^m$ and $g = g_0 b_0^n + \cdots + g_n b_n^n$ be two Bernstein polynomials in the scaled Bernstein-basis representation with coefficients in a field K. Assume that the degree of the representation of g is minimal (i.e. $n = \deg g$). This algorithm computes the quotient q and remained r of f modulo g.*

1. *Compute the leading coefficient of g (with respect to the power basis):*

$$c := \sum_{j=0}^{n} (-1)^{n-j} \cdot g_j.$$

2. *Initialize the quotient $q := 0 b_0^0$ and remainder $r := f$.*
3. *For $k \in \{m, m-1, \ldots, n\}$ proceed as follows:*
   a) *Compute the next power-basis coefficient of the quotient, setting*

$$d := \frac{1}{c} \cdot \sum_{j=0}^{k} (-1)^{k-j} \cdot r_j.$$

   b) *Use Observation* (1.1.22) *to construct a raw of the Pascal triangle*

$$\binom{m-k}{0}, \binom{m-k}{1}, \ldots, \binom{m-k}{m-k}.$$

   c) *Update the quotient, setting*

$$q := q \cdot b_0^1 + \sum_{j=1}^{m-k} \binom{m-k}{j} \cdot d \cdot b_j^{m-k}.$$

d) Update the remainder, setting $r := r - d \cdot g \cdot b_0^{k-n}$.

e) Use Algorithm 1.A.7 to de-elevate the degree of the representation of r.

4. Output q *and* r.

We leave it to the reader to prove the correctness of the algorithm and analyze its complexity. Instead we provide a single example.

Example 16. Take the following two polynomials in the scaled Bernstein-basis representation (in parenthesis we show their power-basis forms):

$$f = -5b_0^3 - 14b_1^3 - 6b_2^3 + 4b_3^3 \ (= x^3 + 7x^2 + x - 5)$$
$$g = -1b_0^2 + 0b_1^2 + 2b_2^2 \ (= x^2 + 2x - 1).$$

The variable c (i.e. the leading coefficient of g in the power-basis form) equals 1. The following table contains the successive steps of the algorithm:

k	d	q	r	r de-elevated
—	—	$0b_0^0$	$-5b_0^3 - 14b_1^3 - 6b_2^3 + 4b_3^3$	$-5b_0^3 - 14b_1^3 - 6b_2^3 + 4b_3^3$
3	1	$1b_0^0$	$-5b_0^3 - 13b_1^3 - 6b_2^3 + 2b_3^3$	$-5b_0^2 - 8b_1^2 + 2b_2^2$
2	5	$5b_0^1 + 6b_1^1$	$0b_0^2 - 8b_1^2 - 8b_2^2$	$0b_0^1 - 8b_1^1$

Thus, the quotient equals $5b_0^1 + 6b_1^1$ $(= x+5)$ and the remainder is $0b_0^1 - 8b_1^1$ $(= -8x)$.

It is evident that the "true" division with remainders of polynomials in the scaled Bernstein-basis forms is equivalent to converting the polynomials to the power-basis representation, computing the quotient and remainder and converting back. Consequently the whole process suffers from all flaws of the conversion procedure including numerical instability. One may wonder if we can do better. In fact in some case we can. We may replace the "true" division by another similar process. First let us notice that an exact division by b_0^k (respectively b_k^k) boils down to dropping k leading (respectively trailing) zeros. This fact is an immediate consequence of Observation 1.A.14.

Observation 1.A.19. *Let* $f = f_0 b_0^n + \cdots + f_n b_n^n$ *be a polynomial in the scaled Bernstein-basis form.*

1. If $f_{k+1} = \cdots = f_n = 0$*, then* f *may be factored into*

$$f = (f_0 b_0^k + \cdots + f_k b_k^k) \cdot b_0^{n-k}.$$

2. If $f_0 = \cdots = f_{k-1} = 0$*, then* f *may be factored into*

$$f = (f_0 b_0^{n-k} + \cdots + f_n b_{n-k}^{n-k}) \cdot b_k^k.$$

Now, let $f = f_0 b_0^m + \cdots + f_m b_m^m$ and $g = g_0 b_0^n + \cdots + g_n b_n^n$ be two polynomials with the coefficients in a field K. Assume that $m \geq n$ (here we compare degrees of representations not the true degrees of the polynomials). Let $k := \max\{j \mid g_j \neq 0\}$ be the biggest index of a nonzero coefficient of g. Write g in the form $g = \hat{g} \cdot b_0^{n-k}$, where

$\hat{g} = g_0 b_0^k + \cdots + g_k b_k^k$ is obtained from g by dropping the leading zeros. We may cancel the term $f_m b_m^m$ setting

$$r^{(1)} := f - \frac{f_m}{g_k} \cdot \hat{g} \cdot b_{m-k}^{m-k},$$

exactly as we did in step (3c) of Algorithm 1.1.12. The resulting polynomial $r^{(1)}$ has a form

$$r^{(1)} = r_0^{(1)} b_0^m + \cdots + r_{m-1}^{(1)} b_{m-1}^m + 0 b_m^m.$$

Hence is divisible by b_0^1. Next we cancel the term $r_{m-1}^{(1)}$ writing

$$r^{(2)} := r^{(1)} - \frac{r_{m-1}^{(1)}}{g_k} \cdot \hat{g} \cdot b_{m-k-1}^{m-k}.$$

The polynomial $r^{(2)}$ is divisible by b_0^2. We may now continue the process constructing polynomials

$$\begin{aligned} r^{(3)} &:= r^{(2)} - \frac{r_{m-2}^{(2)}}{g_k} \cdot \hat{g} \cdot b_{m-k-2}^{m-k}, \\ &\vdots \\ r^{(m-k+1)} &:= r^{(m-k)} - \frac{r_k^{(m-k)}}{g_k} \cdot \hat{g} \cdot b_0^{m-k}. \end{aligned}$$

The process stops, when all coefficients with indices greater than or equal to k are zero. In particular, the polynomial $r^{(m-k+1)}$ is divisible by b_0^{m-k}. Denote the quotient by r so that $r^{(m-k+1)} = r \cdot b_0^{m-k}$. Set

$$q := \frac{r_k^{(m-k)}}{g_k} \cdot b_0^{m-k} + \cdots + \frac{r_{m-1}^{(1)}}{g_k} \cdot b_{m-k-1}^{m-k} + \frac{f_m}{g_k} \cdot b_{m-k}^{m-k}.$$

We have

$$f = q \cdot \hat{g} + r \cdot b_0^{m-k}, \qquad \deg r < k. \tag{1.23}$$

Moreover, in case when $g_n \neq 0$ the formula simplifies to $f = q \cdot g + r \cdot b_0^{m-n}$, which resembles the standard formula of the quotient and remainder of the Euclidean division. In general, the polynomials q and r **are not** the quotient and remainder of f modulo g, although in some applications they can play a similar role. We call them respectively the *quasi-quotient* and *quasi-remainder*, because the whole process closely resembles polynomial division in the power-basis representation.

For the sake of completeness, if the degree m of the representation of f is smaller than the degree n of the representation of g, we define the quasi-quotient to be zero and the quasi-remainder to be f.

Example 17. Take the same two polynomials as in the previous example

$$\begin{aligned} f &= -5b_0^3 - 14b_1^3 - 6b_2^3 + 4b_3^3 \ (= x^3 + 7x^2 + x - 5) \\ g &= -1b_0^2 + 0b_1^2 + 2b_2^2 \ (= x^2 + 2x - 1). \end{aligned}$$

The quasi-quotient and quasi-remainder of f modulo g are computed in the following sequence of steps

$$\begin{aligned} r^{(1)} &= -5b_0^3 - 12b_1^3 - 6b_2^3 + 0b_3^3, & q_1 &= 0b_0^1 + 2b_1^1 \\ r^{(2)} &= -8b_0^3 - 12b_1^3 + 0b_2^3 + 0b_3^3, & q_2 &= -3b_0^1 + 2b_1^1 \end{aligned}$$

Eventually

$$q = -3 \cdot b_0^1 + 2 \cdot b_1^1 \ (= 5x - 3), \qquad \text{and} \qquad r = -8 \cdot b_0^1 - 12 \cdot b_1^1 \ (= -4x - 8).$$

Remark 17. Observe that in the above procedure, instead of canceling terms right-to-left, starting from $f_m b_m^m$, we may also cancel them left-to-right, starting from $f_0 b_0^m$. This is equivalent to reversing the order of coefficients, preforming the quasi-division and reversing back.

Remark 18. We can relate the quasi-quotient/quasi-remainder to the "true" ones using Remark 15. As before let $f = f_0 b_0^m + \cdots + f_m b_m^m$ and $g = g_0 b_0^n + \cdots + g_n b_n^n$ be two polynomials in the scaled Bernstein-basis form. Take

$$f^* = (1+x)^m \cdot f\Big(\frac{x}{1+x}\Big) \qquad \text{and} \qquad g^* = (1+x)^n \cdot g\Big(\frac{x}{1+x}\Big)$$

and let q^* and r^* be respectively the (true) quotient and remainder of f^* modulo g^*. Then the quasi-quotient q and quasi-remainder r can be expressed in terms of q^* and r^* as

$$q = (1-x)^{\deg q^*} \cdot q^*\Big(\frac{x}{1-x}\Big) \qquad \text{and} \qquad r = (1-x)^{\deg r^*} \cdot r^*\Big(\frac{x}{1-x}\Big).$$

Analogously to the power-basis case, when considering polynomials over rings, we may avoid fractional coefficients by pre-multiplying f by a power of g_k. We omit the trivial details, presenting only the conclusion.

Observation 1.A.20. *Let $f = f_0 b_0^m + \cdots + f_m b_m^m$ and $g = g_0 b_0^n + \cdots + g_n b_n^n$ be two polynomials in the scaled Bernstein-basis representation. Assume that $g = \hat{g} \cdot b_0^k$ and $g_k \neq 0$. Then there are polynomials q, r such that*

$$g_k^d \cdot f = q \cdot \hat{g} + r \cdot b_0^{m-k},$$

where $\deg r < k$ and $d = \max\{0, m-k+1\}$.

We let the reader adapt Algorithm 1.1.17 to compute q and r as in the previous observation.

Differentiation

Differentiation of polynomials in the scaled Bernstein-basis form is not much harder than differentiation of polynomials in the power-basis representation. First compute

the derivative of a single basis element:

$$\begin{aligned}\left(b_k^n\right)' &= \left((1-x)^{n-k}\cdot x^k\right)' \\ &= -(n-k)(1-x)^{n-k-1}\cdot x^k + (1-x)^{n-k}\cdot k\cdot x^{k-1} \\ &= k\cdot b_{k-1}^{n-1} - (n-k)\cdot b_k^{n-1}.\end{aligned}$$

Observe that the resulting identity holds for every $k\in\{0,\ldots,d\}$, including the border cases when $k=0$ or $k=d$, due to the fact that the corresponding coefficients turn out to be zero. This proves the correctness of the following algorithm.

Algorithm 1.A.21. *Given a polynomial $f = f_0 b_0^n + \cdots + f_n b_n^n$ in the scaled Bernstein-basis representation, this algorithm computes its derivative.*

1. *Initialize the first coefficient $h_0 := -n\cdot f_0$.*
2. *For each index $i\in\{1,\ldots,n-1\}$ do:*
   a) *Update the previous coefficient setting $h_{i-1} := h_{i-1} + i\cdot f_i$.*
   b) *Initialize the next coefficients $h_i := (i-n)\cdot f_i$.*
3. *Update the last coefficient setting $h_{n-1} := h_{n-1} + n\cdot f_n$.*
4. *Output the result $h = h_0 b_0^{n-1} + \cdots + h_{n-1} b_{n-1}^{n-1}$.*

Time complexity analysis. Each iteration of the loop involves two integer-coefficient multiplications, one addition of coefficients and one subtraction of integers. Hence, for a polynomial represented in degree n, this algorithm uses a total of $2n$ multiplications and $2n$ additions.

Example 18. Take $f = 3\cdot b_0^3 - 2\cdot b_1^3 + 1\cdot b_2^3 - 5\cdot b_3^3$. The successive steps of computing the derivative are:

$$\begin{aligned} f' &= -9b_0^2 && && && (i=0)\\ &\quad +(-2)b_0^2 && +(4)b_1^2 && && (i=1)\\ &\quad && +(2)b_1^2 && +(-1)b_2^2 && (i=2)\\ &\quad && && +(-15)b_2^2 && (i=3)\\ &= -11\cdot b_0^2 + 6\cdot b_1^2 - 16\cdot b_2^2. \end{aligned}$$

Indeed, converting f and f' to the power-basis representation we have

$$f = -11x^3 + 14x^2 - 11x + 3, \qquad f' = -33x^2 + 28x - 11.$$

Polynomial evaluation and composition

When it comes to evaluating polynomials represented with respect to the scaled Bernstein-basis, we basically have two options. The first one is to adapt Horner scheme discussed earlier, as we do in the following algorithm. The second one is to use de Casteljau algorithm. This will be described next.

Algorithm 1.A.22. *Let $f = f_0 b_0^d + \cdots + f_n b_n^n$ be a polynomial in the scaled Bernstein-basis representation and α be an argument. This algorithm computes $f(\alpha)$.*

1. *Initialize $a := \alpha$ and $r := f_0 \cdot (1-\alpha)$.*
2. *For every $i \in \{1, \ldots, n-1\}$ do:*
   a) *Update the result, setting $r := (r + f_i \cdot a) \cdot (1-\alpha)$.*
   b) *Update the auxiliary variable a, setting $a := a \cdot \alpha$.*
3. *Update the result one last time, setting $r := r + f_n \cdot a$;*
4. *Output r.*

Proof of correctness. It is obvious that the algorithm terminates in finite time. The correctness of the result follows by a simple induction. We claim that after k iterations of the main loop one has

$$r = (1-\alpha) \cdot \sum_{i=0}^{k} f_i \cdot (1-\alpha)^{k-i} \cdot \alpha^i = (1-\alpha) \cdot \sum_{i=0}^{k} f_i b_i^k(\alpha).$$

It is true for $k = 0$, since we initialized r with $f_0 \cdot (1-\alpha)$. Suppose that the identity holds for $k-1$, then after k-th iteration we have

$$\begin{aligned} r &= (r + f_k \cdot \alpha^k) \cdot (1-\alpha) \\ &= (1-\alpha) \cdot \sum_{i=0}^{k-1} f_i \cdot (1-\alpha)^{k-i} \alpha^i + f_k \cdot (1-\alpha)\alpha^k \\ &= (1-\alpha) \cdot \sum_{i=0}^{k} f_i b_i^k(\alpha). \end{aligned}$$

Therefore, after step (3) we have

$$r = (1-\alpha) \cdot \sum_{i=0}^{n-1} f_i b_i^{n-1}(\alpha) + f_n \cdot \alpha^n = \sum_{i=0}^{n-1} f_i b_i^n(\alpha) + f_n \cdot \alpha^n = f(\alpha) \qquad \square$$

As mentioned above, there is also a second approach to computing $f(\alpha)$, known as de Casteljau algorithm. For any $k > 0$ and $k < n$, we have

$$\begin{aligned} b_k^n &= (1-x)^{n-k} x^k \\ &= \frac{x}{2} \cdot (1-x)^{(n-1)-(k-1)} \cdot x^{k-1} + \frac{1-x}{2} \cdot (1-x)^{(n-1)-k} \cdot x^k \\ &= \frac{1}{2} \cdot \left(x \cdot b_{k-1}^{n-1} + (1-x) \cdot b_k^{n-1} \right). \end{aligned}$$

The two exceptional cases $k = 0$ and $k = n$ are even simpler:

$$b_0^n = (1-x) \cdot b_0^{n-1} \qquad \text{and} \qquad b_n^n = x \cdot b_{n-1}^{n-1}.$$

Take now a polynomial $f = f_0 b_0^n + \cdots + f_n b_n^n$ and fix an argument α. We wish to compute $f(\alpha)$ applying a certain iterative procedure to the coefficients. The original coefficients of f can be treated as 0-th iterate of the procedure, hence, in order to unify notation, we shall adorn them with a superscript 0, so that $f_i^0 := f_i$ for every $0 \le i \le n$. Using the identities derived above we write

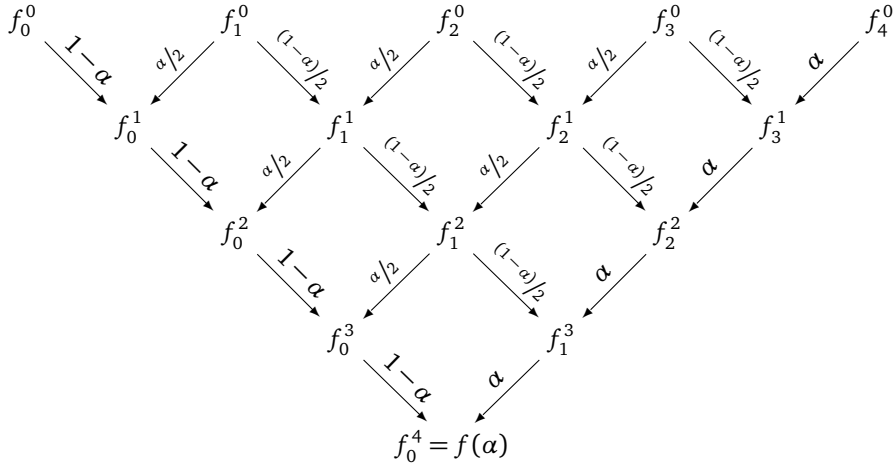
$$\begin{aligned}
f(\alpha) &= f_0^0 \cdot b_0^n(\alpha) + f_1^0 \cdot b_1^n(\alpha) + \cdots + f_{n-1}^0 \cdot b_{n-1}^n(\alpha) + f_n^0 \cdot b_n^n(\alpha) \\
&= f_0^0 \cdot (1-\alpha) \cdot b_0^{n-1}(\alpha) + f_1^0 \cdot \Big(\tfrac{\alpha}{2} \cdot b_0^{n-1}(\alpha) + \tfrac{1-\alpha}{2} \cdot b_1^{n-1}(\alpha) \Big) + \cdots \\
&\qquad \cdots + f_{n-1}^0 \cdot \Big(\tfrac{\alpha}{2} \cdot b_{n-2}^{n-1}(\alpha) + \tfrac{1-\alpha}{2} \cdot b_{n-1}^{n-1}(\alpha) \Big) + f_n^0 \cdot \alpha \cdot b_{n-1}^{n-1}(\alpha) \\
&= \Big((1-\alpha) f_0^0 + \tfrac{\alpha}{2} f_1^0 \Big) \cdot b_0^{n-1}(\alpha) + \Big(\tfrac{1-\alpha}{2} f_1^1 + \tfrac{\alpha}{2} f_1^2 \Big) \cdot b_1^{n-1} + \cdots \\
&\qquad \cdots + \Big(\tfrac{1-\alpha}{2} f_{n-2}^0 + \tfrac{\alpha}{2} f_{n-1}^0 \Big) \cdot b_{n-2}^{n-1} + \Big(\tfrac{1-\alpha}{2} f_{n-1}^0 + \alpha f_d^0 \Big) \cdot b_{n-1}^{n-1}(\alpha) \\
&= f_0^1 \cdot b_0^{n-1}(\alpha) + \cdots + f_{n-1}^1 \cdot b_0^{n-1}(\alpha) \\
&\ \ \vdots \\
&= f_0^k \cdot b_0^{n-k}(\alpha) + \cdots + f_{n-k}^1 \cdot b_0^{n-k}(\alpha) \\
&\ \ \vdots \\
&= f_0^n \cdot b_0^0(\alpha) = f_0^n,
\end{aligned}$$

where the coefficients f_i^k for $k \in \{1, \ldots, n\}$ and $i \in \{0, \ldots, n-k\}$ are given by the formula

$$f_i^k = \begin{cases} (1-\alpha) \cdot f_0^{k-1} + \frac{1}{2} \cdot \alpha \cdot f_1^{k-1} & \text{for } i = 0, \\ \frac{1}{2} \cdot \Big((1-\alpha) \cdot f_i^{k-1} + \alpha \cdot f_{i+1}^{k-1} \Big) & \text{for } 0 < i < n-k, \\ \frac{1}{2} \cdot (1-\alpha) \cdot f_{n-k}^{k-1} + \alpha \cdot f_{n-k+1}^{k-1} & \text{for } i = n-k. \end{cases}$$

All in all, the procedure of calculating $f(\alpha)$ can be depicted in form of a triangular diagram, where each element is computed as a linear combination of its two parents.

Here we present it for $n = 4$:



Algorithm 1.A.23 (de Casteljau, 1959). *Let $f = f_0 b_0^n + \cdots + f_n b_n^n$ be a polynomial in the scaled Bernstein-basis representation and α be an argument. This algorithm computes $f(\alpha)$.*

1. *Denote $f_0^0 := f_0, \ldots, f_d^0 := f_n$.*
2. *For every $k = 1, 2, \ldots, n-1$ compute the next row of the diagram setting*

$$f_0^k := (1-\alpha)\cdot f_0^{k-1} + \frac{1}{2}\cdot\alpha\cdot f_1^{k-1},$$
$$f_j^k := \frac{1}{2}\cdot\left((1-\alpha)\cdot f_j^{k-1} + \alpha\cdot f_{j+1}^{k-1}\right), \qquad \text{for} \quad 0 < j < d-k$$
$$f_{d-k}^k := \frac{1}{2}\cdot(1-\alpha)\cdot f_{d-k}^{k-1} + \alpha\cdot f_{d-k+1}^{k-1}.$$

3. *Output $(1-\alpha)\cdot f_0^{n-1} + \alpha\cdot f_1^{n-1}$.*

The correctness of the algorithm follows from the discussion preceding it. Let us count the number of arithmetic operations used by the algorithm. Observe that all divisions in the algorithm are divisions by 2, which are harmless for binary representations of coefficients. In order to compute $f(\alpha) = f_0^n$, we construct n rows (the initial row, consisting of the coefficients of the polynomial is given). When constructing the row $f_0^k, \ldots, f_{n-k}^k$ we perform $n-k+1$ additions and $2\cdot(n-k+1)$ multiplications. Hence the total number of operations is: $n\cdot(n+1)/2$ additions and twice as many multiplications. Consequently, de Casteljau algorithm is slower than Horner scheme. Yet it is still used from time to time due to its superior numerical stability (see). missing

Remark 19. There is a different and better know version of de Casteljau's algorithm that works with "standard" (i.e unscaled) Bernstein polynomials. It is also a bit more elegant as the corresponding formula for computing f_j^k read as

$$f_j^k = (1-\alpha)\cdot f_j^{k-1} + \alpha\cdot f_{j+1}^{k-1}$$

for every index j without any special cases.

Example 19. This example demonstrates the superiority of de Casteljau algorithm when using floating point numbers. Take a polynomial $f = (x-\frac{9}{10})^2 \cdot (x-\frac{7}{10})^3$ and evaluate it at $\alpha = \frac{4}{5}$. In a traditional form and using exact coefficients we have

$$f = x^5 - \frac{39}{10}x^4 + \frac{303}{50}x^3 - \frac{469}{100}x^2 + \frac{18081}{10000}x - \frac{27783}{100000}.$$

The exact value of $f(\alpha)$ is $\frac{1}{100000}$.

Now suppose that we are using floating point numbers with 5 digits (to emphasize the outcome) at base $\beta = 2$. Hence the machine epsilon is $\varepsilon = \frac{1}{16}$. Converting to floating point numbers and evaluating the resulting polynomial at α using Horner scheme we obtain $f(\alpha) \approx -1.0000 \times 2^{-5}$. This result has a relative error of 48128ε.

On the other hand, in the scaled Bernstein-basis representation the polynomial f has a form

$$f = -\frac{27783}{100000} \cdot b_0^5 + \frac{8379}{20000} \cdot b_1^5 - \frac{2359}{10000} \cdot b_2^5 + \frac{603}{10000} \cdot b_3^5 - \frac{27}{4000} \cdot b_4^5 + \frac{27}{100000} \cdot b_5^5.$$

again converting to floating point numbers and computing $f(\alpha)$ using de Casteljau's algorithm one gets

$$f(\alpha) \approx 1.0000 \times 2^{-16}$$

with the relative error equal 9ε.

The reader should be warned that the previous example is exaggerated. It is intentionally made up to emphasize the differences between the algorithms. The figures are so extreme due to a ridiculously low number of digits used. In real life situations the differences will never be so drastic. Furthermore, for any method it is possible to construct a polynomial f and argument α such that the selected algorithm will produce the most accurate result.

Informally speaking, scaled Bernstein-basis "spreads" information evenly over all coefficients because the corresponding monomials have equal degrees. Conversely, in power-basis representation information is more "concentrated" in leading coefficients due to amplification effect of high degree exponentiation. Consequently, as a rule of thumb one may expect that in vicinity of a root of a polynomial Algorithms 1.A.22 and 1.A.23 are likely to give more accurate results. On the other hand, far away from the roots the leading term of the polynomial becomes to dominate and there is a good chance that Horner scheme in the power-basis representation will beat the other methods.

It is a well known property of the power-basis representation that the value of a polynomial $f = f_d x^d + \cdots + f_1 x + f_0$ at zero is just its constant term f_0. In case of the scaled Bernstein basis we can retrieve two values of f from its list of coefficients.

Observation 1.A.24. *If $f = f_0 b_0^d + \cdots + f_d b_d^d$ is a polynomial in the scaled Bernstein-basis representation, then*

- $f(0) = f_0$;
- $f(1) = f_d$.

Further reading

The most important source offering an in-depth discussion of subjects covered in this chapter is undoubtedly the volume 2 of Donald Knuth's opus magnum [42]. Theorem 1.1.32 and Algorithm 1.1.33, 1.1.34 are based on [77]. Algorithm 1.3.21 was presented in [81]. Various variations of the binary gcd algorithm are discussed in [83], including variants that work in quadratic number fields.

2

Modular arithmetic

2.1 Basic modular arithmetic

When working with integers or polynomials, one often encounters a problem of a rapid growth of sizes of intermediate results. Thus, to keep them on a manageable level, one has to relay on specially crafted algorithms. Algorithm 1.3.17 is a good example here. Surprisingly, in some cases it is possible to completely avoid this obstacle, translating the computational task from the ring of integers/polynomials to *quotient* rings/fields. This comes of course with a price. When solving a problem in a quotient ring, we obtain an answer only in that ring. Most of the time it is not what we want. Hence, we have to subsequently recover the proper answer from one or more its *residuals*. This is the subject of this section.

Take any positive integer $m > 1$. The set $\{0, 1, \ldots, m-1\}$ contains all possible remainders modulo m. Denote this set by $\mathbb{Z}/m\mathbb{Z}$. We can make it into a ring using the following well known properties of congruences:

Proposition 2.1.1. *For any fixed $m > 1$ and any $a, b, c, d \in \mathbb{Z}$, if $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$, then*

$$a + c \equiv b + d \pmod{m}, \qquad a - c \equiv b - d \pmod{m}, \qquad a \cdot c \equiv b \cdot d \pmod{m}.$$

Proof. We will prove only the first congruence, since all others are shown in the same way. By assumption, m divides $a - b$ and $c - d$. Say

$$a - b = k \cdot m \qquad \text{and} \qquad c - d = l \cdot m,$$

for some $k, l \in \mathbb{Z}$. Adding the equation we obtain:

$$(a + c) - (b + d) = (k \cdot l \cdot m) \cdot m$$

and it follows that $a + c \equiv b + d$. □

Corollary 2.1.2. *For every $m > 1$, the set ${}^{\mathbb{Z}}/_{m\mathbb{Z}}$ with operations*

$$a \oplus b := (a + b \bmod m), \qquad a \odot b := (a \cdot b \bmod m)$$

is a ring.

Hereafter, we will drop the circles and denote the operations on remainders using standard arithmetic symbols. The previous corollary is a special case of a more general phenomenon. Recall that an *ideal* of a ring R is a subset $I \subseteq R$ such that

- $a \pm b \in I$ for all $a, b \in I$,
- $a \cdot \beta \in I$ for every $a \in I$ and $\beta \in R$.

The set $m\mathbb{Z} = \{mk \mid k \in \mathbb{Z}\} = \{n \in \mathbb{Z} \mid m \text{ divides } n\}$ is a classic example of an ideal (in the ring of integers).

Proposition 2.1.3. *If I is an ideal in a ring R, then the set ${}^{R}/_{I} := \{a + I \mid a \in R\}$ of cosets modulo I, can be equipped with a ring structure, where addition and multiplication are defined by the formulas:*

$$(a + I) + (b + I) := (a + b) + I \qquad \text{and} \qquad (a + I) \cdot (b + I) := (a \cdot b) + I.$$

The ring ${}^{R}/_{I}$ is called the *quotient ring* of R modulo I. We omit the proof of the proposition, as this result is taught on every basic course of algebra. It follows that the quotient ring ${}^{\mathbb{Z}}/_{m\mathbb{Z}}$ consists in fact of cosets modulo the ideal $m\mathbb{Z}$. Earlier we decided to represent each coset by its minimal non-negative element, which resulted in ${}^{\mathbb{Z}}/_{m\mathbb{Z}}$ being identified with the set $\{0, \ldots, m-1\}$. This is called the *non-negative representation* of the quotient ring ${}^{\mathbb{Z}}/_{m\mathbb{Z}}$. Another natural option, co called the *symmetric representation*, is to take element of minimal absolute value from each coset. This way we identify ${}^{\mathbb{Z}}/_{m\mathbb{Z}}$ (for $m > 2$) with a set $\left\{-\frac{m-1}{2}, \ldots, -1, 0, 1, \ldots, \frac{m-1}{2}\right\}$.

Example 20. We have

$$(9 + 7) \cdot (10 + 11) \equiv 3 \cdot 8 \equiv 9 \equiv -2 \pmod{13}.$$

Thus, $(9+7)\cdot(10+4)$ is congruent to both 9 and -2 modulo 13 or it actually **equals** 9 in the quotient ring ${}^{\mathbb{Z}}/_{13\cdot\mathbb{Z}}$, when using the non-negative representation and equals -2 in the symmetric representation.

Proposition 2.1.1 covers three of four basic arithmetic operations. It is so because division modulo m is a bit trickier and not every element is invertible in ${}^{\mathbb{Z}}/_{m\mathbb{Z}}$.

Proposition 2.1.4. *Let $m > 1$ be a positive integer. An integer k is invertible modulo m if and only if m and k are relatively prime. If it is so, then the inverse of $(k \bmod m)$ in ${}^{\mathbb{Z}}/_{m\mathbb{Z}}$ is a remainder of an integer l satisfying*

$$1 = \gcd(m, k) = m \cdot n + k \cdot l.$$

Proof. If m and k are relatively prime, then Bézout's identity (see Theorem 1.3.4) asserts that there are n, l such that

$$m \cdot n + k \cdot l = \gcd(m, k) = 1.$$

Taking the remainders modulo m of both sides, we have $k \cdot l \equiv 1 \pmod{m}$. Conversely, if k is invertible modulo m, then there is $l \in \mathbb{Z}$ with a property that $k \cdot l \equiv 1 \pmod{m}$. Thus, $kl - 1 = mn$ for some $n \in \mathbb{Z}$. Suppose that $g := \gcd(m, k)$, then g divides $kl - mn = 1$. It follows that m, k are relatively prime. □

In terms of Section 1.3 of Chapter 1, the integer l is a Bézout coefficient. In particular, the inverse of k is computable using extended Euclidean algorithm.

Example 21. In the quotient ring $\mathbb{Z}/_{11 \cdot \mathbb{Z}}$ compute

$$\frac{3 + 7 \cdot 4}{5 + 2 \cdot 6} = \frac{3 + 6}{5 + 1} = 9 \cdot 6^{-1}.$$

Using extended Euclidean algorithm one finds the Bézout coefficients

$$1 = \gcd(11, 6) = (-1) \cdot 11 + 2 \cdot 6.$$

Hence $6^{-1} = 2$ in $\mathbb{Z}/_{11 \cdot \mathbb{Z}}$ and the original expression evaluates to $9 \cdot 6^{-1} = 9 \cdot 2 = 7$.

If p is a prime number, then every positive integer $k < p$ is relatively prime to p. Thus all nonzero elements of $\mathbb{Z}/_{p\mathbb{Z}}$ are invertible. On the other hand, if m factors into $m = k \cdot n$ with $k, n < m$, then in $\mathbb{Z}/_{m\mathbb{Z}}$ the product $k \cdot n$ evaluates to zero and so k, n are *zero divisors*. In particular they are non-invertible. All in all, we have proved:

Corollary 2.1.5. *The quotient ring $\mathbb{Z}/_{m\mathbb{Z}}$ is a field if and only if m is a prime.*

The field $\mathbb{Z}/_{p\mathbb{Z}}$ for p prime is denoted $\mathbb{F}_p$. The above corollary generalizes to arbitrary rings. An ideal $I \subset R$ is *maximal* if there is no ideal J satisfying $I \subsetneq J \subsetneq R$.

Proposition 2.1.6. *Let R be a ring and $I \subset R$ its ideal. The following conditions are equivalent:*

1. *I is a maximal ideal,*
2. *the quotient ring $R/_I$ is a field.*

We again omit a proof referring a reader to any textbook on algebra (see e.g. [51, Chapter II, §2]). It can be proved that an ideal $I = a \cdot R$ in a unique factorization domain R is maximal if and only if it is generated by a prime element. this in particular implies the following fact that we will use in the next section.

Corollary 2.1.7. *If $f \in K[x]$ is an irreducible polynomial (over a field), then the quotient ring $K[x]/_{f \cdot K[x]}$ is a field.*

2.2 Chinese remainder theorem

Before we begin a new subject, imagine that we play a game in which the goal is to guess a number (integer) between 0 and 100. We are only allowed to ask for the remainders modulo small (say single-digit) numbers. How many queries do we need before we are able to pin the number down? It turns out that all we need to know are just three remainders. Every integer $n \in \{0, \ldots, 100\}$ is uniquely determined by its remainders modulo 5, 7 and 9. For example $n = 41$ is the only positive integer not exceeding 100 subject to relations:

$$n \equiv 1 \pmod 5, \qquad n \equiv 6 \pmod 7, \qquad n \equiv 5 \pmod 9.$$

An explanation is provided by the following theorem, dating back to 13th century.

Theorem 2.2.1 (Chinese remainder theorem)**.** *Let $m_1, \ldots, m_k$ be positive integers. Assume they are pairwise relatively prime and denote $M := m_1 \cdots m_k$. For every $a_1, \ldots, a_k \in \mathbb{Z}$, the system of congruences*

$$\begin{cases} a \equiv a_1 \pmod{m_1} \\ \quad \vdots \\ a \equiv a_k \pmod{m_k} \end{cases}$$

has a unique solution a in the set $\{0, \ldots, M-1\}$.

We will prove this theorem in a greater generality replacing the ring of integers by an arbitrary Euclidean domain (see Definition 1.3.1). Namely we will prove:

Theorem 2.2.2 (Generalized Chinese remainder theorem)**.** *Let $m_1, \ldots, m_k$ be pairwise relatively prime elements of an Euclidean domain R. For every $a_1, \ldots, a_k \in R$ the system of congruences*

$$\begin{cases} a \equiv a_1 \pmod{m_1} \\ \quad \vdots \\ a \equiv a_k \pmod{m_k}. \end{cases}$$

has a solution. Moreover if a, b are two such solutions, then $a \equiv b \pmod M$, where $M := m_1 \cdots m_k$.

Proof. For every $i \neq j$, the elements m_i and m_j are relatively prime by assumption. Bézout's identity (Theorem 1.3.4) asserts that there are such elements $\alpha_{ij} \in R$ for $i, j \in \{1, \ldots, k\}$ that

$$\alpha_{ij} m_i + \alpha_{ji} m_j = 1 = \gcd(m_i, m_j).$$

Construct auxiliary elements $A_1, \ldots, A_k$ by the formula

$$A_i := \prod_{\substack{j=1 \\ j \neq i}}^{k} \alpha_{ji} m_j \qquad \text{for} \quad i \in \{1, \ldots, k\}.$$

It is clear that $A_i \equiv 0 \pmod{m_j}$ for every $j \neq i$. On the other hand

$$A_i = \prod_{\substack{j=1 \\ j\neq i}}^{k} (1 - \alpha_{ij} m_i) \equiv 1 \pmod{m_i}.$$

Take now a linear combination of these A_i's with the desired residues $a := a_1 A_1 + \cdots + a_k A_k$, then $a \equiv a_i \pmod{m_i}$ for every $i \in \{1, \ldots, k\}$. Hence, a is our sought solution.

It remains to prove the uniqueness of the solution modulo M. Suppose that there are $a, b \in R$ such that $a \equiv a_i \equiv b \pmod{m_i}$ for every i. Thus, the difference $a - b$ is divisible by every m_i. The elements m_i are relatively prime, hence $a - b$ is divisible by their product, as claimed. □

The proof of Theorem 2.2.2 is constructive and as such it can be directly converted into an algorithm.

Algorithm 2.2.3. *Let R be an Euclidean domain (in which remainders are computable). Given a pairwise relatively prime moduli $m_1, \ldots, m_k \in R$ and arbitrary elements $a_1, \ldots, a_k \in R$, this algorithm constructs $a \in R$ such that $a \equiv a_i \pmod{m_i}$ for every $i \in \{1, \ldots, k\}$.*

1. *Iterate over all $i \in \{1, \ldots, k\}$ and $j \in \{i+1, \ldots, k\}$, for each pair (i, j) execute extended Euclidean algorithm to find Bézout's coefficients α_{ij}, α_{ji} so that $1 = \gcd(m_i, m_j) = \alpha_{ij} m_i + \alpha_{ji} m_j$.*
2. *for every $i \in \{1, \ldots, k\}$, set*
$$A_i := \prod_{\substack{j=1 \\ j\neq i}}^{k} \alpha_{ji} m_j;$$
3. *construct a linear combination*
$$a := \sum_{i=1}^{k} a_i A_i;$$
4. *return a remainder $a \bmod (m_1 \cdots m_k)$.*

Remark 20. One should compare the above construction of $a \in R$ with Lagrange interpolation formula in Proposition 4.1.3. The similarities are not accidental.

Example 22. To see how the above algorithm works, consider the system of congruences from the opening paragraph:

$$a \equiv 1 \pmod{5}, \qquad a \equiv 6 \pmod{7}, \qquad a \equiv 5 \pmod{9}.$$

The moduli $5, 7, 9$ are prime and distinct, hence in particular they are pairwise relatively prime. Compute the Bézout's coefficients:

$$\begin{aligned} 3 \cdot 5 + (-2) \cdot 7 &= 1, \\ 2 \cdot 5 + (-1) \cdot 9 &= 1, \\ 4 \cdot 7 + (-3) \cdot 9 &= 1. \end{aligned}$$

Consequently, the products A_1, A_2, A_3 in the second step of the algorithm are equal:

$$\begin{aligned} A_1 &= (-2)\cdot 7\cdot(-1)\cdot 9 = 126,\\ A_2 &= 3\cdot 5\cdot(-3)\cdot 9 = -405,\\ A_3 &= 2\cdot 5\cdot 4\cdot 7 = 280. \end{aligned}$$

It follows that $a = 1\cdot 126+6\cdot(-405)+5\cdot 280 = -904$ and so the answer is $(-904) \bmod 315 = 41$.

The next few paragraphs require some familiarity with basic notions of algebra. The reader may wish to consult Appendix A. Recall that the intersection of ideals is again an ideal and the sum $\mathfrak{a}+\mathfrak{b}$ of two ideal $\mathfrak{a}, \mathfrak{b}$ is defined to be the ideal generated by all the sums of the form $a+b$, where $a \in \mathfrak{a}$ and $b \in \mathfrak{b}$. Chinese remainder theorem may be expressed in a more abstract form using generalizing the notion of relatively prime elements to relatively prime ideals. If two integers, say m and n, are relatively prime, then Bézout identity asserts that there are $a, b \in \mathbb{Z}$ such that $ma+nb = 1$. The equality still holds, when we multiply both sides by any integer, hence it leads to an equality of ideals $m\mathbb{Z}+n\mathbb{Z} = \mathbb{Z}$. The last condition can be generalized to ideals (not necessarily principal) in an arbitrary ring.

Definition 2.2.4. Two ideal $\mathfrak{a}, \mathfrak{b}$ of a ring R are said to be *relatively prime*, if $\mathfrak{a}+\mathfrak{b} = R$.

Theorem 2.2.5 (Abstract Chinese remainder theorem)**.** *If $\mathfrak{a}_1, \ldots, \mathfrak{a}_k$ are pairwise relatively prime ideals in a ring R, then the quotient ring ${}^{R}/_{(\mathfrak{a}_1\cap\ldots\cap\mathfrak{a}_k)}$ is isomorphic to a Cartesian product ${}^{R}/_{\mathfrak{a}_1}\times\cdots\times{}^{R}/_{\mathfrak{a}_k}$.*

Proof. The proof is nothing else but a recap of the proof of Theorem 2.2.2. Define a map $\varphi : R \to {}^{R}/_{\mathfrak{a}_1}\times{}^{R}/_{\mathfrak{a}_k}$, setting $\varphi(x) := (x+\mathfrak{a}_1, \ldots, x+\mathfrak{a}_k)$. It is clear that φ is a ring homomorphism. Let us compute its kernel. We have $\varphi(x) = (0, \ldots, 0)$ if and only if $x \in \mathfrak{a}_i$ for every $1 \le i \le k$. This means that

$$\ker\varphi = \mathfrak{a}_1 \cap \ldots \cap \mathfrak{a}_k.$$

Next, we show that φ is surjective. To this end fix a tuple $(a_1+\mathfrak{a}_1, \ldots, a_k+\mathfrak{a}_k)$ in the co-domain. We know that $\mathfrak{a}_i+\mathfrak{a}_j = R$ for every $i \neq j$. Thus, there are elements $\alpha_{ij} \in \mathfrak{a}_1$ and $\alpha_{ji} \in \mathfrak{a}_j$ such that $\alpha_{ij}+\alpha_{ji} = 1$. As in the proof of Theorem 2.2.2, construct elements $A_1, \ldots, A_k$ putting

$$A_i := \prod_{\substack{j=1\\ j\neq i}}^{k} \alpha_{ji}.$$

It is clear that A_i lays in all ideals $\mathfrak{a}_1, \ldots, \mathfrak{a}_k$ except $\mathfrak{a}_i$. Take a linear combination $a := a_1\cdot A_1 + \cdots + a_k\cdot A_k$. Since all the summands, but $a_i\cdot A_i$, vanish in the quotient ring ${}^{R}/_{\mathfrak{a}_i}$, we have $\varphi(a) = (a_1+\mathfrak{a}_1, \ldots, a_k+\mathfrak{a}_k)$. This proves our claim. It follows from the isomorphism theorem that ${}^{R}/_{\ker\varphi}$ is isomorphic to $\operatorname{im}\varphi = {}^{R}/_{\mathfrak{a}_1}\times\cdots\times{}^{R}/_{\mathfrak{a}_k}$. □

Algorithm 2.2.3 is a direct implementation of a (constructive) proof of Theorem 2.2.2. Yet, it is hardly optimal as it poses two major problems. First of all, the products $A_1, \dots A_k$ computed in step 2 may easily become unhandleably large if we have to deal with too many moduli. Secondly, there is no easy way to incrementally "improve" the obtained solutions. Suppose that we have solved a system of congruences

$$\begin{cases} a \equiv a_1 \pmod{m_1} \\ \quad \vdots \\ a \equiv a_k \pmod{m_k}, \end{cases}$$

but later we realized that a is a subject to one more relation

$$a \equiv a_{k+1} \pmod{m_{k+1}}.$$

Using Algorithm 2.2.3, we need to recompute nearly everything from scratch. We may keep the calculated α_{ij}'s and we know that the products $A_1, \dots, A_k$ are factors of new products. This is all what we can reuse and is not much. Therefore, our aim is to find an algorithm that let us solve systems of congruences one-by-one, starting from a single pair of congruences and incrementally refining the solution. We begin with a border case of two congruences.

Algorithm 2.2.6. *Let R be a computable Euclidean domain. Given two relatively prime moduli $m_1, m_2 \in R$ and corresponding elements $a_1, a_2 \in R$, this algorithm constructs $a \in R$ such that $a \equiv a_1 \pmod{m_1}$ and $a \equiv a_2 \pmod{m_2}$.*

1. *Using extended Euclidean algorithm, find the Bézout's coefficients α, β so that $\alpha m_1 + \beta m_2 = 1$;*
2. *let r_1 be the remainder of a_1 modulo m_1;*
3. *set $b := (a_2 - r_1) \cdot \alpha$ and let r_2 be the remainder of b modulo m_2;*
4. *return $a = r_1 + r_2 \cdot m_1$.*

Proof of correctness. On one hand we have

$$a = r_1 + r_2 \cdot m_1 \equiv r_1 \equiv a_1 \pmod{m_1}.$$

On the other hand

$$a = r_1 + r_2 \cdot m_1 \equiv r_1 + b \cdot m_1 \pmod{m_2}.$$

Now, $\alpha m_1 + \beta m_2 = 1$, hence $\alpha m_1 \equiv 1 \pmod{m_2}$ and it follows that

$$a \equiv r_1 + (a_2 - r_1) \cdot \alpha \cdot m_1 \equiv r_1 + a_2 - r_1 \pmod{m_2}.$$

Thus, $a \equiv a_2 \pmod{m_2}$ as expected. □

Now, when we know how to solve a system of two congruences, we can deal with a system of an arbitrary size. The following observation is a trivial consequence of the fact that every Euclidean ring is a unique factorization domain, which means that every non-invertible element uniquely factors into primes.

Observation 2.2.7. *If $m_1, \ldots, m_k \in R$ are pairwise relatively prime, then m_k is relatively prime to a product $m_1 \cdots m_{k-1}$.*

The key to an incremental method of solving systems of congruences is provided by the following lemma:

Lemma 2.2.8. *Let $m_1, m_2 \in R$ be two relatively prime moduli and $a_1, a_2 \in R$ two arbitrary elements. The following conditions are then equivalent:*

1. $a_1 \equiv a_2 \pmod{m_1}$ *and* $a_1 \equiv a_2 \pmod{m_2}$,
2. $a_1 \equiv a_2 \pmod{m_1 \cdot m_2}$.

Proof. Set $d := a_1 - a_2$. Assume the first condition, then d is divisible by both m_1 and m_2. But m_1 and m_2 are relatively prime, hence d is divisible by their product. This proves the second condition. Conversely, if d is divisible by $m_1 \cdot m_2$, then it is clearly divisible by both m_1 and m_2. Thus, (2) trivially implies (1). □

Corollary 2.2.9. *A system of congruences*

$$a \equiv a_1 \pmod{m_1}, \quad a \equiv a_2 \pmod{m_2} \quad \ldots \quad a \equiv a_k \pmod{m_k}$$

is equivalent to a system

$$a \equiv b_1 \pmod{m_1 \cdot m_2}, \quad a \equiv a_3 \pmod{m_3} \quad \ldots \quad a \equiv a_k \pmod{m_k},$$

where

$$b_1 \equiv a_1 \pmod{m_1} \qquad \text{and} \qquad b_1 \equiv a_2 \pmod{m_2}.$$

The previous corollary lets us now write another algorithm for solving Chinese remainder problem, that is sometimes accredited to I. Newton[1].

Algorithm 2.2.10. *Let R be a computable Euclidean domain. Given pairwise relatively prime moduli $m_1, \ldots, m_k \in R$ and arbitrary elements $a_1, \ldots, a_k \in R$, this algorithm constructs $a \in R$ such that $a \equiv a_i \pmod{m_i}$ for every $i \in \{1, \ldots, k\}$.*

1. *Initialize $M := m_1$ and $a := (a_1 \bmod m_1)$;*
2. *for every $i \in \{2, \ldots, k\}$ do:*
   a) *execute Algorithm 2.2.6 to find an element $b \in R$ such that*
   $$b \equiv a \pmod{M} \qquad \text{and} \qquad b \equiv a_i \pmod{m_i};$$
   b) *replace a by b and M by a product $M \cdot m_i$;*
3. *return a.*

[1] Sir Isaac Newton (born 4 January 1643, died 31 March 1727) was an English physicist, mathematician and astronomer. He hold the rank of President of the Royal Society of London.

Example 23. Consider the same system of congruences as before:

$$a \equiv 1 \pmod 5, \qquad a \equiv 6 \pmod 7, \qquad a \equiv 5 \pmod 9.$$

We begin with $a = 1$ and $M = 5$. Use Algorithm 2.2.6 to solve a system of two congruences:

$$a \equiv 1 \pmod 5, \qquad a \equiv 6 \pmod 7.$$

The solution is $a = 6$. Replace M by $M \cdot m_2 = 35$ and take the next two congruences:

$$a \equiv 6 \pmod{35}, \qquad a \equiv 5 \pmod 9.$$

Using Algorithm 2.2.6 again, we obtain the desired solution $a = 41$ to our original system.

2.3 Modular exponentiation

Consider a problem of computing a remainder modulo $m \in \mathbb{N}$ of a power a^n for some $a \in \mathbb{Z}$ and $n \in \mathbb{N}$. It is a kind of a problem often encountered in practice. A naïve approach of computing first $a^n \in \mathbb{Z}$ and then dividing the result by m is the worst possible method. Far better strategy is to adapt Algorithm 1.1.10 to our new setup, reducing all intermediate values to the quotient ring. The following pseudo-code is a self-explanatory modification of the algorithm mentioned above.

Algorithm 2.3.1. *Given an integer $a \in \mathbb{Z}$, exponent $n > 1$ and modulus $m > 1$, this algorithm computes the remainder of a^n modulo m.*

1. *Let $(d_k \dots d_0)_2$ be the binary representation of the exponent n;*
2. *initialize $r := 1$ and $b := (a \bmod m)$;*
3. *for each $i \in \{0, \dots, k\}$ do:*
   a) *if $d_i = 1$, then update r setting $r := (r \cdot b \bmod m)$;*
   b) *replace b by the remainder modulo m of the square b^2;*
4. *return r.*

Example 24. Let us compute $49^{19} \bmod 11$. We have $19 = (10011)_2$. The following table presents successive steps of the above algorithm

d_i	b	r
—	$5 = (49 \bmod 11)$	1
1	$3 = (5^2 \bmod 11)$	$5 = (1 \cdot 5 \bmod 11)$
1	$9 = (3^2 \bmod 11)$	$4 = (5 \cdot 3 \bmod 11)$
0	$4 = (9^2 \bmod 11)$	4
0	$5 = (4^2 \bmod 11)$	4
1	$3 = (5^2 \bmod 11)$	$9 = (4 \cdot 5 \bmod 11)$

It follows that $49^{19} \equiv 9 \pmod{11}$.

Remark 21. Observe that the sequence of square remainders that we computed above is cyclic. Indeed, starting from 5, we computed $3, 9, 4$ and returned back to 5. It is always like this. For every $a \in \mathbb{Z}/m\mathbb{Z}$ the sequence $a, a^2, a^4, \ldots$ is cyclic, except possibly the first element, and the cycle length does not exceed $\lceil m/2 \rceil$. An easy proof of this fact is left to the reader. This phenomenon can be used to speed up Algorithm 2.3.1. We will not deal with details here, because we shall present another, much faster method for modular powering.

Theorem 2.3.2 (Fermat's[2] little theorem, 1640)**.** *If p is a prime number, then for every $a \in \mathbb{Z}$:*

$$a^{p-1} \equiv \begin{cases} 0 \pmod{p}, & \text{if } p \mid a, \\ 1 \pmod{p}, & \text{if } p \nmid a. \end{cases}$$

Proof. If p divides a, then it is clear that all powers of a are congruent to 0 modulo p. Assume that a is not divisible by p. By Proposition 2.1.4 the element a is invertible modulo p. It implies that $a \cdot \mathbb{F}_p := \{0, a \bmod p, \ldots, (p-1) \cdot a \bmod p\}$ is the set of all possible remainders modulo p. Thus the sets $a \cdot \mathbb{F}_p$ and $\mathbb{F}_p$ are equal. The products of both sides (in the ring of integers) must be congruent modulo p:

$$\prod\left(a \cdot \mathbb{F}_p\right) = a^p \cdot \prod\left(\mathbb{F}_p\right) = a^p \cdot (p-1)! \equiv (p-1)! \pmod{p}.$$

Now, p being a prime number is relatively prime to $(p-1)!$ and so $(p-1)!$ is invertible modulo p. Consequently $a^p \equiv 1 \pmod{p}$, as claimed. □

Example 25. Let us again calculate $49^{19} \bmod 11$, but this time we will use Fermat's little theorem. We have

$$49^{19} \equiv 5^{19} \equiv 5^{10} \cdot 5^9 \equiv 1 \cdot 5^9 \equiv 9 \pmod{11}.$$

We will now prove a certain consequence of Fermat's little theorem that will be needed later.

Proposition 2.3.3 (Wilson theorem)**.** *An integer n is a prime if and only if $(n-1)!$ is congruent modulo n to -1.*

Proof. First assume that $(n-1)! \equiv -1 \pmod{n}$. This means that n divides $(n-1)! + 1$. Suppose that n is a composite number and let $1 < m < n$ be its proper divisor. Then m also divides $(n-1)! + 1$. On the other hand, m divides the factorial $(n-1)! = 2 \cdots m \cdots (n-1)$. But this is impossible, as $m > 2$ so it cannot divide $(n-1)!$ and $(n-1)! + 1$ at the same time.

Conversely, assume that n is a prime number so that $\mathbb{Z}/n\mathbb{Z} = \mathbb{F}_n$ is a field. Consider a polynomial $f := x^{n-1} - 1 \in \mathbb{F}_n[x]$. Theorem 2.3.2 asserts that $f(a) = 0$ for every

[2] Pierre de Fermat (born in 1607, died on 12 January 1665) was a French lawyer and mathematician.

nonzero $a \in \mathbb{F}_n$. By Bézout's little theorem (Proposition 1.1.18), the polynomial f is divisible by

$$g = \big(x-1\big)\big(x-2\big)\cdots\big(x-(n-1)\big).$$

The two polynomials have the same degree and are both monic, hence they must be equal. In particular their constant terms coincide and we have

$$-1 = f(0) = g(0) = 1\cdot 2\cdots(n-1) = (n-1)!$$

The equality hold in the field $\mathbb{F}_n$, hence for integers we have a congruence $-1 \equiv (n-1)!$ (mod n). □

Exercise 6. Prove the following stronger version of Wilson theorem:

$$(n-1)! \equiv \begin{cases} -1, & \text{when } n \text{ is a prime,} \\ 2, & \text{when } n = 4, \\ 0, & \text{when } n \text{ is composite and } n > 4. \end{cases}$$

We shall generalize Fermat's little theorem to let the modulus be a compound number. Observe that the exponent $p-1$ appearing in Fermat's little theorem is the number of nonzero elements in $\mathbb{F}_p$. Now every nonzero element in $\mathbb{F}_p$ is invertible as $1,\ldots,p-1$ are all relatively prime to p. In general we define:

Definition 2.3.4. *Euler φ-function* is a function that assigns to a positive integer $n > 1$ the number $\varphi(n)$ of elements of the set $\{0,1,\ldots,n-1\}$ that are relatively prime to n.

In other words, $\varphi(n)$ is the number of units in the quotient ring $\mathbb{Z}/n\mathbb{Z}$.

Observation 2.3.5. *If p is a prime, then $\varphi(p) = p-1$.*

Proposition 2.3.6. *If n is a prime power $n = p^k$, where p is a prime, then*

$$\varphi(n) = p^k - p^{k-1} = p^k \cdot \Big(1 - \frac{1}{p}\Big).$$

Proof. Among the numbers $0,1,\ldots,p^k-1$, the ones that are not relatively prime to p^k are precisely those that are divisible by p, namely:

$$0, p, 2p, \ldots, (p-1)\cdot p, p^2, \ldots, (p-1)\cdot p^{k-1}.$$

There are p^{k-1} of them. Hence all the remaining elements are relatively prime to p^k. □

Proposition 2.3.7. *If $m, n \in \mathbb{Z}$ are relatively prime, then*

$$\varphi(m\cdot n) = \varphi(m)\cdot\varphi(n).$$

Proof. We must count the number of integers between 0 and $mn-1$ that are relatively prime to mn. The moduli m, n satisfy $\gcd(m,n) = 1$ and so an integer k is relatively prime to both of them if and only if it is relatively prime to their product by Observation 2.2.7. By definition, $\varphi(m)$ and $\varphi(n)$ are the numbers of in invertibles respectively in $\mathbb{Z}/m\mathbb{Z}$ and $\mathbb{Z}/n\mathbb{Z}$. Thus there is a total of $\varphi(m)\cdot\varphi(n)$ pairs of units (k_1, k_2). This means that k_1, k_2, when treated as integers, satisfy $\gcd(k_1, m) = 1 = \gcd(k_2, n)$. Every such a pair (k_1, k_2) determines a system of congruences

$$k \equiv k_1 \pmod{m} \qquad \text{and} \qquad k \equiv k_2 \pmod{n}.$$

Chinese remainder theorem (Theorem 2.2.1) asserts that this system has a unique solution in $\mathbb{Z}/mn\mathbb{Z}$. Thus, the numbers k relatively prime to mn, being solutions, are in 1-to-1 correspondence with the pairs (k_1, k_2) of the remainders. We have already counted the latter: there are $\varphi(m)\cdot\varphi(n)$ of them. Hence this is also the number of invertibles in $\mathbb{Z}/mn\mathbb{Z}$. □

Remark 22. For those familiar with (basic) ring theory: Theorem 2.2.5 asserts that the quotient ring $\mathbb{Z}/mn\mathbb{Z}$ is isomorphic to the cartesian product $\mathbb{Z}/m\mathbb{Z}\times\mathbb{Z}/n\mathbb{Z}$. It is easy to observe that the units in the latter ring are precisely the pairs of units. The ring isomorphism preserves groups of units and so the group $U(\mathbb{Z}/mn\mathbb{Z})$ is isomorphic to $U(\mathbb{Z}/m\mathbb{Z})\times U(\mathbb{Z}/n\mathbb{Z})$. Counting number of elements of both groups we have

$$\varphi(mn) = \sharp U(\mathbb{Z}/mn\mathbb{Z}) = \sharp U(\mathbb{Z}/m\mathbb{Z})\cdot\sharp U(\mathbb{Z}/n\mathbb{Z}) = \varphi(m)\cdot\varphi(n).$$

We are now ready to present a generalization of little Fermat's theorem that is due to L. Euler[3].

Theorem 2.3.8 (Euler, 1736)**.** *Let $m > 1$ be a positive integer. If $a \in \mathbb{Z}$ is relatively prime to m, then*

$$a^{\varphi(m)} \equiv 1 \pmod{m}.$$

Proof. First assume that m has a unique prime factor, hence $m = p^k$ for a prime number p and some integer k. Then, $\varphi(m) = p^k - p^{k-1}$ by Proposition 2.3.6. We proceed by induction on k. For $k = 1$ the assertion follows from Theorem 2.3.2. Assume that the thesis has been proved for k and any a not divisible by p. Thus, $a^{\varphi(p^k)} = a^{p^k - p^{k-1}} \equiv 1 \pmod{p^k}$ or in another form:

$$a^{p^k - p^{k-1}} = 1 + p^k \cdot b$$

for some integer b. Compute p-th powers of both sides to get

$$a^{p^{k+1} - p^k} = 1 + \binom{p}{1}\cdot p^k \cdot b + \binom{p}{2}\cdot p^{2k}\cdot b^2 + \cdots + \binom{p}{p-1}\cdot p^{k\cdot(p-1)}\cdot b^{p-1} + p^{kp}\cdot b^p.$$

[3]Leonhard Euler (born 15 April 1707, died 18 September 1783) was a Swiss mathematician, physicist, astronomer, logician and engineer.

All the terms on the right-hand-side, except the first one, are divisible by p^{k+1}. Consequently, $a^{p^{k+1}-p^k} \equiv 1 \pmod{p^{k+1}}$ and this concludes the proof in the case when the modulus is a prime power.

We may now show the general case. Assume that $m = p_1^{k_1} \cdots p_n^{k_n}$ is a factorization of the modulus. The previous part let us write a system of congruences

$$\begin{cases} a^{\varphi(p_1^{k_1})} \equiv 1 \pmod{p_1^{k_1}} \\ \quad \vdots \\ a^{\varphi(p_n^{k_n})} \equiv 1 \pmod{p_n^{k_n}}. \end{cases} \tag{2.1}$$

For every $i \in \{1, \ldots, n\}$, raise both sides of the i-th congruence to the power $\frac{\varphi(m)}{\varphi(p_i^{k_i})} = \prod_{j \neq i} \varphi(p_j^{k_j})$ to get:

$$\begin{cases} a^{\varphi(m)} \equiv 1 \pmod{p_1^{k_1}} \\ \quad \vdots \\ a^{\varphi(m)} \equiv 1 \pmod{p_n^{k_n}}. \end{cases}$$

Lemma 2.2.8 asserts now that $a^{\varphi(m)} = 1 \pmod{p_1^{k_1} \cdots p_n^{k_n}}$. □

Remark 23. For those familiar with basic group theory: the set $U(\mathbb{Z}/m\mathbb{Z})$ of invertible elements of $\mathbb{Z}/m\mathbb{Z}$ is a multiplicative group of order $\varphi(m)$. Lagrange's theorem asserts that the order of any element of a group divides the order of the group. Thus for any $a \in U(\mathbb{Z}/m\mathbb{Z})$, if $\operatorname{ord}(a)$ denotes its order, then $\varphi(m) = \operatorname{ord}(a) \cdot k$ for some $k \in \mathbb{Z}$. It follows that $a^{\varphi(m)} = (a^{\operatorname{ord}(a)})^k = 1$.

Example 26. Let us compute 2^{1000} modulo 225. Since $225 = 3^2 \cdot 5^2$, Propositions 2.3.6 and 2.3.7 let us compute the Euler function of the modulus:

$$\varphi\left(3^2 \cdot 5^2\right) = \varphi\left(3^2\right)\varphi\left(5^2\right) = 3^2 \cdot \left(1 - \frac{1}{3}\right) \cdot 5^2 \cdot \left(1 - \frac{1}{5}\right) = 120.$$

Therefore

$$2^{1000} = 2^{8 \cdot 120 + 40} \equiv 2^{40} \equiv 151 \pmod{225}.$$

Theorem 2.3.8 is convenient when we compute modular powers in bulk. We may then pre-compute the Euler function of the modulus and use it many times. On the other hand, when we need to compute just a single modular power or when the modulus changes, one can do better relaying directly on Equation (2.1) from the proof of Euler's theorem.

Algorithm 2.3.9. *Given an integer $a \in \mathbb{Z}$, an exponent $n > 0$ and a modulus m with a known factorization, this algorithm computes the remainder $a^n \bmod m$.*

1. *Let $m = p_1^{k_1} \cdots p_s^{k_s}$ be the factorization of the modulus;*
2. *for every $i \in \{1, \ldots, k\}$ do the following:*
   a) *set $n_i := (n \bmod \varphi(p_i^{k_i}))$, where $\varphi(p_i^{k_i}) = p_i^{k_i} - p_i^{k_i - 1}$ by Proposition 2.3.6;*

b) compute $a_i := (a^{n_i} \bmod p_i^{k_i})$ *using Algorithm 2.3.1;*

3. *using Algorithm 2.2.10 solve the system of congruences*

$$\begin{cases} \alpha \equiv a_i & (\bmod\ p_1^{k_1}) \\ \quad \vdots & \\ \alpha \equiv a_s & (\bmod\ p_s^{k_s}) \end{cases}$$

4. *return the obtained solution* α.

2.4 Modular square roots

Now when we are done with exponentiation, we should deal with a related subject of checking if an element of $\mathbb{Z}/m\mathbb{Z}$ is a square and, if so, finding its square root. Fix a modulus m. An integer $a \in \mathbb{Z}$ is called a *quadratic residue* modulo m, if the remainder of a modulo m is a square in the ring $\mathbb{Z}/m\mathbb{Z}$. Otherwise a is said to be a *quadratic nonresidue*. We will develop method to decide whether a given integer is a quadratic residue modulo a fixed m. To this end, we must first consider prime moduli, subsequently prime-powers and only then we will be ready to deal with compound numbers.

Observe that $\mathbb{F}_2 = \mathbb{Z}/2\mathbb{Z}$ consists of just two elements: classes of even and odd numbers. They are both squares in $\mathbb{F}_2$ since $0^2 = 0$ and $1^2 = 1$. Hence, modulo 2 every integer is a quadratic residue. Therefore for what follows, we will restrict ourselves to odd prime moduli.

Definition 2.4.1. Let $p > 2$ be an odd prime and a be an integer. *Legendre symbol* $\left(\frac{a}{p}\right)$ is defined by the formula:

$$\left(\frac{a}{p}\right) := \begin{cases} 1 & \text{if } a \text{ is quadratic residue modulo } p \\ 0 & \text{if } a \text{ is divisible by } p \\ -1 & \text{if } a \text{ is quadratic nonresidue modulo } p. \end{cases}$$

The following fact is trivial but still worth being formulated explicitly.

Observation 2.4.2. *If* $a \equiv b \pmod p$, *then* $\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$.

We will develop methods for evaluating Legendre symbol. We begin by showing that one may express it using exponentiation. Although the following result gives an explicit formula for Legendre symbol, it is important mostly for its implications and is rarely convenient for actual computations.

Theorem 2.4.3 (Euler's criterion)**.** *If* $p > 2$ *is an odd prime, then for every integer* $a \in \mathbb{Z}$ *the following congruence holds:*

$$\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \pmod p.$$

Proof. If the modulus p divides a then the formula is trivially true as both sides are zero. Thus, without loss of generality we may assume that a is not divisible by p. First assume that a is a quadratic residue modulo p, then $\left(\frac{a}{p}\right) = 1$ and there is b such that $b^2 \equiv a \pmod{p}$. We write

$$a^{\frac{p-1}{2}} \equiv (b^2)^{\frac{p-1}{2}} \equiv b^{p-1} \equiv 1 = \left(\frac{a}{p}\right) \pmod{p}.$$

Here, the last congruence follows from Fermat's little theorem. Thus, the assertion holds in this case.

Conversely, assume that a is a quadratic nonresidue, therefore $\left(\frac{a}{p}\right) = -1$ and an equation $x^2 - a = 0$ has no solution in $\mathbb{F}_p$. Take any integer $b \in \{1, \dots, p-1\}$. Clearly $\gcd(p, b) = 1$ and so the class of b is invertible in $\mathbb{F}_p$. There is $c \in \mathbb{Z}$ such that $bc \equiv 1 \pmod{p}$. Multiplying both sides by a we have $bca \equiv a \pmod{p}$. In other words, for every nonzero $b \in \mathbb{F}_p$, there is $d \in \mathbb{F}_p$ such that $bd = a$. Observe that $b \neq d$ since otherwise a would be a square modulo p.

We know by Proposition 2.3.3 that a factorial $(p-1)!$ is congruent to -1 modulo p. On the other hand, by the previous paragraph, the set $\mathbb{F}_p^* = \{1, \dots, p-1\}$ can be subdivided into $(p-1)/2$ pairs (b_i, d_i) such that a product of each pair is $b_i d_i = a$, $i \in \{1, \dots, (p-1)/2\}$. Thus we have

$$-1 \equiv (p-1)! = \prod_{i=1}^{p-1} i = \prod_{i=1}^{\frac{p-1}{2}} (b_i \cdot d_i) \equiv a^{\frac{p-1}{2}} \pmod{p}. \qquad \square$$

Euler's criterion has a number of useful consequences that will allow us to effectively compute Legendre symbol.

Corollary 2.4.4. *For every odd prime p:*

$$\left(\frac{1}{p}\right) = 1 \qquad \text{and} \qquad \left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}}.$$

Corollary 2.4.5. *For every $a, b \in \mathbb{Z}$ and every odd prime p:*

$$\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right) \cdot \left(\frac{b}{p}\right).$$

Proof. By Euler's criterion, we have

$$\left(\frac{a}{p}\right) \cdot \left(\frac{b}{p}\right) \equiv a^{\frac{p-1}{2}} \cdot b^{\frac{p-1}{2}} = (ab)^{\frac{p-1}{2}} \equiv \left(\frac{ab}{p}\right) \pmod{p}. \qquad \square$$

Proposition 2.4.6. *If p is an odd prime, then 2 is a quadratic residue modulo p if and only $p \equiv \pm 1 \pmod 8$ and nonresidue if and only if $p \equiv \pm 3 \pmod 8$. Equivalently:*

$$\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}.$$

Proof. By assumption, p is odd. Denote $q := {}^{(p-1)}/_{2} \in \mathbb{Z}$. First suppose that $p \equiv 1$ or $-3 \pmod 8$. Using the fact that modulo p we have $p-1 \equiv -1$, $p-3 \equiv -3$,... etc., we may write:

$$\begin{aligned} 2^q \cdot q! &= 2 \cdot 4 \cdot 6 \cdots (p-1) \\ &\equiv 2 \cdot 4 \cdot 6 \cdots \frac{p-1}{2} \cdot \Big(-\frac{p-3}{2}\Big) \cdots (-5) \cdot (-3) \cdot (-1) \\ &\equiv (-1)^{\frac{q}{2}} \cdot q! \pmod p. \end{aligned}$$

Canceling the term $q!$ and applying Euler's criterion we have

$$\left(\frac{2}{p}\right) \equiv 2^q \equiv (-1)^{\frac{q}{2}} \pmod p.$$

It follows that $\left(\frac{2}{p}\right) = 1$ if and only if ${}^{q}/_{2}$ is even and this is equivalent to the condition $p \equiv 1 \pmod 8$.

Now assume that p is congruent to -1 or 3 modulo 8. In a similar fashion we write

$$\begin{aligned} 2^q \cdot q! &= 2 \cdot 4 \cdot 6 \cdots (p-1) \\ &\equiv 2 \cdot 4 \cdot 6 \cdots \frac{p-3}{2} \cdot \Big(-\frac{p-1}{2}\Big) \cdots (-5) \cdot (-3) \cdot (-1) \\ &\equiv (-1)^{\frac{p+1}{4}} \cdot q! \pmod p. \end{aligned}$$

Applying Euler's criterion again, we have

$$\left(\frac{2}{p}\right) \equiv 2^q \equiv (-1)^{\frac{p+1}{4}} \pmod p$$

and so 2 is a quadratic residue when $p \equiv -1 \pmod 8$ and nonresidue when $p \equiv 3 \pmod 8$. □

We may now prove a celebrated quadratic reciprocity law. It was conjectured by L. Euler in 1783 but completely proved only 13 years later by C. Gauß. In fact Gauß discovered eight different proofs (but published only six of them) of this theorem, which he called "aureum theorema" (means golden theorem). Nowadays there are more than two hundred proofs published! The one we present below is due to G. Rousseau.

Theorem 2.4.7 (Quadratic reciprocity law)**.** *Let p, q be two distinct odd primes. Then q is a quadratic residue modulo p if and only if p is a quadratic residue modulo q, except when p and q are simultaneously congruent to 3 modulo 4. In the latter case, q is a quadratic residue modulo p if an only if p is a quadratic nonresidue modulo q. In an abbreviated form:*

$$\left(\frac{q}{p}\right) = \begin{cases} -\left(\frac{p}{q}\right), & \text{if } p \equiv q \equiv 3 \pmod 4, \\ \left(\frac{p}{q}\right), & \text{otherwise.} \end{cases}$$

Proof. Recall that $U(\mathbb{Z}/m\mathbb{Z}) = \{k < m \,|\, \gcd(k,m) = 1\}$ is a (multiplicative) group of units in the quotient ring $\mathbb{Z}/m\mathbb{Z}$. We will write the cosets of a subgroup $\Gamma := \{(1,1),(-1,-1)\}$ of the group $U\mathbb{F}_p \times U\mathbb{F}_q$ in two different ways. One approach is to take a Cartesian product of the first coordinate times the "lower half" of the second coordinate:

$$R := \left\{(a,b) \,|\, a \in U\mathbb{F}_p \text{ and } 0 < b < \frac{q}{2}\right\}.$$

The second approach uses Chinese remainder theorem (Theorem 2.2.1). Write

$$L := \left\{(k \bmod p, k \bmod q) \,|\, k \in U(\mathbb{Z}/pq\mathbb{Z}) \text{ and } k < \frac{pq}{2}\right\}.$$

Theorem 2.2.1 asserts that for every $(a,b) \in R$ there is a unique element $k \in U(\mathbb{Z}/pq\mathbb{Z})$ satisfying

$$k \equiv a \pmod{p} \qquad \text{and} \qquad k \equiv b \pmod{q}.$$

Now, either $k < pq/2$ and so the pair $(k \bmod p, k \bmod q)$ belongs to L or $pq - k < pq/2$. In the latter case $(-k \bmod p, -k \bmod q) \in L$. All in all, we see that with every $(a,b) \in R$ we may associate $k < pq/2$ such that $(a,b) = \pm(k \bmod p, k \bmod q)$. It means that the products of all elements of R and all elements of L are equal to each other up to a sign:

$$\prod_{(a,b)\in R} (a,b) = \varepsilon \cdot \prod_{k \in L} (k,k) \qquad \text{for some } \varepsilon = \pm 1. \tag{2.2}$$

In other words, these product are equal in the quotient group $(U\mathbb{F}_p \times U\mathbb{F}_q)/\Gamma$.

Let us take a closer look at the left-hand-side of Eq. (2.2). For brevity, denote $\overline{p} := (p-1)/2$ and $\overline{q} := (q-1)/2$. Compute

$$\prod_{(a,b)\in R} (a,b) = \prod_{\substack{a<p \\ b<q/2}} (a,b) = \Big((p-1)!^{\overline{q}}, \overline{q}!^{p-1}\Big).$$

Proposition 2.3.3 asserts that the first coordinate evaluates to $(-1)^{\overline{q}}$ in $\mathbb{F}_p$. The second coordinate can be simplified in the following way:

$$\begin{aligned}
(\overline{q}!)^{p-1} &= (\overline{q}!)^{\overline{p}} \cdot (\overline{q}!)^{\overline{p}} \\
&= \big(1 \cdot 2 \cdots \overline{q}\big)^{\overline{p}} \cdot \big((-1)\cdot(-2)\cdots(-\overline{q})\big)^{\overline{p}} \cdot (-1)^{\overline{pq}} \\
&= \big(1 \cdot 2 \cdots \overline{q} \cdot (q-\overline{q}) \cdots (q-2) \cdot (q-1)\big)^{\overline{p}} \cdot (-1)^{\overline{pq}} \\
&= (q-1)!^{\overline{p}} \cdot (-1)^{\overline{pq}} \\
&= (-1)^{\overline{p}} \cdot (-1)^{\overline{pq}},
\end{aligned}$$

where the last equality follows again from Proposition 2.3.3. This means that the left-hand-side of Eq. (2.2) reads as $\big((-1)^{\overline{q}}, (-1)^{\overline{p}(1+\overline{q})}\big)$.

Conversely let us concentrate on the other product. The first coordinate of the right-hand-side of Eq. (2.2) expands into:

$$\prod_{k\in L} k = \frac{\prod\limits_{\substack{k<pq/2\\ p\nmid k}} k}{\prod\limits_{\substack{k<pq/2\\ q\mid k}} k} = \frac{\left(\prod\limits_{0<k<p} k\right)\cdot\left(\prod\limits_{0<k<p}(p+k)\right)\cdots\left(\prod\limits_{0<k<p}\left((\overline{q}-1)\cdot p+k\right)\right)}{(q)(2q)\cdots(\overline{p}q)}\cdot\left(\prod_{0<k<\frac{p}{2}}(\overline{q}p+k)\right)$$

We are working in a finite field $\mathbb{F}_p$ (i.e. modulo p), hence each product in the numerator evaluates to $(p-1)! = -1$ by Proposition 2.3.3. On the other hand, the last product evaluate to $\overline{p}!$ and by Euler's criterion the denominator equals:

$$(q)(2q)\cdots(\overline{p}q) = q^{\overline{p}}\cdot\overline{p}! = \left(\frac{q}{p}\right)\cdot\overline{p}!.$$

All in all the first coordinate of the right-hand-side of Eq (2.2) simplifies to $(-1)^{\overline{q}}\cdot\left(\frac{q}{p}\right)$. By symmetry, the second coordinate gives $(-1)^{\overline{p}}\cdot\left(\frac{p}{q}\right)$.

Putting things together, Eq. (2.2) reads now as

$$\left((-1)^{\overline{q}}, (-1)^{\overline{p}(1+\overline{q})}\right) = \varepsilon\cdot\left((-1)^{\overline{q}}\cdot\left(\frac{q}{p}\right), (-1)^{\overline{p}}\cdot\left(\frac{p}{q}\right)\right).$$

Comparing the respective coordinates we have

$$\varepsilon\cdot\left(\frac{q}{p}\right)\equiv 1 \pmod p \qquad \text{and} \qquad \varepsilon\cdot\left(\frac{p}{q}\right)\equiv(-1)^{\overline{pq}} \pmod q.$$

It follows that $\varepsilon = \left(\frac{q}{p}\right)$ and consequently

$$\left(\frac{p}{q}\right)\cdot\left(\frac{q}{p}\right) = (-1)^{\frac{(p-1)(q-1)}{4}}.$$

This last formula is clearly equivalent to the thesis of the theorem. □

Example 27. Let us check whether $q = 887$ is a quadratic residue modulo $p = 1237$. By the quadratic reciprocity law and Observation 2.4.2 we calculate:

$$\left(\frac{q}{p}\right) = \left(\frac{887}{1237}\right) = \left(\frac{1237}{887}\right) = \left(\frac{350}{887}\right).$$

We have $350 = 2\cdot 5^2\cdot 7$ and using Corollary 2.4.5 we write:

$$\left(\frac{350}{887}\right) = \left(\frac{2}{887}\right)\cdot\left(\frac{5^2}{887}\right)\cdot\left(\frac{7}{887}\right) = \left(\frac{2}{887}\right)\cdot 1\cdot\left(\frac{7}{887}\right) = \left(\frac{2}{887}\right)\cdot\left(\frac{7}{887}\right)$$

By Proposition 2.4.6, the first term equals $\left(\frac{2}{887}\right) = 1$, because $887 \equiv 7 \pmod 8$. Now, calculate the second term using quadratic reciprocity law once again:

$$\left(\frac{7}{887}\right) = -\left(\frac{887}{7}\right) = -\left(\frac{5}{7}\right) = -\left(\frac{7}{5}\right) = -\left(\frac{2}{5}\right) = 1.$$

All in all, $\left(\frac{q}{p}\right) = 1\cdot 1 = 1$, which means that q is a square in $\mathbb{F}_p$.

In the previous example, to compute a Legendre symbol $\left(\frac{350}{887}\right)$, we factored 350 into primes. We will discuss integer factorization in Chapter 5. Anyway, in general, factorization is a time-consuming operation and as such, it should be avoided if possible. Fortunately, when checking whether a number is a square modulo a prime, we can do without factorization (except factoring out a power of 2, which is easy). To this end we generalize the notion of the Legendre symbol to compound moduli.

Definition 2.4.8. Let n be an odd positive integer that factors into primes as $n = p_1^{k_1} \cdots p_s^{k_s}$. For any integer $a \in \mathbb{Z}$, the *Jacobi symbol* $\left(\frac{a}{n}\right)$ is a product of corresponding Legendre symbols:

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{k_1} \cdots \left(\frac{a}{p_s}\right)^{k_s}.$$

Remark 24. Although the Jacobi symbol generalizes the Legendre symbol in a sense that for a prime number n the two notions coincide, it does not catch the basic property of the later. Vanishing of the Jacobi symbol $\left(\frac{a}{n}\right)$ is not a sufficient condition for a to be a quadratic residue modulo n. Take for example $a = 5$ and $n = 21$, then a is not a square in $\mathbb{Z}/n\mathbb{Z}$, yet $\left(\frac{5}{21}\right) = \left(\frac{5}{3}\right) \cdot \left(\frac{5}{7}\right) = (-1) \cdot (-1) = 1$.

It is clear by the way we defined it, that the Jacobi symbol is multiplicative with respect to both arguments.

Observation 2.4.9. *For every m, n and every a, b we have*

$$\left(\frac{a}{mn}\right) = \left(\frac{a}{m}\right) \cdot \left(\frac{a}{n}\right) \qquad \text{and} \qquad \left(\frac{ab}{m}\right) = \left(\frac{a}{m}\right) \cdot \left(\frac{b}{m}\right).$$

Some properties of the Legendre symbol generalize nicely to the Jacobi symbol.

Proposition 2.4.10. *For any odd integer n:*

$$\left(\frac{1}{n}\right) = 1 \qquad \text{and} \qquad \left(\frac{-1}{n}\right) = (-1)^{\frac{n-1}{2}}.$$

Proof. The first assertion is trivial. In order to show the second one it suffices to prove that a map $n \mapsto (-1)^{(n-1)/2}$ is multiplicative on odd integers. Take two odd integers $m, n \in \mathbb{Z}$ and consider all possible remainders modulo 4:

1. if $m \equiv n \equiv 1 \pmod 4$, then also $mn \equiv 1 \pmod 4$ and we have
$$(-1)^{\frac{m-1}{2}} \cdot (-1)^{\frac{n-1}{2}} = 1 \cdot 1 = 1 = (-1)^{\frac{mn-1}{2}};$$
2. if $m \equiv 1 \pmod 4$ and $n \equiv 3 \pmod 4$, then $mn \equiv 3 \pmod 4$ and we have
$$(-1)^{\frac{m-1}{2}} \cdot (-1)^{\frac{n-1}{2}} = 1 \cdot (-1) = -1 = (-1)^{\frac{mn-1}{2}};$$
3. if $m \equiv 3 \pmod 4$ and $n \equiv 1 \pmod 4$, then $mn \equiv 3 \pmod 4$ and we have
$$(-1)^{\frac{m-1}{2}} \cdot (-1)^{\frac{n-1}{2}} = (-1) \cdot 1 = -1 = (-1)^{\frac{mn-1}{2}};$$

4. if $m \equiv n \equiv 3 \pmod 4$, then $mn \equiv 1 \pmod 4$ and we have

$$\left(-1\right)^{\frac{m-1}{2}} \cdot \left(-1\right)^{\frac{n-1}{2}} = (-1)\cdot(-1) = 1 = \left(-1\right)^{\frac{mn-1}{2}}.$$

This proves the multiplicativity and the assertion is now a direct consequence of the definition of Jacobi symbol. □

Proposition 2.4.11. *If n is an odd positive integer, then*

$$\left(\frac{2}{n}\right) = (-1)^{\frac{n^2-1}{8}}.$$

Proof. The left-hand-side of the above formula is multiplicative by the definition of the Jacobi symbol and the equality holds for primes. Thus it suffices to prove that the right-hand-side is multiplicative, as well. All it takes is to check the four possible cases:

1. if $m \equiv \pm 1 \pmod 8$ and $n \equiv \pm 1 \pmod 8$, then $mn \equiv \pm 1 \pmod 8$ hence

$$(-1)^{\frac{m^2-1}{8}} \cdot (-1)^{\frac{n^2-1}{8}} = 1 \cdot 1 = 1 = (-1)^{\frac{(mn)^2-1}{8}};$$

2. if $m \equiv \pm 3 \pmod 8$ and $n \equiv \pm 1 \pmod 8$, then $mn \equiv \pm 3 \pmod 8$ hence

$$(-1)^{\frac{m^2-1}{8}} \cdot (-1)^{\frac{n^2-1}{8}} = (-1)\cdot 1 = -1 = (-1)^{\frac{(mn)^2-1}{8}};$$

3. if $m \equiv \pm 1 \pmod 8$ and $n \equiv \pm 3 \pmod 8$, then $mn \equiv \pm 3 \pmod 8$ hence

$$(-1)^{\frac{m^2-1}{8}} \cdot (-1)^{\frac{n^2-1}{8}} = 1 \cdot (-1) = -1 = (-1)^{\frac{(mn)^2-1}{8}};$$

4. if $m \equiv \pm 3 \pmod 8$ and $n \equiv \pm 3 \pmod 8$, then $mn \equiv \pm 1 \pmod 8$ hence

$$(-1)^{\frac{m^2-1}{8}} \cdot (-1)^{\frac{n^2-1}{8}} = (-1)\cdot(-1) = 1 = (-1)^{\frac{(mn)^2-1}{8}}.$$ □

In a similar way we generalize the quadratic reciprocity law.

Proposition 2.4.12. *If m, n are two positive odd integers, then*

$$\left(\frac{m}{n}\right) = \begin{cases} -\left(\frac{n}{m}\right), & \text{if } m \equiv n \equiv 3 \pmod 4 \\ \left(\frac{n}{m}\right), & \text{otherwise.} \end{cases}$$

Proof. First suppose that m and n are not relatively prime, then there is a prime p that divides both of them. Therefore, $\left(\frac{m}{p}\right) = 0 = \left(\frac{n}{p}\right)$ and it follows that $\left(\frac{m}{n}\right) = 0 = \left(\frac{n}{m}\right)$. Thus, without loss of generality we may assume that $\gcd(m,n) = 1$. Factor both integers into primes, let

$$m = p_1^{k_1} \cdots p_s^{k_s} \qquad \text{and} \qquad n = q_1^{l_1} \cdots q_t^{l_t}.$$

The map $m \mapsto (-1)^{m-1/2}$ is multiplicative on odd integers (we have verified this fact in the proof of Proposition 2.4.10). Therefore, by Theorem 2.4.7 and Corollary 2.4.5 we have

$$\left(\frac{m}{n}\right) = \left(\frac{p_1^{k_1}\cdots p_s^{k_s}}{q_1^{l_1}\cdots q_t^{l_t}}\right) = \prod_{i=1}^{s}\prod_{j=1}^{t}\left(\frac{p_i}{q_j}\right) = \prod_{i=1}^{s}\prod_{j=1}^{t}(-1)^{\frac{p_i-1}{2}}\cdot(-1)^{\frac{q_j-1}{2}}\cdot\left(\frac{q_j}{p_i}\right)$$
$$= (-1)^{\frac{m-1}{2}}\cdot(-1)^{\frac{n-1}{2}}\cdot\prod_{i=1}^{s}\prod_{j=1}^{t}\cdot\left(\frac{q_j}{p_i}\right) = (-1)^{\frac{(m-1)(n-1)}{4}}\cdot\left(\frac{n}{m}\right).$$

As in the proof of Theorem 2.4.7, the last formula is equivalent to the thesis. □

Example 28. Let us return to the previous example. Compute

$$\left(\frac{887}{1237}\right) = \left(\frac{1237}{887}\right) = \left(\frac{350}{887}\right) = \left(\frac{2}{887}\right)\cdot\left(\frac{175}{887}\right)$$
$$= 1\cdot\left(\frac{887}{175}\right) = \left(\frac{12}{175}\right) = \left(\frac{2}{175}\right)^2\cdot\left(\frac{3}{175}\right) = 1\cdot\left(\frac{175}{3}\right) = \left(\frac{1}{3}\right) = 1.$$

We shall formalize the procedure applied in the above example and write it as an algorithm. It can be viewed as a variation of Euclidean algorithm.

Algorithm 2.4.13. *Given an odd moduli $n > 2$ and any integer $m \in \mathbb{Z}$, this algorithm computes the Jacobi symbol $\left(\frac{m}{n}\right)$. In particular, if $n = p$ is a prime number, this algorithm checks whether m is a quadratic residue modulo p.*

1. *Initialize $J := 1$;*
2. *as long as m is nonzero repeat the following steps:*
   a) *if m is negative, then:*
      i. *if in addition $n \equiv 3 \pmod 4$, replace J by $-J$;*
      ii. *replace m by $-m$;*
   b) *if m is even, then:*
      i. *write m as $m = 2^k \cdot m'$, where m' is odd;*
      ii. *if the exponent k is odd and $n \equiv \pm 1 \pmod 8$, replace J by $-J$;*
      iii. *replace m by m';*
   c) *if $m < n$, then:*
      i. *if in addition $m \equiv n \equiv 3 \pmod 4$, then replace J by $-J$;*
      ii. *swap the values of m and n;*
   d) *replace m by its remainder $m \bmod n$;*
3. *if $n = 1$, then return J, otherwise return 0.*

Proof of correctness.

missing

□

Richard P. Brent and Paul Zimmermann, An $O(M(n)\log n)$ algorithm for the Jacobi symbol, Proc. ANTS-IX, LNCS 6197 (2010), 83–95
other(s) sub-quadratic Jacobi alg(s)?

Although the above algorithm computes the Jacobi symbol for any possible moduli, as mentioned earlier, it answers the question whether a given element is a square in a quotient ring only for prime modulus (i.e. when the the quotient ring is actually a field). We will now consider the case, when the modulus is a power of a prime.

We again have to separately treat the even prime. The possible remainders modulo 4 are: $0, 1, 2, 3$ and their squares equal $0^2 = 0$, $1^2 = 1$, $2^2 = 0$ and $2^2 = 1$. Thus we have:

Observation 2.4.14. *An integer a is a quadratic residue modulo 4 if and only if it is congruent (modulo 4) to either 0 or 1.*

Consider now higher powers of two.

Proposition 2.4.15. *If $a \in \mathbb{Z}$ is an odd integer and $k \geq 3$, then a is a quadratic residue modulo 2^k if and only if $a \equiv 1 \pmod 8$.*

Proof. One implication is easy. Take an odd number $b = 2s + 1$ and square it, then $b^2 = 4s^2 + 4s + 1 = 4s \cdot (s + 1) + 1$. Either s or $s + 1$ is even, hence $b^2 \equiv 1 \pmod 8$. We shall now prove the opposite implication. Given an integer congruent to 1 modulo 8, we must show that it is a quadratic residue modulo 2^k. To this end, take a set of all the residues modulo 2^k that are congruent to 1 (mod 8):

$$\mathscr{M} := \left\{a \in U(\mathbb{Z}/2^k\mathbb{Z}) \,|\, a \equiv 1 \pmod 8\right\}.$$

The set $\mathscr{M}$ consists of precisely 2^{k-3} elements: $1, 9, 17, \ldots, 2^k - 7$. We will construct another set of the same cardinality. Let $\mathscr{N}$ consists of all odd integers less than 2^{k-2}:

$$\mathscr{N} := \left\{1, 3, \ldots, 2^{k-2} - 1\right\}.$$

It is clear that there are 2^{k-3} of them. Compute the squares (in the ring $\mathbb{Z}/2^k\mathbb{Z}$) of elements in $\mathscr{N}$:

$$\mathscr{N}^2 := \left\{m^2 \bmod 2^k \,|\, m \in \mathscr{N}\right\}.$$

We claim that the cardinality does not drop and $\mathscr{N}^2$ has the same number of elements as $\mathscr{N}$, hence also $\mathscr{M}$. Suppose a contrario that there are distinct $m, n \in \mathscr{N}$ such that $m^2 \equiv n^2 \pmod{2^k}$. Then $(m+n)(m-n) \equiv 0 \pmod{2^k}$. now, one of the numbers $m+n$, $m-n$ must be congruent to 2 modulo 4. This is easily verified looking at all possible remainders of m, n modulo 4. For the whole product to be divisible by 2^k, the other factor must be divisible by 2^{k-1}. But this is impossible, since $m, n < 2^{k-2}$. This proves our claim.

By the first part of the proof, we know that a square of any odd integer is congruent to 1 modulo 8. Hence $\mathscr{N}^2$ is contained in $\mathscr{M}$. We saw that they have the same cardinality and so they are equal. Thus, every integer a congruent to 1 (mod 8) is a quadratic residue modulo 2^k. □

Remark 25. We are used to the fact that every square a^2 has precisely two square roots: a and $-a$. The situation becomes more complex when working with quadratic residues modulo powers of two. Modulo 2, we have only two possible remainders: $0^2 = 0$ and $1^2 = 1$. This means that a quadratic residue modulo 2 has only one (modular) square root. Modulo 4, we are on familiar grounds, since $1 \equiv 1^2 \equiv 3^2 \pmod 4$ and $0 \equiv 0^2 \equiv 2^2 \pmod 4$. A quadratic residue modulo 4 has precisely two (modular) square roots. Modulo 8 and higher powers of two, an odd quadratic residue has precisely... four (modular) square roots:

$$a^2 \equiv (-a)^2 \equiv (2^{k-1} + a)^2 \equiv (2^{k-1} - a)^2 \pmod{2^k}.$$

For instance a square of any odd integer is congruent to 1 modulo 8 and so the square roots of 1 in $\mathbb{Z}/8\mathbb{Z}$ are: $1, 3, 5, 7$.

Now we turn our attention to modulus being powers of odd primes.

Theorem 2.4.16. *Let p be an odd prime and a be an integer not divisible by p, then a is a quadratic residue modulo p^k if and only if it is a quadratic residue modulo p.*

Proof. One implication is trivial. If $a \equiv b^2 \pmod{p^k}$, then $p^k \mid a - b^2$ and so also $p \mid a - b^2$. Thus, every quadratic residue modulo p^k is a quadratic residue modulo p. Conversely, assume that a is a quadratic residue modulo p. We will prove by induction that a is a quadratic residue modulo every power p^k of p. The claim is trivially true for $k = 1$. Thus, assume that $a \equiv b^2 \pmod{p^k}$ for some $b \in \mathbb{Z}$. This means that there is $m \in \mathbb{Z}$ such that $b^2 = a + mp^k$. Since p is odd and b is not divisible by p (otherwise p would divide a), we may find $n \in \mathbb{Z}$ such that $m + 2bn \equiv 0 \pmod p$. Calculate

$$\begin{aligned}\left(b + np^k\right)^2 &= b^2 + 2bn \cdot p^k + n^2 p^{2k} \\ &= a + (m + 2bn) \cdot p^k + n^2 p^{2k} \\ &\equiv a \pmod{p^{k+1}}.\end{aligned}$$

Therefore, a is a quadratic residue modulo p^{k+1} and so also modulo every power of p. □

Remark 26. Observe that the proof of the above theorem is effective in the sense that once we have a square root of a modulo p, we can "lift" it to a square root of a modulo any power of p. We will use this fact later.

Proposition 2.4.15 and Theorem 2.4.16 deal with residues relatively prime to the modulus. But it is not hard to derive a general criterion now. First let us deal with a trivial case.

Observation 2.4.17. *Let $p \geq 2$ be a prime number. If $a = p^n \cdot \overline{a}$ and $n \geq k$, then $a \equiv 0 \pmod{p^k}$ and so is a quadratic residue.*

Now, the general criterion reads as follows:

Theorem 2.4.18. *Let $p \geq 2$ be a prime number. If $a = p^n \cdot \overline{a}$, where $n < k$ and $\overline{a}$ is not divisible by p, then a is a quadratic residue modulo p^k if and only if n is even and:*

- *$\overline{a}$ is a quadratic residue modulo p, when $p > 2$,*
- *$\overline{a}$ is a quadratic residue modulo 2^{k-n}, when $p = 2$.*

Proof. One implication is immediate. Assume that $n = 2m$ is an even number and $\overline{a}$ is a quadratic residue. Hence, there is $b \in \mathbb{Z}$ such that $\overline{a} \equiv b^2 \pmod{p^k}$. Consequently $a \equiv (p^m \cdot b)^2 \pmod{p^k}$.

Conversely assume that a is a quadratic residue modulo p^k. Therefore, there is some b such that p^k divides $p^n\overline{a} - b^2$. This in particular means that p, being a prime number, divides b. Thus we can write $b^2 = p^{2m}\overline{b}$, where $p \nmid \overline{b}$. By assumption $k > n$, therefore $p^n \mid p^n\overline{a} + p^{2m}\overline{b}$, hence $2m \geq n$. Thus n must be even, because otherwise we would have $p^{n+1} \mid p^n\overline{a} + p^{2m}\overline{b}$, which implies $p^{n+1} \mid p^n\overline{a}$ and this is impossible since $p \nmid \overline{a}$.

It remains to show that $\overline{a}$ is a quadratic residue modulo p^k. By the previous part we can write

$$p^n\overline{a} \equiv p^{2m}\overline{b}^2 \pmod{p^k}, \qquad \text{where} \quad 2m \geq n.$$

This means that for some $s \in \mathbb{Z}$ we have

$$p^n\overline{a} - p^{2m}\overline{b}^2 = p^k s$$

Divide both sides by p^n to see that $\overline{a}$ is a quadratic residue modulo p^{k-n}. This concludes the proof in case $p = 2$ and for $p > 2$ the thesis follows from Theorem 2.4.16. □

We may now combine above results into a single algorithm.

Algorithm 2.4.19. *Let $p \geq 2$ be a prime number and $m = p^k$ be a prime-power modulus. Given an integer a, this algorithm returns true, when a is a quadratic residue modulo p^k and false otherwise.*

1. *If $m = 2$, then return true;*
2. *if $m = 4$, check whether $a \equiv 0$ or $a \equiv 1 \pmod 4$, if so then return true, otherwise return false;*
3. *if $m = 2^k$ and $k \geq 3$, then:*
   a) *write a as $a = 2^n\overline{a}$, where $\overline{a}$ is odd;*
   b) *if $n \geq k$, then return true and quit;*
   c) *if n is odd, then return false and quit;*
   d) *if $k - n \geq 3$, verify whether $a \equiv 1 \pmod 8$, if it is the case, return true, otherwise return false;*
   e) *if $k - n < 3$, execute this algorithm recursively with input $\overline{a}, 2^{k-n}$ and return the result.*
4. *If $m = p^k$, where p is an odd prime, then:*
   a) *write a as $a = p^n\overline{a}$, where $p \nmid \overline{a}$;*
   b) *if $n \geq k$, then return true and quit;*

c) if n is odd, then return false and quit;
d) compute Legendre symbol $\left(\frac{a}{p}\right)$ using Algorithm 2.4.13;
e) if $\left(\frac{a}{p}\right) = 1$, return true and quit;
f) otherwise return false.

Finally we are ready to deal with arbitrary compound moduli. Fix a modulus $m = p_1^{k_1} \cdots p_n^{k_n}$. If $a \in \mathbb{Z}$ is a quadratic residue modulo m, then there is some $b \in \mathbb{Z}$ such that $m \mid a - b^2$. In consequence, every $p_j^{k_j}$ divides $a - b^2$, as well. Hence, a is a quadratic residue modulo $p_1^{k_1}, \ldots, p_n^{k_n}$. Conversely, assume that a is a quadratic residue modulo every $p_j^{k_j}$. Thus, there are $b_1, \ldots, b_n$ satisfying $b_j^2 \equiv a \pmod{p_j^{k_j}}$ for $j \in \{1, \ldots, n\}$. Consider a system of congruences (with pairwise relatively prime moduli):

$$\begin{cases} x \equiv b_1 \pmod{p_1^{k_1}} \\ \quad \vdots \\ x \equiv b_n \pmod{p_n^{k_n}}. \end{cases}$$

Chinese remainder theorem asserts that it has a unique solution $b \in {}^{\mathbb{Z}}/_{m\mathbb{Z}}$. It follows that b^2 is congruent to a modulo m. In particular a is a quadratic residue modulo m. This way we have proved:

Proposition 2.4.20. *If $m = p_1^{k_1} \cdots p_n^{k_n}$ and $a \in \mathbb{Z}$, then a is a quadratic residue modulo m if and only if it is a quadratic residue modulo every $p_j^{k_j}$ for $j \in \{1, \ldots, n\}$.*

A trivial conversion of this proposition into an algorithm is left to the reader. Observe that the discussion preceding Proposition 2.4.20 not only proves the thesis but also offers an effective method of finding a square root modulo m, providing that we are able to compute square roots modulo prime powers. Recall that the proof of Theorem 2.4.16 was also effective and reduced a problem of computing a square root modulo an odd prime power to that of computing one modulo the underlying prime. Thus the last gap that we need to fill is to present a method for computing square roots for prime moduli (and a method for "lifting" square roots modulo powers of 2).

Let p be an odd prime. It is clear that a modular square roots of an integer a divisible by p is just 0. Thus without loss of generality we may concentrate on a's not divisible by p. An odd prime (in fact any odd integer) is congruent modulo 4 to either 1 or 3. An there are infinitely many primes of both types. It turns out that in half of the cases computation of a modular square root is exceptionally easy.

Proposition 2.4.21. *Let p be a prime congruent to 3 modulo 4 and a be an integer not divisible by p. If a is a quadratic residue modulo p, then $b := a^{(p+1)/4}$ is a square root of a in $\mathbb{F}_p$.*

Proof. Since $p \equiv 3 \pmod 4$, the exponent ${}^{(p+1)}/_4$ is integer. Euler's criterion (Theorem 2.4.3) asserts that $1 = \left(\frac{a}{p}\right) \equiv a^{(p-1)/2} \pmod p$. Multiplying both sides by a we have

$$a \equiv a^{\frac{p+1}{2}} = \left(a^{\frac{p+1}{4}}\right)^2 = b^2 \pmod p \qquad \square$$

This leaves us with the other half of primes, namely those that are congruent to 1 (mod 4). Here we do not have such a universal formula as the one presented above, yet still in half of these remaining cases computation of modular square roots is easy.

Proposition 2.4.22 (Atkin)**.** *Let p be a prime congruent to 5 modulo 8 and a be an integer not divisible by p. Set*

$$c := \left(2a\right)^{\frac{p-5}{8}} \qquad \text{and} \qquad b := ac \cdot \left(2ac^2 - 1\right).$$

If a is a quadratic residue modulo p, then $b^2 \equiv a \pmod{p}$.

Proof. Proposition 2.4.6 asserts that 2 is a nonresidue modulo p. A product of a residue and a nonresidue is a nonresidue. Thus $2a \notin \mathbb{F}_p^2$ and by Euler's criterion $\left(2a\right)^{(p-1)/2} \equiv -1 \pmod{p}$ and so we have

$$c^4 \equiv (2a)^{\frac{p-5}{2}} = \frac{(2a)^{\frac{p-1}{2}}}{(2a)^2} \equiv \frac{-1}{4a^2} \pmod{p}.$$

Our goal is to show that $b^2 \equiv a \pmod{p}$ hence compute

$$b^2 = a^2c^2 \cdot \left(2ac^2 - 1\right)^2 = a^2c^2 \cdot \left(4a^2c^4 - 4ac^2 + 1\right) \equiv a^2c^2 \cdot \left(-4ac^2\right) = a \cdot (-4a^2) \cdot c^4 \equiv a \pmod{p}$$

□

The two preceding propositions solve the problem in 75% of cases. Yet there are still the remaining 25% for which we need to devise a general algorithm.

Remark 27. In fact there are known formulas—similar to the above ones—also for $p \equiv 9 \pmod{16}$ and $p \equiv 17 \pmod{32}$. Therefore one really needs a general method only in 6.25% of cases. For references see the 'Bibliographic notes' section at the end of this chapter.

The following algorithm was independently discovered by A. Tonelli in 1891 and D. Shanks in 1973. It computes a modular square root of a in a sequence of consecutive "approximations" $b_0, b_1, \ldots, b_k$, where $b_k^2 \equiv a \pmod{p}$.

Algorithm 2.4.23 (Tonelli–Shanks, 1891/1973)**.** *Let p be an odd prime and ζ a fixed nonresidue modulo p. Given a quadratic residue a, not divisible by p, this algorithm computes b such that $b^2 \equiv a \pmod{p}$.*

1. *If $p \equiv 3 \pmod{4}$ apply Proposition 2.4.21 and quit;*
2. *if $p \equiv 5 \pmod{8}$ apply Proposition 2.4.22 and quit;*
3. *write the modulus p in form $p = 1 + 2^s \cdot q$ for some odd q and $s > 1$;*
4. *initialize $i := 0$ and $b_0 := a^{(q+1)/2}$;*
5. *initialize auxiliary sequences:*

$$s_0 := s, \quad t_0 := a^q, \quad z_0 := \zeta^q;$$

6. *as long as* $t_i \not\equiv 1 \pmod p$ *do the following:*
   a) *by repeating squaring find the lowest exponent* s_{i+1} *such that* $t^{2^{s_{i+1}}} \equiv 1 \pmod p$*;*
   b) *denote* $c_{i+1} := z_i^{2^{s_i - s_{i+1} - 1}}$*;*
   c) *construct the next "approximation" of the square root:*

$$b_{i+1} := b_i \cdot c_{i+1} \bmod p;$$

   d) *compute successive elements of auxiliary sequences*

$$t_{i+1} := t_i \cdot c_{i+1}^2 \bmod p \qquad \textit{and} \qquad z_{i+1} := c_{i+1}^2;$$

   e) *increment* i*;*
7. *return* $b = b_i$*;*

Proof of correctness. We claim that throughout the execution of the algorithm the following identity holds:

$$b_i^2 \equiv t_i \cdot a \pmod p \qquad \text{for every } i.$$

Since the definitions of all the sequences used in the algorithm are recursive, it should not be surprising that we prove our claim by induction. For $i = 0$, we have $b_0 \equiv a^{(q+1)/2}$ and $t_0 \equiv a^q \pmod 2$, hence $b_0^2 \equiv ta \pmod p$. Assume that the claim holds true for some $i \geq 0$. Elements b_{i+1} and t_{i+1} are constructed as

$$b_{i+1} := b_i \cdot c_{i+1} \bmod p \qquad \text{and} \qquad t_{i+1} := t_i \cdot c_{i+1}^2 \bmod p,$$

therefore using the inductive hypothesis we write

$$b_{i+1}^2 \equiv b_i^2 \cdot c_{i+1}^2 \equiv (t_i \cdot a) \cdot c_{i+1}^2 \equiv t_{i+1} \cdot a \pmod p.$$

This proves our claim. The algorithm stops when $t_i \equiv 1 \pmod p$. By our claim this means that $b_i^2 \equiv a \pmod p$ as requested.

It remains to show that the algorithm terminates in finite time and that for every t_i there is s_{i+1} such that $t_i^{s_{i+1}} \equiv 2$ modulo p. Inductively we may express z_i as

$$z_i \equiv c_i^2 \equiv \left(z_{i-1}^{2^{s_{i-1} - s_i - 1}}\right)^2 = z_{i-1}^{2^{s_{i-1} - s_i}} \equiv z_{i-2}^{2^{s_{i-2} - s_i}} \equiv \cdots \equiv z_0^{2^{s_0 - s_i}} \equiv \zeta^{2^{s - s_i} \cdot q} \pmod p.$$

We have $t_0 = a^q \pmod p$, which implies that $t_0^{2^s} \equiv a^{2^s \cdot q} = a^{p-1} \equiv 1$ modulo p by Fermat's little theorem. Consequently, by mathematical induction we have

$$t_{i+1}^{2^{s_{i+1}}} \equiv \left(t_i \cdot c_{i+1}^2\right)^{2^{s_{i+1}}} \equiv 1 \cdot c_{i+1}^{2^{s_{i+1}+1}} \equiv z_i^{2^{s_i - s_{i+1} - 1} \cdot 2^{s_{i+1}+1}} = z_i^{2^{s_i}} \equiv \zeta^{2^{s_i} \cdot 2^{s - s_i} \cdot q} = \zeta^{p-1} \equiv 1 \pmod p.$$

Thus the rank of every t_i in the group $U\mathbb{F}_p$ divides 2^{s_i} and so must be a power of two itself. It is clear that the sequence of s_i's is strictly decreasing, hence the algorithm terminates after finitely many iterations. □

Table 2.1: Smallest quadratic nonresidues for primes less than 200

prime	nonresidue	prime	nonresidue	prime	nonresidue
3	2	5	2	7	3
11	2	13	2	17	3
19	2	23	5	29	2
31	3	37	2	41	3
43	2	47	5	53	2
59	2	61	2	67	2
71	7	73	5	79	3
83	2	89	3	97	5
101	2	103	3	107	2
109	2	113	3	127	3
131	2	137	3	139	2
149	2	151	3	157	2
163	2	167	5	173	2
179	2	181	2	191	7
193	5	197	2	199	3

In order to work, Tonelli–Shanks algorithm needs a quadratic nonresidue ζ. (The same can be said about Cipolla–Lehmer algorithm that we discuss next.) To the best of the author's knowledge, there is no deterministic algorithm constructing a quadratic nonresidue modulo an odd prime $p \not\equiv 3 \pmod 4$, that is provable to run in polynomial time with respect to the number of bits of p (i.e. polynomial in $\lg p$). There is however a very simple probabilistic method. The field $\mathbb{F}_p$ consists of $(p+1)/2$ squares (including zero) and $(p-1)/2$ non-squares. Hence the probability that a random $z \in \mathbb{Z}$ is a quadratic residue modulo p equals $(p-1)/2p$, which is approximately $1/2$ for large p. Thus, on average one needs to take two random integers to find a quadratic nonresidue.

It should be remarked that if the so-called *generalized Riemann hypothesis* is true (which most experts believe to be so), then it implies that for every odd prime p there exists an integer $z < 2 \cdot (\ln p)^2$ which is a quadratic nonresidue modulo p. Therefore, checking sequentially all candidates $z = 2, 3, \ldots$, one will find a nonresidue in polynomial time. Table 2.1 gathers smallest (positive) nonresidues for primes under 200.

Example 29. Let us calculate a square root of 2 in a field $\mathbb{F}_{17}$. Since 17 is neither congruent to 3 modulo 4, nor to 5 modulo 8, we cannot use Propositions 2.4.21 and 2.4.22. Refer to Table 2.1 and take $\zeta := 3$. We have $17 = 1 + 2^s \cdot q = 1 + 2^4 \cdot 1$ so that $s = 4$ and $q = 1$. The first "approximation" of the modular square root of 2 is

$$b_0 := a^{\frac{q+1}{2}} = 2^{\frac{1+1}{2}} = 2.$$

Definitely $b_0^2 \neq 2$, therefore we need at least one iteration of the main loop of Algo-

rithm 2.4.23. To this end we initialize the auxiliary sequences:

$$s_0 := s = 4, \qquad t_0 := a^q = 2^1 = 2, \qquad z_0 := \zeta^q = 3^1 = 3.$$

By successive squaring, we find that

$$t_0 = 2, \qquad t_0^2 = 4, \qquad t_0^{2^2} = 16, \qquad t_0^{2^3} \equiv 1 \pmod{17},$$

hence $s_1 = 3$. Consequently $c_1 = z_0^{2^{s_0-s_1-1}} = 3^{2^{4-3-1}} = 3$ and our next "approximation" equals

$$b_1 := b_0 \cdot c_1 = 2 \cdot 3 = 6.$$

Next, we update the auxiliary sequences, setting

$$t_1 := t_0 \cdot c_1^2 = 2 \cdot 3^2 \equiv 1 \pmod{17} \qquad \text{and} \qquad z_1 := c_1^2 = 3^2 = 9.$$

Now, $t_1 = 1$, which means that $b_1 = 6$ is the sought square root of 2 modulo 17, Indeed $6^2 \equiv 2 \pmod{17}$. The other square root is $(-6) \bmod 17 = 11$.

Recall how the field $\mathbb{C}$ of complex numbers is constructed. We just take pairs (x, y) of reals, written as $x + yi$ and we define arithmetic operations on them, extending the operations from the reals by the means of the distributive law and an identity $i^2 = 1$. Observe that the only property which we really need to make it works is that -1 is not a square in $\mathbb{R}$. This construction *extends* the field of reals by appending a (formal) square roots of -1 to it. We may repeat the same process with any field providing that we know an element which is not a square (see also the next section for a general notion of field extensions). Assume that p is an odd prime and $\zeta \in \mathbb{F}_p$ is not a square. Take the set of pairs of elements from $\mathbb{F}_p$:

$$\mathbb{F}_{p^2} := \mathbb{F}_p \times \mathbb{F}_p = \big\{(x, y) \,|\, x, y \in \mathbb{F}_p\big\}.$$

We define addition and multiplication in $\mathbb{F}_{p^2}$ as follows

$$(x_1, y_1)+(x_2, y_2) := (x_1+x_2, y_1+y_2), \qquad (x_1, y_1)\cdot(x_2, y_2) := (x_1x_2+y_1y_2\zeta, x_1y_2+x_2y_1).$$

If we now identify the field $\mathbb{F}_p$ with the subset $\mathbb{F}_p \times \{0\} \subset \mathbb{F}_{p^2}$, then the arithmetic operations on $\mathbb{F}_p$ agree with those defined above, indeed:

$$x + y \sim (x, 0) + (y, 0) = (x + y, 0), \qquad x \cdot y \sim (x, 0) \cdot (y, 0) = (xy, 0).$$

Denote $i := (0, 1)$ and observe that $i^2 = (0, 1) \cdot (0, 1) = (\zeta, 0)$. Hence under our identification, $i^2 = \zeta \in \mathbb{F}_p \subset \mathbb{F}_{p^2}$. Therefore we may write $x + yi$ for a pair (x, y) and this way simplify the formulas for addition and multiplication to

$$(x_1+y_1i)+(x_2+y_2i) = (x_1+x_2)+(y_1+y_2)\cdot i, \qquad (x_1+y_1i)\cdot(x_2+y_2i) = (x_1x_2+y_1y_2\zeta)+(x_1y_2+x_2y_1)\cdot i$$

It is clear that $-(x + yi) = (-x) + (-y) \cdot i$. The multiplicative inverse is computed by a well known process of "rationalizing the denominator". Let $x + yi \neq 0$, then

$$\frac{1}{x+yi} = \frac{x-yi}{(x+yi)(x-yi)} = \frac{x-yi}{x^2-y^2\zeta} = \frac{x}{x^2-y^2\zeta} + \frac{-y}{x^2-y^2\zeta} \cdot i.$$

The denominator $x^2 - y^2\zeta$ is nonzero, since ζ is not a square.

Observation 2.4.24. *The set $\mathbb{F}_{p^2}$ with arithmetic operations defined above is a field.*

Do not confuse the field $\mathbb{F}_{p^2}$ with $\mathbb{Z}_{p^2}$ defined on the beginning of this section. The latter is ring of remainders modulo p^2. It is not a field since it contains a zero-divisor p. We will now use the field structure of $\mathbb{F}_{p^2}$ to compute square roots in modulo p. Fix an odd prime p and a quadratic residue a. Further let $z \in \mathbb{F}_p$ be an element selected in such a way that $\zeta := z^2 - a$ is a nonresidue modulo p. For this choice of ζ build a field $\mathbb{F}_{p^2}$, as described above.

Lemma 2.4.25. *For every $x + iy \in \mathbb{F}_{p^2}$ we have*

$$(x + iy)^p = x - iy.$$

Proof. By definition, $i^2 = \zeta$ is a nonresidue modulo p. Hence, Euler's criterion (Theorem 2.4.3) asserts that in $\mathbb{F}_p$:

$$-1 = \left(\frac{\zeta}{p}\right) = \zeta^{\frac{p-1}{2}} = i^{p-1}.$$

It follows that $i^p = -i$. (Recall how in $\mathbb{C}$ we have $i^{-1} = -i$.) Compute

$$(x + iy)^p = x^p + \binom{p}{1} x^{p-1} iy + \cdots + \binom{p}{p-1} x i^{p-1} y^{p-1} + i^p y^p.$$

The binomial coefficients of all the terms, except the first and the last, are divisible by p, hence vanish in $\mathbb{F}_p$. Therefore

$$(x + iy)^p = x^p + i^p y^p = x^p - iy^p = x - iy.$$

Here the last equality follows from Fermat's little theorem (Theorem 2.3.2). □

We may now compute the square root of a.

Proposition 2.4.26. *Denote $b := (z + i)^{(p+1)/2}$, then*

- $b^2 = a$,
- $b \in \mathbb{F}_p$.

Proof. The first assertion is obtained by a direct calculation:

$$b^2 = (z + i)^{p+1} = (z + i)(z + i)^p = (z + i)(z - i) = z^2 - i^2 = z^2 - \zeta = a$$

We need to show that the element b, although constructed in the bigger field $\mathbb{F}_{p^2}$, actually belongs to $\mathbb{F}_p$. To this end, recall that a is a quadratic residue modulo p. Thus, there is an element $c \in \mathbb{F}_p$ such that $x^2 = a$. The modulus p is odd, hence $c \neq -c$ and the letter is also a (modular) square root of a. It follows that b, c and $-c$ are roots of a polynomial $x^2 - a \in \mathbb{F}_{p^2}[x]$. But a polynomial of degree 2 can have no more than 2 roots by Corollary 1.1.19. It follows that b is either c or $-c$ and so belongs to $\mathbb{F}_p$. □

To sum up the above discussion, let us write down an algorithm, due to M. Cipolla[4] and D. Lehmer[5], that calculates modular square roots.

Algorithm 2.4.27 (Cipolla–Lehmer, 1907/1969)**.** *Let p be an odd prime. Given a quadratic residue a, not divisible by p, this algorithm computes b such that $b^2 \equiv a \pmod{p}$.*

1. *If $p \equiv 3 \pmod 4$ apply Proposition 2.4.21 and quit;*
2. *if $p \equiv 5 \pmod 8$ apply Proposition 2.4.22 and quit;*
3. *by the trial and error method, pick a random element $z \in \mathbb{F}_p$ such that $\zeta := z^2 - a$ is a quadratic nonresidue modulo p;*
4. *construct the field*
$$\mathbb{F}_{p^2} := \{x + yi \mid x, y \in \mathbb{F}_p\},$$
where $i^2 = \zeta$;
5. *using the arithmetic of $\mathbb{F}_{p^2}$ compute*
$$b := (z + i)^{\frac{p+1}{2}};$$
6. *return b.*

Example 30. We shall again compute a square root of 2 modulo 17. This time, however, we will use Cipolla–Lehmer algorithm, rather than Tonelli–Shanks, that we used in the previous example. Pick $z = 3$, then $\zeta = z^2 - a = 7$ is a quadratic nonresidue, because
$$\left(\frac{7}{17}\right) \equiv 7^8 \equiv -17 \pmod{17}$$
by Euler's criterion (Theorem 2.4.3). Extend the field $\mathbb{F}_{17}$ by $i := \sqrt{7}$, letting $\mathbb{F}_{17^2} := \{x + yi \mid x, y \in \mathbb{F}_{17}\}$. Proposition 2.4.26 asserts that
$$b := (z + i)^{\frac{p+1}{2}} = (3 + i)^9 = 6$$
is a modular square root of 2. Clearly 6 is an element of the base field $\mathbb{F}_{17}$.

Summarize to an algorithm computing $\sqrt{a} \pmod n$ for compound n

2.5 Applications of modular arithmetic

Modular gcd algorithm

Another similar, however slightly more involving, algorithm can be used for computing gcd of two polynomials over $\mathbb{Z}$ using modular methods. Hereafter for any polynomial

[4] Michele Cipolla (born 28 October 1880, died 7 September 1947) was an Italian number theorist.
[5] Derrick Henry Lehmer (born 23 February 1905, died 22 May 1991) was an American number theorist.

$f = f_0 + f_1 x + \cdots + f_n x^n \in \mathbb{Z}[x]$ and a prime number p, by $f_{(p)} \in \mathbb{F}_p[x]$ we denote a polynomial obtained from f by reducing every coefficient modulo p, i.e.

$$f_{(p)} = (f_0 \bmod p) + (f_1 \bmod p) \cdot x + \cdots + (f_n \bmod p) \cdot x^n.$$

It is a trivial observation that $\deg f_{(p)} \leq \deg f$ with an equality holding when p does not divide $\operatorname{lc} f = f_n$. We begin with a simple lemma.

Lemma 2.5.1. *Let f, g be two polynomials with integer coefficients. Denote their greatest common divisor by h. If p is a prime number not dividing the leading coefficients of f and g, then:*

1. $\deg h_p = \deg h$;
2. $\deg h \leq \deg\big(\gcd(f_{(p)}, g_{(p)})\big)$;
3. *if* $\deg h = \deg\big(\gcd(f_{(p)}, g_{(p)})\big)$, *then up to a multiplication by a nonzero constant* $h_p = \gcd(f_{(p)}, g_{(p)})$.

Proof. The leading coefficient of $h = \gcd(f, g)$ divides the leading coefficients of f and g. Since p does not divide any of them, it follows that p cannot divide $\operatorname{lc} h$. This shows that $\operatorname{lc} h_p = (\operatorname{lc} h \bmod p) \neq 0$ and so $\deg h_p = \deg h$. Now, h divides both f and g, hence h_p divides the reduced polynomials $f_{(p)}$ and $g_{(p)}$. Therefore h_p divides $\gcd(f_{(p)}, g_{(p)})$, which implies that $\deg h_p \leq \deg\big(\gcd(f_{(p)}, g_{(p)})\big)$ and point 2 follows from 1. Finally, if the degrees of h and $\gcd(f_{(p)}, g_{(p)})$ agree, then also $\deg h_p = \deg\big(\gcd(f_{(p)}, g_{(p)})\big)$, hence $h_p = \alpha \cdot \gcd(f_{(p)}, g_{(p)})$ for some $\alpha \in \mathbb{F}_p$. □

Let p be a prime number. In what follows, by a *lift* of a polynomial $f = f_0 + f_1 x + \cdots + f_n x^n \in \mathbb{F}_p[x]$ we understand a polynomial $F = F_0 + F_1 x + \cdots + F_n x^n \in \mathbb{Z}[x]$ such that $F_p = f$ and each coefficient F_i is the smallest non-negative integer with reminder f_i modulo p. In layman's terms, F is just a polynomial f, where the coefficients $f_i \in \{0, \ldots, p-1\}$ are regarded as elements of $\mathbb{Z}$ rather than $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$.

Since $\mathbb{F}_p$ is a field the class $\gcd(f_{(p)}, g_{(p)})$ consists of $p-1$ elements. Any nonzero element of $\mathbb{F}_p$ can be chosen as the leading coefficient of $\gcd(f_{(p)}, g_{(p)})$. In order for the Chinese remainder theorem to work correctly, we need to be consistent in selecting the representatives of these classes. To this end we will always pick up $\gcd(f_{(p)}, g_{(p)})$ with the leading coefficient equal to $\gcd(\operatorname{lc} f, \operatorname{lc} g) \bmod p$.

Let f, g be two primitive polynomials with integer coefficients. By the previous lemma, if we know a priori the degree (denote it d) of the greatest common divisor of f and g and have some bound M for the absolute values of its coefficients, we may compute $\gcd(f, g)$ along the following lines:

1. *pick any prime p not dividing* $\operatorname{lc} f$, $\operatorname{lc} g$ *and such that* $\deg\big(\gcd(f_{(p)}, g_{(p)})\big) = d$;
2. *initialize $m := p$ and let H be a lift to $\mathbb{Z}[x]$ of $\gcd(f_{(p)}, g_{(p)}) \in \mathbb{F}_p[x]$, where* $\operatorname{lc}\gcd(f_{(p)}, g_{(p)}) = \big(\gcd(\operatorname{lc} f, \operatorname{lc} g) \bmod p\big)$;
3. *as long as $m < 2M$ repeat the following steps:*

   a) *pick a new prime p not dividing* $\operatorname{lc} f$, $\operatorname{lc} g$ *and such that* $\deg\big(\gcd(f_{(p)}, g_{(p)})\big) = d$;
   b) *set h to be a lift to $\mathbb{Z}[x]$ of* $\gcd(f_{(p)}, g_{(p)}) \in \mathbb{F}_p[x]$, *again* $\operatorname{lc}\gcd(f_{(p)}, g_{(p)}) = \big(\gcd(\operatorname{lc} f, \operatorname{lc} g) \bmod p\big)$;
   c) *replace the polynomial H by a polynomial obtained after applying Algorithm 2.2.6 to coefficients of H, H and moduli m, p;*
   d) *update m setting $m := m \cdot p$;*
4. *replace H by its primitive part;*
5. *replace each coefficient H_i of H greater than $m/2$ by $H_i - m$;*
6. *return* $\gcd(f, g) = H$.

Example 31. Take the two polynomials

$$f = x^6 + 2x^5 - 4x^4 + 11x^2 - 9x + 2,$$
$$g = 7x^5 - 11x^4 - 18x^3 + 48x^2 - 29x + 6,$$

that we have already encountered in Examples 9–11. Suppose that we know a priori that the resulting gcd has degree 1 and the absolute values of its coefficients are bounded by 5. (Of course in this particular case we know that since we have already computed this gcd three times.) It suffices to take just one prime $p = 23$, then

$$f_{23} = x^6 + 2x^5 + 7x^4 + 2x + 2,$$
$$g_{23} = 7x^5 + 4x^3 + 4x^2 + 4x + 6.$$

Applying Euclidean algorithm and selecting the monic representative of the gcd (since $1 = \gcd(\operatorname{lc} f, \operatorname{lc} g)$, we have

$$\gcd(f_{23}, g_{23}) = x + 2 \in \mathbb{F}_{23}[x].$$

Consequently also $\gcd(f, g) = x + 2 \in \mathbb{Z}[x]$.

Suppose, however, that we do not know the bound for the absolute values of the coefficients of the greatest common divisor of f and g is 5, but instead we estimate it to be 5000. Then we have to continue our computations taking primes: $13, 17, 19, 23$, while the resulting polynomial H stays to be equal $x + 2$ through all these steps.

The last part of the example suggests a better stopping criterion for the loop in step 3 of the algorithm Rather than checking if $m \geq 2M$, we need just to check whether H divides both f and g. If it does, then it divides their greatest common divisor, too. And since the degree of both polynomials are equal, H must be the sought gcd. This modification saves us from the need to search for a good bound on the absolute value of the coefficients of the greatest common divisor, given only the input polynomials f and g. Although there are such bounds (e.g. Landau–Mignotte theorem), we do not need them here.

We have already managed to free ourselves from the need to know a priori a bound for the size of coefficients of gcd. Can we also release the requirement to know a

prori the degree of the greatest common divisor? In fact we can. The idea is to keep calculating modular gcds with whatever prime we have at our disposal, omitting those that either divide one of the leading coefficients or result in gcd with a bigger degree than already computed. If for any prime p, $\gcd(f_{(p)}, g_{(p)})$ have a strictly smaller degree than our previously computed gcd, we must discard all the previous computation and start over. Otherwise if the degrees agree, we reconstruct the greatest common divisor using Chinese remainder theorem. All in all, the resulting algorithm has the following form.

Algorithm 2.5.2. *Given two polynomials f, g with integer coefficients, this algorithm returns their greatest common divisor.*

1. *Let d be the greatest common divisor of the contents of f and g, i.e. $d = \gcd\big(\operatorname{cont}(f), \operatorname{cont}(g)\big)$;*
2. *replace f and g by their respective primitive parts;*
3. *pick any prime p not dividing $\operatorname{lc} f$, $\operatorname{lc} g$ and initialize $m := p$;*
4. *let H be a lift to $\mathbb{Z}[x]$ of $\gcd(f_{(p)}, g_{(p)}) \in \mathbb{F}_p[x]$, where the representative of $\gcd(f_{(p)}, g_{(p)})$ is selected in such a way that its leading coefficient equals $\big(\gcd(\operatorname{lc} f, \operatorname{lc} g) \bmod p\big)$;*
5. *set $\hat{H}$ to be a polynomial obtained from H by replacing each coefficient H_i greater than $m/2$ by $H_i - m$;*
6. *while $\hat{H}$ does not divide either f or g, repeat the following steps:*
   a) *pick a new random prime p such that*
   $$p \nmid \operatorname{lc} f, \qquad p \nmid \operatorname{lc} g \qquad \text{and} \qquad \deg \gcd(f_{(p)}, g_{(p)}) \leq \deg H;$$
   b) *let h be a lift to $\mathbb{Z}[x]$ of $\gcd(f_{(p)}, g_{(p)} \in \mathbb{F}_p[x]$, where again $\operatorname{lc} \gcd(f_{(p)}, g_{(p)}) = \big(\gcd(\operatorname{lc} f, \operatorname{lc} g) \bmod p\big)$; if $\deg h < \deg H$, then discard all previous computations and set*
   $$m := 1 \qquad \text{and} \qquad H := h;$$
   c) *otherwise, if $\deg h = \deg H$, then update H, replacing it by a polynomial obtained by applying Chinese remainder theorem (Algorithm 2.2.6) to coefficients of H and h and moduli m, p;*
   d) *update m setting $m := m \cdot p$;*
   e) *update the candidate for gcd, setting $\hat{H}$ to be the primitive part of a polynomial obtained from H by replacing each coefficient H_i greater than $m/2$ by $H_i - m$;*
7. *return $\gcd(f, g) = \hat{H} \cdot d$.*

Arithmetic in finite field extensions

Missing

We now prove the following generalization of Fermat's little theorem (Theorem 2.3.2).

Theorem 2.5.3. *If $\mathbb{F}_q$ is a finite field ($q = p^n$), then for every $a \in \mathbb{F}_q$ we have*

$$a^{q-1} = \begin{cases} 0 & \text{if } a = 0, \\ 1 & \text{if } a \neq 0. \end{cases}$$

Proof. The multiplicative group $\mathbb{F}_q^* = \mathbb{F}_q \setminus \{0\}$ of $\mathbb{F}_q$ has $q-1$ elements. Lagrange theorem (see e.g. [51, Proposition I.4.1]) asserts that the order of any element of a group divides the order of the whole group. Hence we have $a^{q-1} = 1$ for every nonzero $a \in \mathbb{F}_q$. □

It follows from the above theorem that $a^q - a = 0$ and this equality holds for every $a \in \mathbb{F}_q$, including $a = 0$. Thus, every $a \in \mathbb{F}_q$ is a root of a polynomial $x^q - x$ and so $x - a$ divides $x^q - x$ by little Bézout theorem (Proposition 1.1.18). Comparing the degrees we prove:

Corollary 2.5.4. *With the above notation*

$$x^q - x = \prod_{a \in \mathbb{F}_q} (x - a).$$

Bibliographic notes and further reading

Proof of the quadratic reciprocity law (Theorem 2.4.7) is based on [74].

Proposition 2.4.21 is well known and its origin are hard to track. It was probably known already to C. Gauss. On the other hand, Proposition 2.4.22 is quite recent. It was presented for the first time in [6]. Since then there appeared some more results of similar form. In 2004 (see [65]) S. Müller introduced a method of computing square rots in $\mathbb{F}_p$ for $p \equiv 9 \pmod{16}$. His algorithm was improved two years later in [44]. Even more recently Koo et al. [45] presented a general method for generating "Atkin's style" closed-form square-root formulas for $p \equiv 2^s + 1 \pmod{2^{s+1}}$. In particular authors derived formulas for $p \equiv 5 \pmod 8$, $9 \pmod{16}$ and $17 \pmod{32}$.

Modular gcd algorithm (Algorithm 2.5.2) is based on [13].

3

Matrices

The Matrix is everywhere. It is all around us.

"Matrix" (movie, 1999)

3.1 Matrix arithmetic

The next class of objects we wish to consider is the class of matrices. The sum (respectively the difference) of matrices is formed by adding (resp. subtracting) entries with the same indices and so the corresponding algorithms are nearly identical to addition and subtraction of polynomials. In fact, for matrices, the situation is even a tiny bit simpler, since one can add polynomials of different degrees, but when summing matrices, the dimensions must be equal.

Algorithm 3.1.1. *Given two matrices $A = (a_{ij})$, $B = (b_{ij})$ of equal dimensions, this algorithm computes their sum $C = A + B$ (respectively difference $C = A - B$).*

1. *Let m denote the number of rows and n the number of columns in A and B;*
2. *for every $i \in \{1, \ldots, m\}$ and every $j \in \{1, \ldots, n\}$ set*

$$c_{ij} := a_{ij} + b_{ij} \qquad (\text{respectively } c_{ij} := a_{ij} - b_{ij});$$

3. *return a matrix $C = (c_{ij})$.*

The above algorithm requires $m \cdot n$ arithmetic operations in the base ring, where m is the number of rows and n the number of columns of A, B. It is clear that the algorithm is optimal as the number of operations equals the number of elements and each element of A and B must be accessed at least one. Let us now turn our attention to matrix multiplication. As we know from a basic course of linear algebra, two matrices A

and B can be multiplied if and only if the number of columns of the left operand equals the number of rows of the right one. The product consists of all dot products of rows of A and columns of B.

Algorithm 3.1.2. *Let $A=(a_{ij})$ be a matrix of dimension $m\times l$ and $B=(b_{ij})$ be a matrix of dimension $l\times n$. This algorithm computes their product $C=A\cdot B$.*

1. *For every $i\in\{1,\ldots,m\}$ and every $j\in\{1,\ldots,n\}$ compute*

$$c_{ij}:=\sum_{k=1}^{l}a_{ik}b_{kj};$$

2. *return a matrix*

$$C=\begin{pmatrix}c_{11} & \cdots & c_{1n}\\ \vdots & & \vdots\\ c_{m1} & \cdots & c_{mn}\end{pmatrix}.$$

This standard multiplication scheme requires l scalar multiplications (and $l-1$ additions) per an entry of the result. Hence time complexity of this algorithm is $O(l\cdot m\cdot n)$, or shortly $O(n^3)$ for square $n\times n$ matrices. In the earlier sections of this chapter we observed that time complexity of integer (respectively polynomial) multiplication can be substantially reduced by successive subdivisions. This was an idea of Karatsuba's divide-and-conquer algorithm (cf. Algorithms 1.1.9). It turns out that we can can approach a problem of matrix multiplication along a similar line, without however such a splendid success.

Consider a product of two 2×2 matrices $A=(a_{ij})$ and $B=(b_{ij})$:

$$\begin{pmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\end{pmatrix}\cdot\begin{pmatrix}b_{11} & b_{12}\\ b_{21} & b_{22}\end{pmatrix}=\begin{pmatrix}a_{11}b_{11}+a_{12}b_{21} & a_{11}b_{12}+a_{12}b_{22}\\ a_{21}b_{11}+a_{22}b_{21} & a_{21}b_{12}+a_{22}b_{22}\end{pmatrix}.$$

The above formula uses four scalar multiplications. Transform the expressions in top-right and bottom-left corners as follows:

$$\begin{aligned}a_{11}b_{12}+a_{12}b_{22}&=a_{11}\cdot(b_{12}-b_{22})+(a_{11}+a_{12})\cdot b_{22}\\ a_{21}b_{11}+a_{22}b_{21}&=(a_{21}+a_{22})\cdot b_{11}+a_{22}\cdot(b_{21}-b_{11}).\end{aligned}$$

The terms that appear on the right hand sides can now be reused to compute the elements at the top-left and bottom-right corners of the product $A\cdot B$. Indeed,

$$\begin{aligned}a_{11}b_{11}+a_{12}b_{21}&=a_{22}\cdot(b_{21}-b_{11})-(a_{11}+a_{12})\cdot b_{22}\\ &\quad+(a_{11}+a_{22})\cdot(b_{11}+b_{22})+(a_{12}-a_{22})\cdot(b_{21}+b_{22})\\ a_{21}b_{12}+a_{22}b_{22}&=a_{11}\cdot(b_{12}-b_{22})-(a_{21}+a_{22})\cdot b_{11}\\ &\quad+(a_{11}+a_{22})\cdot(b_{11}+b_{22})+(a_{21}-a_{11})\cdot(b_{11}+b_{12}).\end{aligned}$$

Thus, we have managed to compute all entries of the product $A\cdot B$ using only 7 (instead of 8) scalar multiplications. What is even more important, we did it without assuming

commutativity of the matrix entries. Therefore, we may use these formulas for block matrices. The following method is due to V. Strassen.

Let A be a matrix with $2m$ rows and $2l$ columns and B be a matrix with $2l$ rows and $2n$ columns. Denote their product by C. Subdivide all three matrices into blocks of dimensions respectively $m \times l$, $l \times n$ and $m \times n$:

$$A = \left(\begin{array}{c:c} A_{11} & A_{12} \\ \hdashline A_{21} & A_{22} \end{array}\right), \qquad B = \left(\begin{array}{c:c} B_{11} & B_{12} \\ \hdashline B_{21} & B_{22} \end{array}\right), \qquad C = \left(\begin{array}{c:c} C_{11} & C_{12} \\ \hdashline C_{21} & C_{22} \end{array}\right). \tag{3.1}$$

Define new matrices:

$$\begin{aligned} D_1 &:= (A_{11} + A_{22}) \cdot (B_{11} + B_{22}), & D_2 &:= (A_{21} + A_{22}) \cdot B_{11}, \\ D_3 &:= A_{11} \cdot (B_{12} - B_{22}), & D_4 &:= A_{22} \cdot (B_{21} - B_{11}), \\ D_5 &:= (A_{11} + A_{12}) \cdot B_{22}, & D_6 &:= (A_{21} - A_{11}) \cdot (B_{11} + B_{12}), \\ D_7 &:= (A_{12} - A_{22}) \cdot (B_{21} + B_{22}). \end{aligned} \tag{3.2}$$

Each definition involves only one multiplication. Exactly as for 2×2 matrices, we have

$$\begin{aligned} C_{11} &= D_1 + D_4 - D_5 + D_7, & C_{12} &= D_3 + D_5, \\ C_{21} &= D_2 + D_4, & C_{22} &= D_1 - D_2 + D_3 + D_6. \end{aligned}$$

This way we computed all blocks C_{ij} with only 7 block products (and 18 additions/-subtractions). Subdividing recursively D_i's one computes all the products needed.

Algorithm 3.1.3 (Strassen)**.** *Given two matrices A and B of compatible dimensions, this algorithm computes their product.*

1. *If any the dimensions of A and B are lower than a certain threshold d, then compute the product $C = A \cdot B$ using Algorithm 3.1.2, return the result and quit;*
2. *if any of the dimensions of A and B is odd, then conceptually append a column and/or row of zeros ;*
3. *subdivide the matrices A, B as in Eq.* (3.1)*;*
4. *construct matrices $D_1, \ldots, D_7$ in Eq.* (3.2) *using this algorithm recursively for multiplications;*
5. *construct and return a matrix*

$$C = \begin{pmatrix} D_1 + D_4 - D_5 + D7 & D_3 + D_5 \\ D_2 + D_4 & D_1 - D_2 + D_3 + D_6 \end{pmatrix}.$$

Remark 28. In actual implementation, one does not really appends any zeros to a matrix in step 2 instead, when constructing D_i's, one just adds/subtracts a smaller matrix to/from the top-left corner of the other matrix.

Complexity analysis: To simplify the analysis, assume that we are multiplying two square $n \times n$ matrices. The depth of recursion in Algorithm 3.1.3 is clearly $\lceil \log_2 n \rceil$, since matrices are subdivided in half with respect to both dimensions. At each recursion level, multiplication branches into 7 multiplications of submatrices. Thus we end up with

$$7^{\lceil \log_2 n \rceil} \leq 7 \cdot n^{\log_2 7} \approx 7 \cdot n^{2.807}$$

scalar products.

Observation 3.1.4. *The time complexity of Strassen multiplication is* $\mathcal{O}\left(n^{\log_2(7)}\right) \approx O(n^{2.807})$*, providing that scalar-multiplication is a dominating operation.*

Remark 29. The starting point for the derivation of the above algorithm was Strassen's formula for a product of 2×2 matrices. There exist also other formulas for products of small matrices that use fewer scalar multiplications than required by the standard matrix-product formula. Some of them are mentioned in 'Further reading' section at the end of this chapter.

3.2 Gauss elimination

Introduction

A process of transforming an arbitrary matrix to a "nicer" form (so called *row-echelon form*), known as *Gaussian elimination*, is definitely the most important and most versatile matrix, that we shall discuss in this book. It is used for: solving systems of linear equations, computing determinants, inverting matrices and many more tasks. Even Buchberger's algorithm for constructing a system of generators (so called *Gröbner basis*) of an ideal in a multivariate polynomial ring, that we will meet in Chapter 9, can be viewed as a certain generalization of Gaussian elimination.

We first explain the process on a simple example. Consider a system of linear equations.

$$\begin{cases} x - 2y + 3z = 2 \\ 2x + y - z = (-1) \\ x + y - 2z = (-4). \end{cases}$$

In matrix notation the system can be shorty expressed as $A \cdot X = b$, where A is a matrix of coefficients, $X = (x, y, z)^T$ is a vector of variables and b is a vector of constant terms. The *augmented matrix* of this system has a form

$$\left(A \,\middle|\, b\right) = \left(\begin{array}{ccc|c} 1 & -2 & 3 & 2 \\ 2 & 1 & -1 & -1 \\ 1 & 1 & -2 & -4 \end{array}\right).$$

We wish to eliminate variable x from all but the first equations. To this end we substract the first equation twice from the second one and once from the last one. This of course

does not alter the set of solutions. Now we have

$$\begin{cases} x-2y+3z=2 \\ 5y-7z=(-5) \\ 3y-5z=(-6) \end{cases} \qquad \left(\begin{array}{ccc|c} 1 & -2 & 3 & 2 \\ 0 & 5 & -7 & -5 \\ 0 & 3 & -5 & -6 \end{array}\right).$$

Any equation can be harmlessly multiplied by a nonzero constant. In particular, if we multiply the second one by $1/5$ we obtain

$$\begin{cases} x-2y+3z=2 \\ y-\frac{7}{5}z=(-1) \\ 3y-5z=(-6) \end{cases} \qquad \left(\begin{array}{ccc|c} 1 & -2 & 3 & 2 \\ 0 & 1 & -\frac{7}{5} & -1 \\ 0 & 3 & -5 & -6 \end{array}\right).$$

We can now eliminate y from the two other equations, adding the second equation twice to the first one and subtracting it thrice from the last one:

$$\begin{cases} x+\frac{1}{5}z=0 \\ y-\frac{7}{5}z=(-1) \\ -\frac{4}{5}z=(-3) \end{cases} \qquad \left(\begin{array}{ccc|c} 1 & 0 & \frac{1}{5} & 0 \\ 0 & 1 & -\frac{7}{5} & -1 \\ 0 & 0 & -\frac{4}{5} & -3 \end{array}\right).$$

Finally, multiply the last equation by $-5/4$ and use the result to eliminate z from the other two equations. The gives us

$$\begin{cases} x=\left(-\frac{3}{4}\right) \\ y=\left(\frac{17}{4}\right) \\ z=\left(\frac{15}{4}\right) \end{cases} \qquad \left(\begin{array}{ccc|c} 1 & 0 & 0 & -\frac{3}{4} \\ 0 & 1 & 0 & \frac{17}{4} \\ 0 & 0 & 1 & \frac{15}{4} \end{array}\right).$$

The resulting matrix is “nicer” than the original one, since we can simply read the solutions out of it.

In the above example. we derived the final form of a matrix using only two operations, namely:

- multiplying a row by a nonzero scalar,
- adding to one row a scalar multiple of another one.

Observe that it is not always possible to proceed from top to bottom as we did above. If for example the top-left element of a given matrix is null, we cannot use the first row to eliminate the elements in the first column, since that would lead to a division by zero. Thus, beside the two operations that we used in the example, we need one more:

- rearranging the order of rows.

It turns out that these three operations suffice.

Echelon and elementary matrices

In what follows, given a matrix $M = \left(m_{ij}\right)$ of size $m \times n$, by M_i we shall denote its i-th row, $M_i = (m_{i1}, \ldots, m_{in})$.

Definition 3.2.1. Let M be a matrix. The left-most nonzero element of a row M_j of M is called the *pivot* of M_j.

Remark 30. If we identify a row of a matrix with a vector of coefficients of a multivariate linear polynomial, then the pivot is the leading coefficient of this polynomial with respect to the *lexicographic order* of monomial. We will say more about this correspondence in Chapter 9.

Definition 3.2.2. A matrix M of size $m \times n$ is said to be an *echelon matrix* (or a matrix in an *echelon form*), if:

- the nonzero rows of M precede the zero ones,
- if M_i, M_j are two nonzero rows and $i < j$, then the pivot of M_i is to the left of the pivot of M_j.

Definition 3.2.3. An echelon matrix M is said to be in *reduced* echelon form if all the pivots equal 1 and each pivot is the only nonzero element of its column.

Example 32. Consider two matrices:

$$\begin{pmatrix} 0 & 3 & 1 & 2 & -1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & -2 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & 5 \end{pmatrix}.$$

The first of them is an echelon matrix but not reduced. The second one is a reduced echelon matrix.

If M is a non-singular square matrix, then it is in an echelon form if and only if it is upper triangular (with arbitrary nonzero elements on the main diagonal). A non-singular square matrix is in a reduce echelon form if and only if it is just the identity matrix.

We now introduce three special types of matrices of size $k \times k$:

- matrix $E_1^k(i, j)$ is constructed from the identity matrix I_k by swapping i-th row and j-th row:

$$E_1^k(i,j) = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 0 & & 1 & & \\ & & & \ddots & & & \\ & & 1 & & 0 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix} \begin{matrix} \\ \\ i \\ \\ j \\ \\ \\ \end{matrix}$$

- matrix $E_2^k(i;c)$ is like the identity matrix I_k but the i-th element on the main diagonal is c instead of 1:

$$E_2^k(i;\alpha) = \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & & \alpha & & \\ & & & & 1 & \\ & & & & & \ddots \\ & & & & & & 1 \end{pmatrix} \quad i$$

- matrix $E_3^k(i,j;\alpha)$ is like the identity matrix I_k but the element at the position (i,j) equals c instead of 0:

$$E_3^k(i,j;\alpha) = \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & & \ddots & & \\ & & \alpha & & 1 & \\ & & & & & \ddots \\ & & & & & & 1 \end{pmatrix}$$

The matrices of the above three types are called *elementary*. The name comes from the fact that the three elementary matrix transformations described in the previous subsection are obtained by multiplication of a given matrix with a matrix of one of the above types. The following observation is immediate.

Observation 3.2.4. *Let M be a matrix of size $m \times n$, then*

- *$E_1^m(i,j) \cdot M$ is a matrix obtained from M by swapping the positions of the i-the and j-th rows.*
- *$E_2^m(i;\alpha) \cdot M$ is a matrix obtained from M by multiplying its i-th row by α.*
- *$E_3^m(i,j;\alpha) \cdot M$ is a matrix obtained from M by replacing the row M_i with $M_i + \alpha M_j$.*

Gauss and Gauss–Jordan algorithms

For the sake of clarity, in the rest of this section we assume that the maximal principal minor of the matrix we consider is nonzero (i.e. the maximal square sub-matrix aligned to the top-left corner is non-singular). This means that the pivots of the echelon form occupy consecutive columns, i.e. they lie on the main diagonal. It is possible, and in fact quite simple, to generalize Algorithms 3.2.5–3.2.8 so that they work with arbitrary matrices, but the resulting pseudo-codes are less readable. Therefore, in this case the generality should give way to clarity. Nevertheless, we encourage the reader to generalize all three algorithms and make them work without the above assumption.

The first algorithm we present is the "classical" Gaussian elimination. It transform a matrix to the echelon form.

Algorithm 3.2.5 (Gaussian elimination)**.** *Let $M = (m_{ij})$ be a matrix with m rows and n columns. Assume that its maximal principal minor is nonzero. This algorithm returns an echelon matrix $\tilde{M}$ obtained from M together with a list $\mathscr{L} = (E_1, \ldots, E_k)$ of elementary matrices such that $\tilde{M} = E_k \cdots E_1 \cdot M$.*

1. *Let* $d := \min\{m, n\}$ *be the minimum of the number of rows and the number of columns of* M*;*
2. *initialize* $\mathscr{L}$ *to be an empty list;*
3. *loop over the rows* M_k *of* M *for* $k \in \{1, \ldots, d\}$*:*
   a) *find the pivot* $m_{jk} \neq 0$ *with* $j \geq k$ (*see Remark 31 below*)*;*
   b) *if no such* m_{jk} *exists, then* M *does not satisfy our assumption, hence throw an exception and quit;*
   c) *if* $j \neq k$*, then swap the rows* M_k *and* M_j *and append* $E_1^m(k, j)$ *to the list* $\mathscr{L}$*.*
   d) *for every row* M_j *below* M_k*, i.e. for* $j \in \{k+1, \ldots, m\}$*:*
      i. *set* $c := {}^{m_{jk}}/_{m_{kk}}$*;*
      ii. *replace the row* M_j *by* $M_j - c \cdot M_k$ *and append* $E_3^m(j, k; -c)$ *to* $\mathscr{L}$*;*
4. *return* $\tilde{M} = M$ *and* $\mathscr{L}$*.*

The correctness of the algorithm is clear, because replacing (in the inner loop) rows M_j by $M_j - {}^{m_{jk}}/_{m_{kk}} \cdot M_k$ for $j > k$, we make sure that all the elements under m_{kk} are cleared. Elements to the left of the k-th column are already zero and they remain so, since m_{kk} is a pivot. Thus, after d iteration of the main loop the matrix M is in an echelon form.

Complexity analysis: Let M be a square matrix of size $n \times n$. The outer loop is repeated n times. During the k-th iteration of the outer loop, the inner loop is executed $(n-k)$ times. Inside the inner loop we do one scalar division, one scalar-vector multiplication and one vector subtraction, hence at most $1 + 2n$ scalar operations[1]. Consequently the complexity of the algorithm is bounded by

$$\sum_{k=1}^{n} (1 + 2n) \cdot (n - k) = \frac{n^3 - n^2 - n}{2} = O(n^3).$$

A simple modification of the above algorithm, known as Gauss–Jordan elimination, computes the reduced echelon matrix. To this end, the transformations of the form $M_j \leftarrow M_j - c\dot{M}_k$ are applied not only to the rows below M_k but also to the ones above it. Moreover, each row has to be divided by its pivot.

Algorithm 3.2.6 (Gauss–Jordan elimination)**.** *Let* $M = (m_{ij})$ *be a matrix with* m *rows and* n *columns. Assume that its maximal principal minor is nonzero. This algorithm returns a reduced echelon matrix* $\tilde{M}$ *obtained from* M *together with a list* $\mathscr{L} = (E_1, \ldots, E_k)$ *of elementary matrices such that* $\tilde{M} = E_k \cdots E_1 \cdot M$*.*

1. *Let* $d := \min\{m, n\}$ *be the minimum of the number of rows and the number of columns of* M*;*

[1] One can possibly exploit the fact that elements $m_{j,1}, \ldots, m_{j,k-1}$ are already known to be zero for $j \geq k$. This may allow to reduce the actual number of scalar operations on each row. This will not change the asymptotic complexity, though.

2. *initialize $\mathscr{L}$ to be an empty list;*
3. *loop over the rows M_k of M for $k \in \{1, \ldots, d\}$:*
   a) *find the pivot $m_{jk} \neq 0$ with $j \geq k$ (see Remark 31 below);*
   b) *if no such m_{jk} exists, then M does not satisfy our assumption, hence throw an exception and quit;*
   c) *if $j \neq k$, then swap the rows M_k and M_j and append $E_1^m(k, j)$ to the list $\mathscr{L}$.*
   d) *multiply the row M_k by $1/m_{kk}$ and append $E_2^m(k; 1/m_{kk})$ to the list $\mathscr{L}$;*
   e) *for $j \in \{1, \ldots, k-1, k+1, m\}$:*
      i. *replace the row M_j by $M_j - m_{jk} \cdot M_k$ and append $E_3^m(j, k; -m_{jk})$ to $\mathscr{L}$;*
4. *return $\tilde{M} = M$ and $\mathscr{L}$.*

The correctness of this algorithm is obvious is is the fact that the asymptotic complexity as the same as in the case of Algorithm 3.2.5.

Remark 31. The strategy for selecting a pivot in step (3a) in above algorithms, heavily depends on what kind of objects the coefficients are. For floating-point numbers, it is convenient to select an element with a maximal absolute value among all the entries on and under the diagonal in k-th column:

$$|m_{jk}| = \max\{|m_{ik}| \,|\, k \leq i \leq m\}$$

On the other hand, for exact types (rational numbers, rational functions), it is usually better to select a nonzero element with a minimal length of the representation.

Integer-preserving elimination

An annoying feature of Gauss elimination is that it uses division. This is not an issue if we are working over finite fields (see the next chapter) but in other cases it can cause problems. It may introduce round-off errors, when working with floating-point numbers. On the other hand, for exact types, like integers and polynomials (especially multivariate polynomials), Gaussian elimination introduces fractions and may lead to an explosive growth of size of coefficients in intermediate steps. Thus, there is a need for elimination algorithms that either use exact division only or avoid it completely. In this subsection we discuss a variant of an elimination algorithm that uses only exact divisions.

The assumption that the maximal principal minor is nonzero is still used throughout this section. Moreover, we shall use superscript to enumerate iterations of the main loop of the elimination procedure. Hence, from now on, $M^{(0)} := M$ denotes the initial matrix, $M^{(1)}$ is the matrix obtained after one iteration of the loop and so on, till we reach $M^{(d)} = \tilde{M}$, the sough echelon matrix. This notation applies also to rows and elements of the matrix, so that $M_i^{(k)}$ is the i-th row of $M^{(k)}$ and $m_{ij}^{(k)}$ is the j-th entry of $M_i^{(k)}$. Additionally we denote $m_{00}^{(-1)} := 1$.

With this convention, the core of Gauss elimination can be written as

$$M_i^{(k)} := M_i^{(k-1)} - \frac{m_{ik}^{(k-1)}}{m_{kk}^{(k-1)}} \cdot M_k^{(k-1)} \qquad \text{for } k \geq 1 \text{ and } i > k.$$

If we wish to avoid division, we should get rid of the denominator and use a row transformation of a form:

$$M_i^{(k)} := m_{kk}^{(k-1)} \cdot M_i^{(k-1)} - m_{ik}^{(k-1)} \cdot M_k^{(k-1)}. \tag{3.3}$$

It is obvious that this transformation still clears elements under the pivot $m_{kk}^{(k-1)}$. Thus, it leads to a division-free variant of Gauss elimination. Unfortunately, in this variant of the elimination algorithm the entries of the intermediate matrices can grow out of control. The following lemma explains the reason for this phenomenon (see the "factorial-like" formula of the divisor) as well as provides a solution.

Lemma 3.2.7. *Let $M = \left(m_{ij}\right)$ be a $m \times n$ matrix with integer coefficients* (*more generally: with elements from some integral domain*). *Assume that matrices $M^{(1)}, \ldots, M^{(k)}$ are constructed using Eq.* (3.3). *Then all the entries of $M_i^{(k)}$ are divisible by $m_{0,0}^{(-1)} \cdot m_{1,1}^{(0)} \cdots m_{k-1,k-1}^{(k-2)}$.*

Proof. We proceed by induction on k. For $k = 1$ the assertion holds trivially since $m_{00}^{(-1)} = 1$. Fix now $k > 1$ and take an element $m_{ij}^{(k)}$ with $i > k$. By Eq. (3.3) we have

$$m_{ij}^{(k)} = m_{kk}^{(k-1)} \cdot m_{ij}^{(k-1)} - m_{ik}^{(k-1)} \cdot m_{kj}^{(k-1)}$$

Apply formula (3.3) to every element on the right-hand side

$$\begin{aligned} &= \left(m_{k-1,k-1}^{(k-2)} m_{k,k}^{(k-2)} - m_{k-1,k}^{(k-2)} m_{k,k-1}^{(k-2)}\right) \cdot \left(m_{k-1,k-1}^{(k-2)} m_{i,j}^{(k-2)} - m_{k-1,j}^{(k-2)} m_{i,k-1}^{(k-2)}\right) \\ &\quad - \left(m_{k-1,k-1}^{(k-2)} m_{i,k}^{(k-2)} - m_{k-1,k}^{(k-2)} m_{i,k-1}^{(k-2)}\right) \cdot \left(m_{k-1,k-1}^{(k-2)} m_{k,j}^{(k-2)} - m_{k-1,j}^{(k-2)} m_{k,k-1}^{(k-2)}\right) \end{aligned}$$

Expand the products and observe that some terms cancel

$$\begin{aligned} &= \left(m_{k-1,k-1}^{(k-2)}\right)^2 m_{i,j}^{(k-2)} m_{k,k}^{(k-2)} - m_{k-1,k-1}^{(k-2)} m_{k,k}^{(k-2)} m_{k-1,j}^{(k-2)} m_{i,k-1}^{(k-2)} \\ &\quad - m_{k-1,k-1}^{(k-2)} m_{k-1,k}^{(k-2)} m_{k,k-1}^{(k-2)} m_{i,j}^{(k-2)} + \cancel{m_{k-1,k}^{(k-2)} m_{k,k-1}^{(k-2)} m_{k-1,j}^{(k-2)} m_{i,k-1}^{(k-2)}} \\ &\quad - \left(m_{k-1,k-1}^{(k-2)}\right)^2 m_{i,k}^{(k-2)} m_{k,j}^{(k-2)} + m_{k-1,k-1}^{(k-2)} m_{k,j}^{(k-2)} m_{k-1,k}^{(k-2)} m_{i,k-1}^{(k-2)} \\ &\quad + m_{k-1,k-1}^{(k-2)} m_{k-1,j}^{(k-2)} m_{k,k-1}^{(k-2)} m_{i,k}^{(k-2)} - \cancel{m_{k-1,j}^{(k-2)} m_{k,k-1}^{(k-2)} m_{k-1,k}^{(k-2)} m_{i,k-1}^{(k-2)}} \end{aligned}$$

All the remaining products are divisible by $m_{k-1,k-1}^{(k-2)}$ and we may write

$$\begin{aligned} &= m_{k-1,k-1}^{(k-2)} \cdot \Big(m_{k-1,k-1}^{(k-2)} \mathbf{m_{i,j}^{(k-2)}} m_{k,k}^{(k-2)} - m_{k,k}^{(k-2)} m_{k-1,j}^{(k-2)} \mathbf{m_{i,k-1}^{(k-2)}} - m_{k-1,k}^{(k-2)} m_{k,k-1}^{(k-2)} \mathbf{m_{i,j}^{(k-2)}} \\ &\quad - m_{k-1,k-1}^{(k-2)} \mathbf{m_{i,k}^{(k-2)}} m_{k,j}^{(k-2)} + m_{k,j}^{(k-2)} m_{k-1,k}^{(k-2)} \mathbf{m_{i,k-1}^{(k-2)}} + m_{k-1,j}^{(k-2)} m_{k,k-1}^{(k-2)} \mathbf{m_{i,k}^{(k-2)}} \Big). \end{aligned}$$

By the inductive hypothesis, the terms in bold are divisible by $m_{0,0}^{(-1)} \cdots m_{k-2,k-2}^{(k-3)}$. This proves the lemma. □

With the aim of the above result we may refine the row transformation. Instead of Eq. (3.3) we shall use the following formula (recall that $m_{00}^{(-1)} = 1$ by definition):

$$M_i^{(k)} := \frac{1}{m_{k-1,k-1}^{(k-2)}} \cdot \Big(m_{kk}^{(k-1)} \cdot M_i^{(k-1)} - m_{ik}^{(k-1)} \cdot M_k^{(k-1)} \Big). \tag{3.4}$$

The previous lemma ensures that the division is exact. We leave it as an exercise for the reader to explain why we divide only by $m_{k-1,k-1}^{(k-2)}$ and not the full product $m_{0,0}^{(-1)} \cdots m_{k-1,k-1}^{(k-2)}$. (Hint: this is a recursive formula.) We may now present a fraction-free variant of the elimination algorithm. All operations in the algorithm are assumed to be performed in situ, hence we drop the superscript and use just the matrix M all the time.

Algorithm 3.2.8 (Jordan–Bareiss integer-preserving elimination)**.** *Let $M = (m_{ij})$ be a matrix with m rows and n columns. Assume that its maximal principal minor is nonzero. This algorithm returns an echelon matrix $\tilde{M}$ obtained from M together with a list $\mathscr{L} = (E_1, \ldots, E_k)$ of elementary matrices such that $\tilde{M} = E_k \cdots E_1 \cdot M$.*

1. *Let $d := \min\{m, n\}$ be the minimum of the number of rows and the number of columns of M;*
2. *initialize $\mathscr{L}$ to be an empty list and set $c := m_{00}^{(-1)} = 1$, this variable will store the previous pivot;*
3. *loop over the rows M_k of M for $k \in \{1, \ldots, d\}$:*
   a) *find the pivot $m_{ik} \neq 0$ with $i \geq k$;*
   b) *if no such m_{ik} exists, then M does not satisfy our assumption, hence throw an exception and quit;*
   c) *if $i \neq k$, then swap the rows M_k and M_i and append $E_1^m(k, i)$ to the list $\mathscr{L}$.*
   d) *for every row M_i below M_k, i.e. for $i \in \{k+1, \ldots, m\}$:*
      i. *replace the row M_i by*

$$\frac{1}{c} \cdot \big(m_{kk} \cdot M_i - m_{ik} \cdot M_k \big);$$

      ii. *append the following three matrices to the list $\mathscr{L}$:*

$$E_2^m(i; m_{kk}), \qquad E_3^m(i, k; -m_{ik}), \qquad E_2^m(i, 1/c);$$

   e) *update the variable c setting $c := m_{kk}$;*
4. *return $\tilde{M} = M$ and $\mathscr{L}$.*

The correctness of the algorithm follows from the discussion preceding it. Since the algorithm differs from standard Gauss elimination only by row transformation formula, it is clear that the asymptotic complexity of the presented algorithm is also cubic. In the next section, while discussing the computation of a determinant, we show an example of Jordan–Bareiss elimination.

Remark 32. It is convenient to rewrite the row transformation of Jordan–Bareiss algorithm in a slightly different form that is more compact and easier to remember. This utilizes the notion of a determinant. Although we talk about the matrix determinant in the next section, but we assume that the reader is already familiar with it or at the very least knows how to compute the determinant of a 2×2 matrix. We have

$$\begin{aligned} m_{i,j}^{(k)} &= \frac{1}{m_{k-1,k-1}^{(k-2)}} \cdot \left(m_{k,k}^{(k-1)} \cdot m_{i,j}^{(k-1)} - m_{i,k}^{(k-1)} \cdot m_{k,j}^{(k-1)} \right) \\ &= \frac{1}{m_{k-1,k-1}^{(k-2)}} \cdot \det \begin{pmatrix} m_{k,k}^{(k-1)} & m_{k,j}^{(k-1)} \\ m_{i,k}^{(k-1)} & m_{i,j}^{(k-1)} \end{pmatrix}. \end{aligned}$$

The 2×2 matrix appearing in the above formula consists of the four corners of a submatrix of $M^{(k-1)}$ determined by the positions of the pivot $m_{kk}^{(k-1)}$ and the element $m_{ij}^{(k-1)}$ being updated:

$$\begin{pmatrix} & \vdots & & \vdots & \\ \cdots & m_{kk}^{(k-1)} & \cdots & m_{kj}^{(k-1)} & \cdots \\ & \vdots & & \vdots & \\ \cdots & m_{ik}^{(k-1)} & \cdots & m_{ij}^{(k-1)} & \cdots \\ & \vdots & & \vdots & \end{pmatrix}.$$

Matrix inversion as an application of Gauss elimination

3.3 Matrix decomposition

QR decomposition, LU decomposition, polar decomposition (?), SVD

3.4 Determinant

Introduction

The determinant of a square matrix M of size $n \times n$ with entries m_{ij}, $1 \leq i, j \leq n$ is defined by the formula

$$\det M := \sum_{\sigma} \operatorname{sgn}(\sigma) \cdot m_{1\sigma(1)} \cdots m_{n\sigma(n)}, \tag{3.5}$$

where the summation is taken over all the permutations of the set $\{1,\dots,n\}$. Here, $\operatorname{sgn}(\sigma)$ is the *signature* of the permutation and equals $(-1)^t$ with t being a number of transpositions in some decomposition of σ. An alternative formula is $(-1)^{n-m}$, where m is the number of cycles in a decomposition of σ. Indeed, the signature is multiplicative, hence if σ decomposes into disjoint cycles

$$\sigma = \big(s_{1,1},\dots,s_{1,k_1}\big)\cdots\big(s_{m,1},\dots,s_{m,k_m}\big),$$

then

$$\begin{aligned}\operatorname{sgn}(\sigma) &= \operatorname{sgn}\big(s_{1,1},\dots,s_{1,k_1}\big)\cdots\operatorname{sgn}\big(s_{m,1},\dots,s_{m,k_m}\big)\\ &= (-1)^{k_1-1}\cdots(-1)^{k_m-1}\\ &= (-1)^{n-m}.\end{aligned}$$

One of the fundamental properties of a determinant is its multiplicativity.

Theorem 3.4.1 (Cauchy)**.** *For any two square matrices M, N of the same size,*

$$\det(M\cdot N) = \det(M)\cdot\det(N).$$

The proof of this theorem may be found in virtually every linear algebra textbook (see for example [72, Theorem 3.20]), here we omit it to save some space. Explicit formulas exist for determinants of small matrices. They can be derived directly from the definition. For 1×1 and 2×2 matrices we have:

$$\det\big(m_{11}\big) = m_{11}, \qquad \det\begin{pmatrix} m_{11} & m_{12}\\ m_{21} & m_{22}\end{pmatrix} = m_{11}m_{22} - m_{12}m_{21}.$$

and for a 3×3 matrix:

$$\begin{aligned}\det\begin{pmatrix} m_{11} & m_{12} & m_{13}\\ m_{21} & m_{22} & m_{23}\\ m_{31} & m_{32} & m_{33}\end{pmatrix} &= m_{11}m_{22}m_{33} + m_{12}m_{23}m_{31} + m_{13}m_{21}m_{32}\\ &\quad - m_{11}m_{23}m_{32} - m_{12}m_{21}m_{33} - m_{13}m_{22}m_{31}.\end{aligned}$$

This last formula is called the *rule of Sarrus*[2]

A set with n elements admits $n!$ permutations. Therefore, the time complexity of computing a determinant directly by definition is $O(n!)$ and as such is prohibitive for large matrices. The following formula, named after P.-S. de Laplace[3], is usually taught at an introductory course of linear algebra and is sometimes used when computing a determinant by hand.

[2] Pierre Frédéric Sarrus (born 10 March 1798, died 20 November 1861) was a French mathematician.

[3] Pierre-Simon marquis de Laplace (born 23 March 1749, died 5 March 1827) was a French aristocrat and scientist. He worked in the field of mathematics, statistics, physics, and astronomy.

Theorem 3.4.2 (Laplace expansion)**.** *Let $M = (m_{ij})_{1\le i,j\le n}$ be a square matrix. Denote by M_{ij} a matrix obtained from M after deleting its i-th row and j-th column. Then for every i and j the determinant of M can be expanded into a sum:*

$$\det M = \begin{cases} \sum_{i=1}^{n} (-1)^{i+j} m_{ij} \det M_{ij} & (\textit{expansion with respect to } j\textit{-th row}), \\ \sum_{j=1}^{n} (-1)^{i+j} m_{ij} \det M_{ij} & (\textit{expansion with respect to } i\textit{-th column}). \end{cases}$$

Again the proof is to be found in almost every linear algebra textbook (e.g. [72, Theorems 3.23 and 3.24]). Although it is very easy to transform Laplace formula into a recursive algorithm for computing determinants, it is also easy to show that the complexity of such algorithm is again exponential. Consequently, the importance of this theorem is more theoretical than practical. In particular the following two results are immediate:

Corollary 3.4.3. *If M is a triangular matrix (either upper or lower-triangular), then $\det M$ is the product of elements on the main diagonal of M.*

Corollary 3.4.4. *The determinants of elementary matrices equal:*

$$\det E_1^m(i,j) = -1, \qquad \det E_2^m(i;\alpha) = \alpha; \qquad \det E_3^m(i,j;\alpha) = 1.$$

Recall that an elementary row transformation is equivalent to a multiplication by one of the elementary matrices. Thus, combining the above corollary with Cauchy theorem, we prove the following know properties of the determinant.

Observation 3.4.5. *Let M be a square matrix and α a nonzero scalar.*

- *If a matrix N is obtained from M by swapping its two rows, then $\det N = -\det M$.*
- *If a matrix N is obtained from M by replacing its row M_i by its multiple αM_i, then $\det N = \alpha \cdot \det M$.*
- *If a matrix N is obtained from M by replacing a row M_i by $M_i + \alpha M_j$ with $i \neq j$, then $\det N = \det M$.*

Computing determinant by elimination

Observation 3.4.5 suggest a fast method for computing the determinant: convert a given matrix into an upper-triangular (echelon) form, keeping track of the row transformations used. Then apply Corollary 3.4.3 together with Observation 3.4.5 to get the determinant. Formally speaking, if M is a square matrix and $E_1, \ldots, E_k$ are elementary matrices such that $\tilde{M} := E_k \cdots E_1 \cdot M$ is an upper-triangular matrix, then

$$\det M = (\det E_1)^{-1} \cdots (\det E_k)^{-1} \cdot (\det \tilde{M})$$

by Cauchy theorem. The matrices $\tilde{M}, E_1, \ldots, E_k$ are computable using any of the elimination algorithms we discussed in the previous section. Of course, if we are interested in just the determinant of M, we do not need to store all these matrices. Instead we may incorporate Observation 3.4.5 directly into Gauss elimination procedure. The following algorithm is a straightforward adaptation of Algorithm 3.2.5. Nevertheless, considering its importance, it deserves being stated explicitly.

Algorithm 3.4.6. *Given a square matrix M of size $n \times n$, this algorithm computes its determinant.*

1. *Initialize the sign $s := 1$;*
2. *for rows M_k with $k \in \{1, \ldots, n\}$ proceed as follows:*
   a) *find the pivot $m_{ik} \neq 0$ with $i \geq k$ (see Remark 31 on page 123);*
   b) *if no such m_{ik} exists, then return 0 and quit;*
   c) *if $i \neq k$, then swap the rows M_k and M_i and update the sign accordingly, setting $s := -s$.*
   d) *replace every row M_i for $i \in \{k+1, \ldots, n\}$ by a linear combination*

$$M_i - \frac{m_{ik}}{m_{kk}} \cdot M_k;$$

3. *set $d := m_{11} \cdots m_{nn}$ to be the product of elements on the main diagonal;*
4. *return $\det M = s \cdot d$.*

In a similar way as we adapted Algorithm 3.2.5, we can also adapt Jordan–Bareiss algorithm and take advantage of its superior numerical stability (when working with floating-point data) and the fact that it avoids fractions (when the matrix entries are integers/polynomials). As in the derivation of Jordan–Bareiss algorithm, below we use superscript to indicate iterations of the main loop.

To make things easier to follow, we will pretend for a moment, that we do not need to permute rows. In Jordan–Bareiss algorithm, we replace a row $M_i^{(k-1)}$ by an expression

$$\frac{m_{k,k}^{(k-1)}}{m_{k-1,k-1}^{(k-2)}} \cdot \left(M_i^{(k-1)} - \frac{m_{i,k}^{(k-1)}}{m_{k,k}^{(k-1)}} \cdot M_k^{(k-1)} \right).$$

We do it for every row beneath the current row $M_k^{(k)}$. Therefore, after clearing the first column of M, we have

$$\det M^{(1)} = \frac{\overbrace{m_{11}^{(0)} \cdots m_{11}^{(0)}}^{n-1 \text{ times}}}{\underbrace{m_{00}^{(-1)} \cdots m_{00}^{(-1)}}_{n-1 \text{ times}}} \cdot \det M^{(0)} = \left(m_{11}^{(0)}\right)^{n-1} \cdot \det M,$$

by Observation 3.4.5. (Recall that $m_{00}^{(-1)} = 1$, by definition.) Next, after erasing elements in the second column we obtain

$$\det M^{(2)} = \frac{\left(m_{22}^{(1)}\right)^{n-2}}{\left(m_{11}^{(0)}\right)^{n-2}} \cdot \det M^{(1)}$$
$$= \frac{\left(m_{22}^{(1)}\right)^{n-2}}{\left(m_{11}^{(0)}\right)^{n-2}} \cdot \left(m_{11}^{(0)}\right)^{n-1} \cdot \det M = \left(m_{22}^{(1)}\right)^{n-2} \cdot m_{11}^{(0)} \cdot \det M.$$

Continuing this process we eventually get the formula

$$\det M^{(n-1)} = m_{n-1,n-1}^{(n-2)} \cdots m_{11}^{(0)} \cdot \det M.$$

Now, $M^{(n-1)}$ is an upper-triangular matrix and so its determinant is the product of elements on its main diagonal:

$$\det M^{(n-1)} = m_{11}^{(n-1)} \cdots m_{nn}^{(n-1)} = m_{11}^{(0)} \cdots m_{nn}^{(n-1)}.$$

It follows that $\det M = m_{nn}^{(n-1)}$, hence the determinant of M is nothing else but the bottom-right element of the echelon matrix obtained from M using Jordan–Bareiss algorithm.

In the above discussion we assumed that the rows were not permuted. Since, every time we swap two rows, the sign of the determinant changes, we need to keep track of the number of rows swapped and update the sign accordingly. All in all, Jordan–Bareiss algorithm for computing the determinant takes a form:

Algorithm 3.4.7 (Jordan–Bareiss)**.** *Given a square matrix M of size $n \times n$ this algorithm computes its determinant.*

1. *Initialize the sign $s := 1$ and set $c := m_{00}^{(-1)} = 1$;*
2. *loop over rows M_k of M for $k \in \{1, \ldots, n\}$:*
   a) *find the pivot $m_{ik} \neq 0$ with $i \geq k$;*
   b) *if no such m_{ik} exists, then return 0 and quit;*
   c) *if $i \neq k$, then swap the rows M_k and M_i and update the sign, setting $s := -s$.*
   d) *replace every row M_i with $i \in \{k+1, \ldots, n\}$ by an expression*

$$\frac{\left(m_{kk} \cdot M_i - m_{ik} \cdot M_k\right)}{c};$$

   e) *update the divisor c setting $c := m_{kk}$;*
3. *return $\det M = s \cdot m_{nn}$.*

Example 33. We demonstrate how Jordan–Bareiss algorithm works on the following example. Take a matrix

$$M = \begin{pmatrix} 5 & 1 & -9 & 0 \\ -1 & 3 & -1 & 1 \\ 8 & -2 & 1 & -3 \\ 2 & 2 & 1 & 2 \end{pmatrix}.$$

The successive iterations of the main loop produce the following matrices:

$$M^{(1)} = \begin{pmatrix} 5 & 1 & -9 & 0 \\ \frac{\det\left(\begin{smallmatrix} 5 & 5 \\ -1 & -1 \end{smallmatrix}\right)}{1} & \frac{\det\left(\begin{smallmatrix} 5 & 1 \\ -1 & 3 \end{smallmatrix}\right)}{1} & \frac{\det\left(\begin{smallmatrix} 5 & -9 \\ -1 & -1 \end{smallmatrix}\right)}{1} & \frac{\det\left(\begin{smallmatrix} 5 & 0 \\ -1 & 1 \end{smallmatrix}\right)}{1} \\ \frac{\det\left(\begin{smallmatrix} 5 & 5 \\ 8 & 8 \end{smallmatrix}\right)}{1} & \frac{\det\left(\begin{smallmatrix} 5 & 1 \\ 8 & -2 \end{smallmatrix}\right)}{1} & \frac{\det\left(\begin{smallmatrix} 5 & -9 \\ 8 & 1 \end{smallmatrix}\right)}{1} & \frac{\det\left(\begin{smallmatrix} 5 & 0 \\ 8 & -3 \end{smallmatrix}\right)}{1} \\ \frac{\det\left(\begin{smallmatrix} 5 & 5 \\ 2 & 2 \end{smallmatrix}\right)}{1} & \frac{\det\left(\begin{smallmatrix} 5 & 1 \\ 2 & 2 \end{smallmatrix}\right)}{1} & \frac{\det\left(\begin{smallmatrix} 5 & -9 \\ 2 & 1 \end{smallmatrix}\right)}{1} & \frac{\det\left(\begin{smallmatrix} 5 & 0 \\ 2 & 2 \end{smallmatrix}\right)}{1} \end{pmatrix} = \begin{pmatrix} 5 & 1 & -9 & 0 \\ 0 & 16 & -14 & 5 \\ 0 & -18 & 77 & -15 \\ 0 & 8 & 23 & 10 \end{pmatrix},$$

$$M^{(2)} = \begin{pmatrix} 5 & 1 & -9 & 0 \\ 0 & 16 & -14 & 5 \\ 0 & \frac{\det\left(\begin{smallmatrix} 16 & 16 \\ -18 & -18 \end{smallmatrix}\right)}{5} & \frac{\det\left(\begin{smallmatrix} 16 & -14 \\ -18 & 77 \end{smallmatrix}\right)}{5} & \frac{\det\left(\begin{smallmatrix} 16 & 5 \\ -18 & -15 \end{smallmatrix}\right)}{5} \\ 0 & \frac{\det\left(\begin{smallmatrix} 16 & 16 \\ 8 & 8 \end{smallmatrix}\right)}{5} & \frac{\det\left(\begin{smallmatrix} 16 & -14 \\ 8 & 23 \end{smallmatrix}\right)}{5} & \frac{\det\left(\begin{smallmatrix} 16 & 5 \\ 8 & 10 \end{smallmatrix}\right)}{5} \end{pmatrix} = \begin{pmatrix} 5 & 1 & -9 & 0 \\ 0 & 16 & -14 & 5 \\ 0 & 0 & 196 & -30 \\ 0 & 0 & 96 & 24 \end{pmatrix},$$

$$M^{(3)} = \begin{pmatrix} 5 & 1 & -9 & 0 \\ 0 & 16 & -14 & 5 \\ 0 & 0 & 196 & -30 \\ 0 & 0 & \frac{\det\left(\begin{smallmatrix} 196 & 196 \\ 96 & 96 \end{smallmatrix}\right)}{16} & \frac{\det\left(\begin{smallmatrix} 196 & -30 \\ 96 & 24 \end{smallmatrix}\right)}{16} \end{pmatrix} = \begin{pmatrix} 5 & 1 & -9 & 0 \\ 0 & 16 & -14 & 5 \\ 0 & 0 & 196 & -30 \\ 0 & 0 & 0 & 474 \end{pmatrix}.$$

We did not swap any rows, hence we do not need to worry about the sign, consequently $\det M = 474$.

Dodgson condensation

Laplace expansion (Theorem 3.4.2) allows us to compute a determinant of order n by computing n determinants of order $n-1$, each. This is a recursive procedure, however, as we observed, its complexity is exponential, hence it does not have much practical use. But imagine that we could reduce the computation of the determinant of a $n \times n$ matrix to a **single** determinant of order $n-1$. This may lead to effective recursive procedures. Such algorithm are called *condensation* methods. In this section we describe two condensation schemes. The reader should be aware, though, that these methods are better suited for doing computations by hand (after all they were invented in mid-19th century) than for computer implementations. The first condensation method we discuss is due to F. Chiò[4].

[4]Felice Chiò (born 29 April 1813, died 28 May 28 1871) was an Italian mathematician and politician.

Theorem 3.4.8 (Chiò, 1853). *Let $M = (m_{ij})$ be a square matrix of size $n \times n$. Assume that $m_{11} \neq 0$. Then*

$$\det M = \frac{1}{m_{11}^{n-2}} \cdot \det \begin{pmatrix} \det\left(\begin{smallmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{smallmatrix}\right) & \det\left(\begin{smallmatrix} m_{11} & m_{13} \\ m_{21} & m_{23} \end{smallmatrix}\right) & \cdots & \det\left(\begin{smallmatrix} m_{11} & m_{1n} \\ m_{21} & m_{2n} \end{smallmatrix}\right) \\ \det\left(\begin{smallmatrix} m_{11} & m_{12} \\ m_{31} & m_{32} \end{smallmatrix}\right) & \det\left(\begin{smallmatrix} m_{11} & m_{13} \\ m_{31} & m_{33} \end{smallmatrix}\right) & \cdots & \det\left(\begin{smallmatrix} m_{11} & m_{1n} \\ m_{31} & m_{3n} \end{smallmatrix}\right) \\ \vdots & \vdots & & \vdots \\ \det\left(\begin{smallmatrix} m_{11} & m_{12} \\ m_{n1} & m_{n2} \end{smallmatrix}\right) & \det\left(\begin{smallmatrix} m_{11} & m_{13} \\ m_{n1} & m_{n3} \end{smallmatrix}\right) & \cdots & \det\left(\begin{smallmatrix} m_{11} & m_{1n} \\ m_{n1} & m_{nn} \end{smallmatrix}\right) \end{pmatrix}.$$

Proof. Multiply every row of M, except the first one, by m_{11}, and then replace rows $M_2, \ldots, M_n$ by linear combinations $M_2 - m_{21} \cdot M_1, \ldots, M_n - m_{n1} \cdot M_n$, respectively. This way we have

$$\begin{aligned} m_{11}^{n-1} \cdot \det M &= \det \begin{pmatrix} m_{11} & m_{12} & \cdots & m_{1n} \\ m_{11}m_{21} & m_{11}m_{22} & \cdots & m_{11}m_{2n} \\ \vdots & \vdots & & \vdots \\ m_{11}m_{n1} & m_{11}m_{n2} & \cdots & m_{11}m_{nn} \end{pmatrix} \\ &= \det \begin{pmatrix} m_{11} & m_{12} & \cdots & m_{1n} \\ 0 & \det\left(\begin{smallmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{smallmatrix}\right) & \cdots & \det\left(\begin{smallmatrix} m_{11} & m_{1n} \\ m_{21} & m_{2n} \end{smallmatrix}\right) \\ \vdots & \vdots & & \vdots \\ 0 & \det\left(\begin{smallmatrix} m_{11} & m_{12} \\ m_{n1} & m_{n2} \end{smallmatrix}\right) & \cdots & \det\left(\begin{smallmatrix} m_{11} & m_{1n} \\ m_{n1} & m_{nn} \end{smallmatrix}\right) \end{pmatrix} \end{aligned}$$

The last determinant can now be computed using Laplace expansion with respect to the first column:

$$= m_{11} \cdot \det \begin{pmatrix} \det\left(\begin{smallmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{smallmatrix}\right) & \cdots & \det\left(\begin{smallmatrix} m_{11} & m_{1n} \\ m_{21} & m_{2n} \end{smallmatrix}\right) \\ \vdots & & \vdots \\ \det\left(\begin{smallmatrix} m_{11} & m_{12} \\ m_{n1} & m_{n2} \end{smallmatrix}\right) & \cdots & \det\left(\begin{smallmatrix} m_{11} & m_{1n} \\ m_{n1} & m_{nn} \end{smallmatrix}\right) \end{pmatrix}.$$

This proves the assertion. □

Observe that the 2×2 matrices appearing in Chiò's theorem are the same as in Jordan–Bareiss algorithm. In fact, the row operations in the above proof are nothing else but one iteration of the main loop of Algorithm 3.4.7. Therefore Chiò's condensation method can be viewed as a variant of Jordan–Bareiss algorithm. We leave it to the reader to write down an explicit algorithm based on the above theorem. Instead we just show an example calculation.

Example 34. Let us compute the determinant of a 4×4 matrix (this is the same matrix

as in Example 33 using Chiò's method:

$$\det\begin{pmatrix} 5 & 1 & -9 & 0 \\ -1 & 3 & -1 & 1 \\ 8 & -2 & 1 & -3 \\ 2 & 2 & 1 & 2 \end{pmatrix} = \frac{1}{5^2}\cdot\det\begin{pmatrix} \det\left(\begin{smallmatrix} 5 & 1 \\ -1 & 3 \end{smallmatrix}\right) & \det\left(\begin{smallmatrix} 5 & -9 \\ -1 & -1 \end{smallmatrix}\right) & \det\left(\begin{smallmatrix} 5 & 0 \\ -1 & 1 \end{smallmatrix}\right) \\ \det\left(\begin{smallmatrix} 5 & 1 \\ 8 & -2 \end{smallmatrix}\right) & \det\left(\begin{smallmatrix} 5 & -9 \\ 8 & 1 \end{smallmatrix}\right) & \det\left(\begin{smallmatrix} 5 & 0 \\ 8 & -3 \end{smallmatrix}\right) \\ \det\left(\begin{smallmatrix} 5 & 1 \\ 2 & 2 \end{smallmatrix}\right) & \det\left(\begin{smallmatrix} 5 & -9 \\ 2 & 1 \end{smallmatrix}\right) & \det\left(\begin{smallmatrix} 5 & 0 \\ 2 & 2 \end{smallmatrix}\right) \end{pmatrix}$$

$$= \frac{1}{25}\cdot\det\begin{pmatrix} 16 & -14 & 5 \\ -18 & 77 & -15 \\ 8 & 23 & 10 \end{pmatrix} = \frac{1}{25}\cdot\frac{1}{16}\cdot\det\begin{pmatrix} \det\left(\begin{smallmatrix} 16 & -14 \\ -18 & 77 \end{smallmatrix}\right) & \det\left(\begin{smallmatrix} 16 & 5 \\ -18 & -15 \end{smallmatrix}\right) \\ \det\left(\begin{smallmatrix} 16 & -14 \\ 8 & 23 \end{smallmatrix}\right) & \det\left(\begin{smallmatrix} 16 & 5 \\ 8 & 10 \end{smallmatrix}\right) \end{pmatrix}$$

$$= \frac{1}{400}\cdot\det\begin{pmatrix} 980 & -150 \\ 480 & 120 \end{pmatrix} = \frac{189600}{400} = 474.$$

A major drawback of Chiò's method is that it lets the coefficients of the intermediate matrices to grow out of control. To some extend it can be already observed in the previous example. Notice that in Theorem 3.4.8 it is the resulting determinant—not the matrix entries like in Jordan–Bareiss agorithm—that is being divided by the pivot. Since we keep the pivots "outside" the matrix and only divide the determinant, the coefficients may grow considerable after a couple of condensation steps. This problem is somehow mitigated in the next condensation scheme, which is due to Ch. Dodgson[5].

Fix a square matrix $M = (m_{ij})$ of size $n \times n$. By the *interior* of M, we understand a a matrix $\operatorname{int} M$ obtained from M after dropping its first and last columns and rows. Hence, $\operatorname{int} M$ consists of all m_{ij} with $1 < i, j < n$. Define recursively a (finite) sequence of matrices of decreasing sizes. Let $M^{(-1)}$ be a matrix of size $(n+1)\times(n+1)$ consisting entirely of ones and $M^{(0)} := M$. Given $M^{(k)}$ construct $M^{(k+1)}$ by the formula

$$M^{(k+1)} := \begin{pmatrix} \dfrac{\det\left(\begin{smallmatrix} m_{1,1}^{(k)} & m_{1,2}^{(k)} \\ m_{2,1}^{(k)} & m_{2,2}^{(k)} \end{smallmatrix}\right)}{m_{2,2}^{(k-1)}} & \dfrac{\det\left(\begin{smallmatrix} m_{1,2}^{(k)} & m_{1,3}^{(k)} \\ m_{2,2}^{(k)} & m_{2,3}^{(k)} \end{smallmatrix}\right)}{m_{2,3}^{(k-1)}} & \cdots & \dfrac{\det\left(\begin{smallmatrix} m_{1,n-1}^{(k)} & m_{1,n}^{(k)} \\ m_{2,n-1}^{(k)} & m_{2,n}^{(k)} \end{smallmatrix}\right)}{m_{2,n}^{(k-1)}} \\ \dfrac{\det\left(\begin{smallmatrix} m_{2,1}^{(k)} & m_{2,2}^{(k)} \\ m_{2,1}^{(k)} & m_{3,2}^{(k)} \end{smallmatrix}\right)}{m_{3,2}^{(k-1)}} & \dfrac{\det\left(\begin{smallmatrix} m_{2,2}^{(k)} & m_{2,3}^{(k)} \\ m_{2,2}^{(k)} & m_{3,3}^{(k)} \end{smallmatrix}\right)}{m_{3,3}^{(k-1)}} & \cdots & \dfrac{\det\left(\begin{smallmatrix} m_{2,n-1}^{(k)} & m_{2,n}^{(k)} \\ m_{3,n-1}^{(k)} & m_{3,n}^{(k)} \end{smallmatrix}\right)}{m_{3,n}^{(k-1)}} \\ \vdots & \vdots & & \vdots \\ \dfrac{\det\left(\begin{smallmatrix} m_{n-1,1}^{(k)} & m_{n-1,2}^{(k)} \\ m_{n,1}^{(k)} & m_{n,2}^{(k)} \end{smallmatrix}\right)}{m_{n,2}^{(k-1)}} & \dfrac{\det\left(\begin{smallmatrix} m_{n-1,2}^{(k)} & m_{n-1,3}^{(k)} \\ m_{n,2}^{(k)} & m_{n,3}^{(k)} \end{smallmatrix}\right)}{m_{n,3}^{(k-1)}} & \cdots & \dfrac{\det\left(\begin{smallmatrix} m_{n-1,n-1}^{(k)} & m_{n-1,n}^{(k)} \\ m_{n,n-1}^{(k)} & m_{n,n}^{(k)} \end{smallmatrix}\right)}{m_{n,n}^{(k-1)}} \end{pmatrix}$$

Observe that the denominators are exactly the elements of the interior of $M^{(k-1)}$. Therefore all the above fractions are defined if and only if this interior does not contain zero.

[5] Charles Lutwidge Dodgson (born 27 January 1832, died 14 January 1898) was an English mathematician, writer and photographer. He is better known as the author of children's books ("Alice's adventures in wonderland" and "Through the looking glass") that he wrote under a nickname Lewis Carroll

Theorem 3.4.9 (Dodgson, 1866)**.** *Assume that the interior of $M^{(k-1)}$ does not contain zero, then all the above divisions are exact and* $\det M^{(k)} = \det M^{(k+1)}$.

We omit the proof that can be found in [57]. The problem of the Dodgson method is that it requires all the internal elements of M to be nonzero. This hurdle can often be circumvented using elementary row operations. If some element m_{ij} with $1 < i, j < n$ is zero, we search for a row M_k containing a nonzero element m_{kj} in the same column. If there are no such rows, then M contains a column consisting of zeros, hence $\det M = 0$ and we are done. Once we located such a row M_k, we replace M_i by $M_i + \alpha M_k$ for some random nonzero scalar α. This operation does not change the determinant, yet at the same time it forces m_{ij} to become nonzero. The point of selecting α at random is to minimize chances that we will accidentally "spoil" other elements. However, we will not go into a detailed analysis here, as it would drive us too far afield.

Example 35. Let us demonstrate Dodgson method using the same matrix as before.

$$\det\begin{pmatrix} 5 & 1 & -9 & 0 \\ -1 & 3 & -1 & 1 \\ 8 & -2 & 1 & -3 \\ 2 & 2 & 1 & 2 \end{pmatrix}$$

$$= \det\begin{pmatrix} \frac{\det\begin{pmatrix} 5 & 1 \\ -1 & 3 \end{pmatrix}}{1} & \frac{\det\begin{pmatrix} 1 & -9 \\ 3 & -1 \end{pmatrix}}{1} & \frac{\det\begin{pmatrix} -9 & 0 \\ -1 & 1 \end{pmatrix}}{1} \\ \frac{\det\begin{pmatrix} -1 & 3 \\ 8 & -2 \end{pmatrix}}{1} & \frac{\det\begin{pmatrix} 3 & -1 \\ -2 & 1 \end{pmatrix}}{1} & \frac{\det\begin{pmatrix} -1 & 1 \\ 1 & -3 \end{pmatrix}}{1} \\ \frac{\det\begin{pmatrix} 8 & -2 \\ 2 & 2 \end{pmatrix}}{1} & \frac{\det\begin{pmatrix} -2 & 1 \\ 2 & 1 \end{pmatrix}}{1} & \frac{\det\begin{pmatrix} 1 & -3 \\ 1 & 2 \end{pmatrix}}{1} \end{pmatrix} = \det\begin{pmatrix} 16 & 26 & -9 \\ -22 & 1 & 2 \\ 20 & -4 & 5 \end{pmatrix}$$

$$= \det\begin{pmatrix} \frac{\det\begin{pmatrix} 16 & 26 \\ -22 & 1 \end{pmatrix}}{3} & \frac{\det\begin{pmatrix} 26 & -9 \\ 1 & 2 \end{pmatrix}}{-1} \\ \frac{\det\begin{pmatrix} -22 & 1 \\ 20 & -4 \end{pmatrix}}{-2} & \frac{\det\begin{pmatrix} 1 & 2 \\ -4 & 5 \end{pmatrix}}{1} \end{pmatrix} = \det\begin{pmatrix} 196 & -61 \\ -34 & 13 \end{pmatrix}$$

$$= \det\begin{pmatrix} \frac{\det\begin{pmatrix} 196 & -61 \\ -34 & 13 \end{pmatrix}}{1} \end{pmatrix} = \det\begin{pmatrix} 474 \end{pmatrix} = 474.$$

Samuelson-Berkowitz algorithm

The Gaussian elimination discussed above is probably the most popular method for computing determinants. Nevertheless, it has two major drawbacks. First of all, it involves division of matrix elements. This can introduce round-off errors, when floating-point arithmetic is in use. On the other hand, for matrices with exact entries, like integer and polynomials (especially multivariate polynomials), Gaussian elimination may lead to an explosive growth of size of coefficients in intermediate steps. What is more, division is not always possible, yet the determinant still exists. This is especially true for rings with zero-divisors. The second problem with Gaussian elimination is that it is purely sequential and does not yield itself to parallel computation. In this subsection we present a radically different approach to calculating a determinant that addresses both above issues. Nevertheless, we will elaborate only on the first one, since concurrent programming falls way beyond the scope of this book.

The algorithm described below is called Samuelson-Berkowitz algorithm despite of some controversies concerning to whom it should really be attributed (for details refer to [73, §2.8]). This algorithm computes not only the determinant of a given matrix but also the characteristic polynomials (and so also determinants) of all its principal minors.

Definition 3.4.10. Let M be a square matrix of size $n \times n$ with coefficients in some ring R. The polynomial $\chi_M := \det(xI_n - M) \in R[x]$ is called the *characteristic polynomial* of M. Here, as usually, I_n denotes the unit matrix of size n.

Notice that χ_M is a monic polynomial, by definition. Evaluating χ_M at $x = 0$ we immediately have:

Observation 3.4.11. *The constant term of the characteristic polynomial of a matrix M of size $n \times n$ equals $(-1)^n \cdot \det M$.*

The fundamental property of characteristic polynomial is that it annihilates its matrix.

Theorem 3.4.12 (Cayley–Hamilton)**.** *If M is a square matrix and χ_M its characteristic polynomial, then*

$$\chi_M(M) = 0.$$

For the proof refer to any decent textbook on linear algebra (e.g. [72, Theorem 4.10]). Another classical notion concerning matrices is the notion of an adjoint.

Definition 3.4.13. The *adjoint matrix* (also known as *adjugate* or *adjunct*) of a square matrix $M = (m_{ij})_{1 \le i,j \le n}$ is a matrix

$$\operatorname{adj}(M) := \Big((-1)^{i+j} \cdot \det M_{ij}\Big)^T_{1 \le i,j \le n},$$

where M_{ij} denotes a matrix obtained from M by deleting the i-th row and j-th column.

Example 36. Consider a 3×3 matrix M with the following entries

$$M = \begin{pmatrix} 1 & 0 & 3 \\ 2 & -1 & 0 \\ 0 & 3 & 1 \end{pmatrix}.$$

By definition, the adjoint of M has a form

$$\begin{aligned}\operatorname{adj}(M) &= \begin{pmatrix} (-1)^{1+1}\cdot\det\begin{pmatrix} -1 & 0 \\ 3 & 1 \end{pmatrix} & (-1)^{1+2}\cdot\det\begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} & (-1)^{1+3}\cdot\det\begin{pmatrix} 2 & -1 \\ 0 & 3 \end{pmatrix} \\ (-1)^{2+1}\cdot\det\begin{pmatrix} 0 & 3 \\ 3 & 1 \end{pmatrix} & (-1)^{2+2}\cdot\det\begin{pmatrix} 1 & 3 \\ 0 & 1 \end{pmatrix} & (-1)^{2+3}\cdot\det\begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix} \\ (-1)^{3+1}\cdot\det\begin{pmatrix} 0 & 3 \\ -1 & 0 \end{pmatrix} & (-1)^{3+2}\cdot\det\begin{pmatrix} 1 & 3 \\ 2 & 0 \end{pmatrix} & (-1)^{3+3}\cdot\det\begin{pmatrix} 1 & 0 \\ 2 & -1 \end{pmatrix} \end{pmatrix}^T \\ &= \begin{pmatrix} -1 & 9 & 3 \\ -2 & 1 & 6 \\ 6 & -3 & -1 \end{pmatrix}.\end{aligned}$$

A well known property of an adjoint matrix is that it is a "nearly inverse" of M. More precisely:

Proposition 3.4.14. *Let M be a square $n \times n$ matrix, then*

$$M \cdot \operatorname{adj}(M) = \operatorname{adj}(M) \cdot M = \det(M) \cdot I_n.$$

The proof can again be found in linear algebra textbooks (see e.g. [72, Theorem 3.25]). The next lemma relates the notions of an adjoint and a characteristic polynomial.

Lemma 3.4.15. *Let M be a $n \times n$ matrix and $\chi_M = c_0 + c_1 x + \cdots + c_n x^n$ ($c_n = 1$) its characteristic polynomial. Then*

$$\operatorname{adj}(xI_n - M) = \sum_{j=1}^{n}\left(\sum_{i=1}^{j} c_{n-j+i}\cdot M^{i-1}\right)\cdot x^{n-j}. \tag{3.6}$$

Proof. Denote the right-hand-side of Eq. (3.6) by A and compute $A\cdot(xI_n - M)$. We have

$$\begin{aligned} A\cdot(xI_n - M) &= A\cdot x\cdot I_n - A\cdot M \\ &= \sum_{j=1}^{n}\left(\sum_{i=1}^{j} c_{n-j+i}\cdot M^{i-1}\right)\cdot x^{n-j+1} - \sum_{j=1}^{n}\left(\sum_{i=1}^{j} c_{n-j+i}\cdot M^{i}\right)\cdot x^{n-j}\end{aligned}$$

Change the indexation of the first sum

$$\begin{aligned} &= \sum_{j=0}^{n-1}\left(\sum_{i=1}^{j+1} c_{n-j+i-1}\cdot M^{i-1}\right)\cdot x^{n-j} - \sum_{j=1}^{n}\left(\sum_{i=1}^{j} c_{n-j+i}\cdot M^{i}\right)\cdot x^{n-j} \\ &= c_n M^0 x^n + \sum_{j=1}^{n-1}\left(\sum_{i=1}^{j+1} c_{n-j+i-1}\cdot M^{i-1} - \sum_{i=1}^{j} c_{n-j+i}\cdot M^{i}\right)\cdot x^{n-j} - \sum_{i=1}^{n} c_i M^i\end{aligned}$$

The characteristic polynomial is monic, hence $c_n = 1$. Change the indexation of the second sum in the parenthesis and write

$$\begin{aligned}
&= x^n \cdot I + \sum_{j=1}^{n-1}\left(\sum_{i=1}^{j+1} c_{n-j+i-1}\cdot M^{i-1} - \sum_{i=2}^{j+1} c_{n-j+i-1}\cdot M^{i-1}\right)\cdot x^{n-j} - \left(\sum_{i=0}^{n} c_i M^i - c_0 I\right)\\
&= x^n \cdot I + \sum_{j=1}^{n-1} c_{n-j}\cdot M^0 \cdot x^{n-j} - \chi_M(M) + c_0 \cdot I
\end{aligned}$$

Cayley-Hamilton theorem asserts that $\chi_M(M) = 0$, thus changing the indexation one last time we have

$$\begin{aligned}
&= x^n \cdot I + \sum_{j=1}^{n-1} c_j \cdot x^j \cdot I + c_0 \cdot I\\
&= \chi_M \cdot I\\
&= \det(xI_n - M)\cdot I_n\\
&= \operatorname{adj}(xI_n - M)\cdot(xI_n - M),
\end{aligned}$$

where the last equality follows from Proposition 3.4.14. All in all, we have proved that

$$A\cdot(xI_n - M) = \operatorname{adj}(xI_n - M)\cdot(xI_n - M).$$

The common term $xI_n - M$ is clearly invertible so that we may cancel it. This yields the thesis. □

Again let $M = (m_{ij})_{1\le i,j\le n}$ be a fixed $n\times n$ matrix. For any $k\in\{1,\dots,n\}$ by M_k denote the principal minor of M of size k. In other words, M_k is the submatrix of M consisting of elements located in the first k rows and columns. In particular $M_1 = (m_{11})$ and $M_n = M$. Subdivide the minor M_{k+1} into four cells:

$$M_{k+1} = \left(\begin{array}{c:c} M_k & C_k \\ \hdashline R_k & m_{k+1,k+1}\end{array}\right),$$

where $R_k = (m_{k+1,1},\dots,m_{k+1,k})$ is a row vector and $C_k = (m_{1,k+1},\dots,m_{k,k+1})^T$ is a column vector. Let $\chi_k := \chi_{M_k}$ be the characteristic polynomial of M_k. The idea behind Samuelson-Berkowitz algorithm is to express χ_{k+1} in terms of χ_k. This way we can recursively build all the successive characteristic polynomials, starting from $\chi_1 = \det(xI_1 - M_1) = x - m_{11}$. Then Observation 3.4.11 yields $\det M = (-1)^n\chi_n(0)$. The recursive formula is provided by the following lemma.

Lemma 3.4.16 (Samuelson's formula)**.** *Assume that $\chi_k = c_0 + c_1 x + \cdots + c_k x^k$, with $c_k = 1$, then*

$$\chi_{k+1} = (x - m_{k+1,k+1})\cdot\chi_k - \sum_{j=1}^{k}\left(\sum_{i=1}^{j} c_{k-j+i}\cdot R_k\cdot M_k^{i-1}\cdot C_k\right)\cdot x^{k-j}.$$

Proof. By definition $\chi_{k+1} = \det(xI_{k+1} - M_{k+1})$, using the proposed subdivision of M_{k+1} we may write

$$\chi_{k+1} = \det\left(\left(\begin{array}{c:c} xI_k & 0 \\ \hdashline 0 & x \end{array}\right) - \left(\begin{array}{c:c} M_k & C_k \\ \hdashline R_k & m_{k+1,k+1} \end{array}\right)\right)$$

Expand the determinant along the last row using Laplace's formula

$$= \sum_{j=1}^{k} (-1)^{j+k+1} \cdot (-m_{k+1,j}) \cdot \det B_j + (x - m_{k+1,k+1}) \cdot \det(xI_k - M_k)$$

Here, B_j is a $k \times k$ matrix obtained from $(xI_k - M_k | C_k)$ by dropping the j-th column. Expand each determinant along the last column, again using Laplace's formula.

$$= \sum_{j=1}^{k} \left(-1)^{j+k+1} \cdot (-m_{k+1,j}) \cdot \sum_{i=1}^{k} (-1)^{i+k} \cdot \det A_{ij} \right) + (x - m_{k+1,k+1}) \cdot \chi_k$$

Now, the matrix A_{ij} is nothing else but $(xI_k - M_k)$ with the i-th row and j-th column deleted. Rearranging the terms we have

$$= (-1) \cdot \sum_{j=1}^{k} \left(m_{k+1,j} \cdot \sum_{i=1}^{k} (-1)^{i+j} \cdot \det A_{ij} \cdot m_{i,k+1} \right) + (x - m_{k+1,k+1}) \cdot \chi_k$$

Observe that $(-1)^{i+j} A_{ij}$ the the entry of $\operatorname{adj}(xI_k - M_k)$ located in the j-th row and i-th column. The double sum can now be interpreted as a matrix multiplication so that

$$= (x - m_{k+1,k+1}) \cdot \chi_k - R_k \cdot \operatorname{adj}(xI_k - M_k) \cdot C_k$$

The previous lemma provides a formula for $\operatorname{adj}(xI_k - M_k)$, consequently

$$= (x - m_{k+1,k+1}) \cdot \chi_k - R_k \cdot \sum_{j=1}^{k} \left(\sum_{i=1}^{j} c_{k-j+1} M_k^{i-1} \right) \cdot x^{k-j} \cdot C_k$$

This proves the thesis. □

Example 37. We now show how to compute the determinant of a matrix using Samuelson's formula. Consider a 4×4 matrix

$$M = \begin{pmatrix} 5 & 1 & -9 & 0 \\ -1 & 3 & -1 & 1 \\ 8 & -2 & 1 & -3 \\ 2 & 2 & 1 & 2 \end{pmatrix}.$$

The first principal minor M_1 consists of a single entry at the top-left corner of M, that is $M_1 = \begin{pmatrix} 5 \end{pmatrix}$. The characteristic polynomial of M_1 is trivial to compute, $\chi_1 = x - 5$. Subdivide the second principal minor:

$$M_2 = \left(\begin{array}{c|c} 5 & 1 \\ \hline -1 & 3 \end{array}\right).$$

Therefore, $R_1 = (-1)$ and $C_1 = (1)$. Compute χ_2 using Lemma 3.4.16:

$$\begin{aligned} \chi_2 &= (x - m_{22}) \cdot \chi_1 - \chi_{1,1} \cdot R_1 M_1^0 C_1 \cdot x^0 \\ &= (x-3)(x-5) - (-1) \\ &= x^2 - 8x + 16. \end{aligned}$$

Here, $\chi_{i,j}$ denotes the coefficient at x^j in χ_i. Analogously, express M_3 as

$$M_3 = \left(\begin{array}{cc|c} 5 & 1 & -9 \\ -1 & 3 & -1 \\ \hline 8 & -2 & 1 \end{array}\right).$$

It follows that

$$\begin{aligned} \chi_3 &= (x - m_{33}) \cdot \chi_2 - \Big(\chi_{2,2} \cdot R_2 M_2^0 C_2 x + \big(\chi_{2,1} \cdot R_2 M_2^0 C_2 + \chi_{2,2} \cdot R_2 M_2^1 C_2 \big) \cdot x^0 \Big) \\ &= x^3 - 9x^2 + 94x - 196. \end{aligned}$$

Finally, compute χ_4 as

$$\begin{aligned} \chi_4 &= (x - m_{44}) \cdot \chi_3 - \sum_{j=1}^{3} \Big(\sum_{i=1}^{j} c_{3-j+i} \cdot R_3 \cdot M_3^{i-1} \cdot C_3 \Big) \cdot x^{3-j} \\ &= x^4 - 11x^3 + 113x^2 - 456x + 474. \end{aligned}$$

The determinant of M is the same as the constant term of χ_4 by Observation 3.4.11, hence $\det M = 474$.

Before we formalize the procedure described in the above example, first we introduce a convenient matrix notation that simplifies both: the description of the algorithm, as well as the actual computations. For any $k \in \{1, \dots, n\}$ construct a $(k+2) \times (k+1)$ matrix

$$T_k := \begin{pmatrix} -m_{k+1,k+1} & -R_k M^0 C_k & \cdots & -R_k M_k^{k-3} C_k & -R_k M_k^{k-2} C_k & -R_k M_k^{k-1} C_k \\ 1 & -m_{k+1,k+1} & \cdots & -R_k M_k^{k-4} C_k & -R_k M_k^{k-3} C_k & -R_k M_k^{k-2} C_k \\ \vdots & 1 & \ddots & \vdots & \vdots & \vdots \\ 0 & \vdots & \ddots & -m_{k+1,k+1} & -R_k C_k & -R_k M_k C_k \\ 0 & 0 & \cdots & 1 & -m_{k+1,k+1} & -R_k C_k \\ 0 & 0 & \cdots & 0 & 1 & -m_{k+1,k+1} \\ 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix}$$

More formally, the entry in the i-th row and j-th column equals:

- $-R_k \cdot M_k^{j-i-1} \cdot C_k$ if $i < j-1$.
- $-m_{k+1,k+1}$ if $i = j-1$,
- 1 if $i = j$,
- 0 if $i > j$,

This a a *Toeplitz matrix* (the values on each diagonal are the same) with zeros everywhere below the main diagonal. Formally, such matrices are called *upper-triangular*, yet here the nonzero entries form not a triangle but a trapezoid, since T_k is nor square. Nevertheless, there is no such a term as "upper-trapezoid matrix".

Let $(d_0, \ldots, d_{k+1} = 1)$ be the coefficients of the characteristic polynomial χ_{k+1} of M_{k+1} and $(c_0, \ldots, c_k = 1)$ the coefficients of χ_k. The Samuelson's formula can now be interpreted in terms of matrix multiplication as

$$\begin{pmatrix} d_0 \\ d_1 \\ \vdots \\ d_{k+1} \end{pmatrix} = T_k \cdot \begin{pmatrix} c_0 \\ \vdots \\ c_k \end{pmatrix} \tag{3.7}$$

or (slightly abusing the notation) in an abbreviated form $\chi_{k+1} = T_k \cdot \chi_k$.

Example 38. For the matrix M from the previous example, we have $\chi_3 = x^3 - 9x^2 + 94x - 196$ and

$$T_3 = \begin{pmatrix} -m_{4,4} & -R_3C_3 & -R_3M_3C_3 & -R_3M_3^2C_3 \\ 1 & -m_{4,4} & -R_3C_3 & -R_3M_3C_3 \\ 0 & 1 & -m_{4,4} & -R_3C_3 \\ 0 & 0 & 1 & -m_{44} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} -2 & 1 & -63 & -579 \\ 1 & -2 & 1 & -63 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -2 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Compute the coefficients of χ_4 using Eq. (3.7):

$$\begin{pmatrix} -2 & 1 & -63 & -579 \\ 1 & -2 & 1 & -63 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} -196 \\ 94 \\ -9 \\ 1 \end{pmatrix} = \begin{pmatrix} 474 \\ -456 \\ 113 \\ -11 \\ 1 \end{pmatrix}.$$

Finally $\chi_4 = x^4 - 11x^3 + 113x^2 - 456x + 474$.

We are now ready to present Samuelson–Berkowitz algorithm in an explicit form.

Algorithm 3.4.17 (Samuelson-Berkowitz)**.** *Given a square matrix $M = (m_{ij})_{1 \le i,j \le n}$ with coefficients in some ring, this algorithm computes the characteristic polynomial χ_M of M.*

1. *Initialize* $V := \binom{-m_{11}}{1}$;

2. *for every* $k \in \{1, \dots, n-1\}$ *proceed as follows:*
   a) *let* M_k *be the* k*-th principal minor of* M*;*
   b) *set* $R_k := (m_{k+1,1}, \dots, m_{k+1,k})$ *and* $C_k := (m_{1,k+1}, \dots, m_{k,k+1})^T$*;*
   c) *compute the products*

$$-R_k M_k^0 C_k, \dots, -R_k M_k^{k-1} C_k;$$

   d) *construct a Toeplitz matrix*

$$T_k := \begin{pmatrix} -m_{k+1,k+1} & -R_k C_k & \cdots & -R_k M_k^{k-1} C_k \\ 1 & -m_{k+1,k+1} & \ddots & -R_k M_k^{k-2} C_k \\ 0 & 1 & \ddots & -R_k M_k^{k-3} C_k \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

   e) *update* V *setting* $V := T_k \cdot V$*;*
3. *return* $\chi_M = (1, x, \dots, x^{n-1}, x^n) \cdot V$.

Bird's algorithm

We now present another, quite recent, method for computing a determinant, which is due to R. Bird. Like Samuelson–Berkowitz algorithm, this method also avoids division, hence is well suited for matrices with exact/symbolic entries.

Take a square matrix M of size $n \times n$. We construct a (finite) sequence of matrices A_k for $k \in \{0, \dots, n-1\}$ by a recurrent formula. Let $A_0 := M$ and for $k > 0$, if $A_k = \left(a_{i,j}^{(k)}\right)_{1 \le i,j \le n}$ set

$$A_{k+1} := \begin{pmatrix} -\sum_{j=2}^{n} a_{j,j}^{(k)} & a_{1,2}^{(k)} & \cdots & a_{1,n-1}^{(k)} & a_{1,n}^{(k)} \\ 0 & -\sum_{j=3}^{n} a_{j,j}^{(k)} & \cdots & a_{2,n-1}^{(k)} & a_{2,n}^{(k)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & -\sum_{j=n}^{n} a_{j,j}^{(k)} & a_{n-1,n}^{(k)} \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix} \cdot M.$$

In layman's terms, the matrix A_{k+1} is constructed as a product of a certain upper-triangular matrix and the original matrix M. This upper-triangular matrix is constructed from A_k as follows. Entries above the main diagonal are left intact. Below the main diagonal we fill the matrix with zeros. Finally, the entries on the main diagonal are computed as the sums of the entries on the main diagonal of A_k below and to the right of the current position.

Theorem 3.4.18 (Bird)**.** *The matrix* A_{n-1} *contains zeros everywhere but in its top-left corner, which equals* $(-1)^{n-1} \cdot \det M$.

The proof of the theorem needs some preparations. Let α, β be two sequences (tuples) of indices of the same length. By $D_{\alpha,\beta} = D_{\alpha,\beta}(M)$ we denote the determinant of a submatrix of M consisting of rows indexed by α and columns indexed by β. In particular, if $\alpha = (i)$ and $\beta = (j)$ are two singletons, then $D_{\alpha,\beta} = m_{ij}$.

Observation 3.4.19. *If α (or β) contains a duplicated entry, then $D_{\alpha,\beta} = 0$.*

Denote by $\alpha \oplus \beta$ the concatenation of two sequences α and β and let $S_k(\alpha)$ be the set of all subsequences of α of length k. If $A_k = (a_{ij}^{(k)})_{1\le i,j\le n}$ is one of the matrices constructed above, then the entries of A_k are computed by the following technical lemma.

Lemma 3.4.20. *For every $k \in \{1,\dots,n-1\}$ and all $i, j \in \{1,\dots,n\}$*

$$a_{ij}^{(k)} = (-1)^k \cdot \sum_{\alpha \in S_k\left((i+1,\dots,n)\right)} D_{(i)\oplus\alpha,(j)\oplus\alpha}.$$

Proof

Proof of Theorem 3.4.18. Use the above lemma to compute elements of A_{n-1}:

$$a_{ij}^{(n-1)} = (-1)^{n-1} \cdot \sum_{\alpha \in S_{n-1}\left((i+1,\dots,n)\right)} D_{(i)\oplus\alpha,(j)\oplus\alpha}.$$

Observe that $S_{n-1}\big((i+1,\dots,n)\big)$ is empty for all indices i, except for $i = 1$, when $S_{n-1}\big((2,\dots,n)\big) = \big\{(2,\dots,n)\big\}$. Therefore, the above formula simplifies to

$$a_{ij}^{(n-1)} = (-1)^{n-1} \cdot D_{(i)\oplus(2,\dots,n),(j)\oplus(2,\dots,n)},$$

which evaluates to $(-1)^{n-1} \cdot D_{(1,\dots,n),(1,\dots,n)} = (-1)^{n-1} \cdot \det M$ for $i = j = 1$ and to zero for all other choices of i and j. □

Example 39. We illustrate Bird's algorithm computing the determinant of

$$M = \begin{pmatrix} 5 & 1 & -9 & 0 \\ -1 & 3 & -1 & 1 \\ 8 & -2 & 1 & -3 \\ 2 & 2 & 1 & 2 \end{pmatrix}.$$

The consecutive elements of the sequence $(A_k)_{1\le k<n-1}$ are computed as follows. First $A_0 = M$, then

$$A_1 = \begin{pmatrix} -6 & 1 & -9 & 0 \\ 0 & -3 & -1 & 1 \\ 0 & 0 & -2 & -3 \\ 0 & 0 & 0 & 0 \end{pmatrix} \cdot M = \begin{pmatrix} -103 & 15 & 44 & 28 \\ -3 & -5 & 3 & 2 \\ -22 & -2 & -5 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$A_2 = \begin{pmatrix} 10 & 15 & 44 & 28 \\ 0 & 5 & 3 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \cdot M = \begin{pmatrix} 443 & 23 & -33 & -61 \\ 23 & 13 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$A_3 = \begin{pmatrix} -13 & 23 & -33 & -61 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \cdot M = \begin{pmatrix} -474 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Consequently $\det M = (-1)^3 \cdot -474 = 474$.

Time complexity analysis. Construction of the matrix A_{k+1} requires $n-1$ additions of coefficients followed by one matrix multiplication. To compute the determinant, one needs to construct $n-1$ matrices, namely $A_1, \ldots, A_{n-1}$. Thus, the time complexity of Bird's algorithm is

$$O(n \cdot \text{complexity of matrix multiplication}).$$

Exercise 7. Express Bird's method in form of an explicit algorithm (pseudo-code).

Modular computation of a determinant

An illustrative example application of modular arithmetic is a computation of the determinant of an integer matrix. In Section 3.4 we learned a number of different methods for computing the determinant. Gaussian elimination is fast, but requires division. When applied to integer matrices, not only does it force us to work in $\mathbb{Q}$, but it also suffers from the growth of sizes of coefficients. Division-free methods avoid these problems but are asymptotically slower. We may however benefit from the best of both worlds, if we decide to use modular arithmetic. The idea is to compute the determinant modulo a number of distinct (small) primes and then use Chinese remainder theorem. This way, we may use the (fast) Gaussian elimination method and, since we work over a finite field $\mathbb{F}_p = \{0, \ldots, p-1\}$, we do not have to worry about fractions and growing sizes of coefficients.

Fix a prime number p and take a square matrix $M = \left(m_{ij}\right)$ with integer entries. For every m_{ij} denote its residue modulo p by $\overline{m}_{ij} := (m_{ij} \bmod p)$. Compute the determinant of $\overline{M} := (M \bmod p) = \left(\overline{m}_{ij}\right)$. Using the fact that congruences preserve sums and

products (see Proposition 2.1.1), by definition of the determinant, we have:

$$\det \overline{M} = \sum_{\sigma} \operatorname{sgn}(\sigma) \cdot \prod_{i=1}^{n} \overline{m}_{i\sigma(i)} \equiv \sum_{\sigma} \operatorname{sgn}(\sigma) \cdot \prod_{i=1}^{n} m_{i\sigma(i)} = \det M \pmod{p}.$$

Thus, modular methods are indeed suitable for determinant computation.

We need the following estimate, due to J. Hadamard[6], of the size of a determinant. We will use it to decide how many primes we must consider.

Theorem 3.4.21 (Hadamard inequality, 1893). *Let $M = \left(m_{ij}\right)$ be a matrix of size $n \times n$ with real entries, then*

$$|\det M| \leq \prod_{j=1}^{n} \sqrt{\sum_{i=1}^{n} m_{ij}^2}.$$

In the proof of the theorem, by $M_{(j)}$ we denote the j-th column if M. With this convention, Hadamard inequality can be written as

$$|\det M| \leq \prod_{j=1}^{n} \|M_{(j)}\|,$$

where $\|M_{(j)}\| = \sqrt{M_{(j)}^T \cdot M_{(j)}}$ is the 2-norm of $M_{(j)}$.

Proof. The assertion is trivially true when $\det M = 0$, hence without loss of generality we may assume that M is non-singular. The QR-decomposition provides us with two matrices: an orthogonal matrix $Q = (q_{ij})$ and an upper-triangular matrix $R = (r_{ij})$ such that $M = Q \cdot R$. A single entry m_{ij} of M can be, therefore, expresses as

missing

$$m_{ij} = \sum_{k=1}^{n} q_{ik} r_{kj}.$$

The matrix R is upper-triangular, hence $r_{kj} = 0$ whenever $k > j$, so we have

$$m_{ij} = \sum_{k=1}^{j} q_{ik} r_{kj}.$$

Equivalently

$$M_{(j)} = \sum_{k=1}^{j} r_{kj} Q_{(k)}.$$

[6] Jacques Salomon Hadamard (born 8 December 1865, died 17 October 1963) was a French mathematician.

The columns of Q form an orthonormal basis of $\mathbb{R}^n$. In particular $Q_{(i)}^T \cdot Q_{(j)}$ equals either 1 (when $i = j$) or 0 (when $i \neq j$). Let us compute the squared norm of $M_{(j)}$. We have

$$\|M_{(j)}\|^2 = M_{(j)}^T \cdot M_{(j)} = \left(\sum_{k=1}^{j} r_{kj} Q_{(k)}^T\right)\left(\sum_{l=1}^{j} r_{lj} Q_{(l)}\right) = \sum_{k=1}^{j} r_{kj}^2 \geq r_{jj}^2.$$

It follows that $|r_{jj}| \leq \|M_{(j)}\|$ for every $j \leq n$.

Recall that R is upper-triangular, hence $\det R = r_{11} \cdots r_{nn}$. On the other hand Q is orthogonal and so $\det Q = 1$. Combining these two facts with Cauchy theorem (Theorem 3.4.1) we obtain the desired inequality:

$$|\det M| = |\det(QR)| = |\det Q| \cdot |\det R| = 1 \cdot \prod_{j=1}^{n} r_{jj} \leq \prod_{j=1}^{n} \|M_{(j)}\|. \qquad \square$$

W dowodzie tw. Mahlera potrzebna jest wersja nierownosci H. dla wspolczynnikow zespolonych, a nie tylko rzeczywstych

Corollary 3.4.22. *If there is a constant $B > 0$, such that $|m_{ij}| \leq B$ for every $i, j \leq n$, then*

$$|\det M| \leq B^n \cdot \sqrt{n^n}.$$

We may now present the modular algorithm for computing the determinant.

Algorithm 3.4.23. *Given a square $n \times n$ matrix $M = \left(m_{ij}\right)$ with integer entries, this algorithm computes the determinant $\det M$.*

1. *Find a bound B such that $B \geq |m_{ij}|$ for all $i, j \leq n$;*
2. *find odd primes $p_1, \ldots, p_k$, whose product is greater than twice the Hadamard's bound, i.e.*
$$m := p_1 \cdots p_k > 2 \cdot B^n \cdot \sqrt{n^n};$$
3. *for every $i \in \{1, \ldots, k\}$ compute the determinant over $\mathbb{F}_{p_i}$*
$$d_i := \det(M \bmod p_i)$$
using e.g. Gaussian elimination;
4. *with Chinese remainder theorem find a solution d to the system of congruences*
$$d \equiv d_i \pmod{p_i} \quad \text{for every } i \in \{1, \ldots, k\};$$
5. *if $d > m/2$, then replace it by $d - m$;*
6. *return the determinant $\det M = d$.*

Proof of correctness. We have already observed that a congruence $\det(M \bmod p_i) \equiv \det M \pmod{p_i}$ holds for every prime p_i. Lemma 2.2.8 asserts that

$$\det M \equiv d \pmod{m},$$

where $m = p_1 \cdots p_k$. Hadamard's inequality says that $\det M < m/2$. If also $d < m/2$, then the congruence $d - \det M \equiv 0 \pmod{m}$ implies that $d = \det M$. In particular $\det M$ is positive in this case. On the other hand, if $d > m/2$ (it cannot be equal $m/2$, since all primes p_i are odd), then $\det M$ must be negative and $\det M = d - m$. □

Example 40. Let us compute the determinant of a matrix

$$M := \begin{pmatrix} 3 & 1 & -2 \\ 2 & -3 & -1 \\ 1 & 3 & 0 \end{pmatrix}$$

using Algorithm 3.4.23. Take $B = 3$. Hadamard's inequality asserts that

$$|\det M| \leq 3^3 \cdot \sqrt{3^3} = 81\sqrt{3} \approx 140.296.$$

It suffices to consider three primes: 3, 7 and 17, since $m := 3 \cdot 7 \cdot 17 = 357$ is greater than twice the Hadamard's bound. We will successively compute the determinants modulo each prime and use Algorithm 2.2.10 to incrementally solve the resulting system of congruences.

First we compute $\det(M \bmod 3)$. We have

$$d_1 = \det(M \bmod 3) = \det \begin{pmatrix} 0 & 1 & 1 \\ 2 & 0 & 2 \\ 1 & 0 & 0 \end{pmatrix} = 2.$$

Therefore, we know that $\det M \equiv 2 \pmod 3$. Analogously we compute the determinant of M modulo 7:

$$d_2 := \det(M \bmod 7) = 4.$$

Solving the system of congruences (using Algorithm 2.2.6)

$$\begin{cases} d \equiv 2 \pmod 3 \\ d \equiv 4 \pmod 7 \end{cases}$$

to find that $d \equiv 11 \pmod{21}$. Finally we compute

$$d_3 := \det(M \bmod 17) = 7$$

and solve a system of congruences

$$\begin{cases} d \equiv 11 \pmod{21} \\ d \equiv 7 \pmod{17}. \end{cases}$$

Then $d = 347$ is a unique solution less than m. Now $d > m/2$ and it follows that

$$\det M = d - m = -10.$$

3.5 Systems of linear equations

Recap of Gauss elimination, iterative methods

3.6 Eigenvalues and eigenvectors

Let M be a square $n \times n$ matrix with coefficients in some field K. Recall from the course of linear algebra that a scalar $\lambda \in K$ is said to be an *eigenvalue* of M, if there is a nonzero vector $v \in K^n$ such that

$$M \cdot v = \lambda \cdot v.$$

Any vector v satisfying this condition is called an *eigenvector* of M associated with λ.

Example 41. Take a matrix $M := \begin{pmatrix} 1 & 1 \\ -1 & 3 \end{pmatrix}$ over the field of rationals. Then $M \cdot (1,1)^T = (2,2)^T = 2 \cdot (1,1)^T$. Hence $\lambda = 2$ is an eigenvalue of M and $v = (1,1)^T$ is an associated eigenvector. Of course any multiple of v is an eigenvector, too.

In geometric terms, multiplication by M, when restricted to the direction of an eigenvector v, behaves exactly like a scaling by the associated eigenvalue λ; it either stretches or shrinks vectors along the line determined by v. It follows immediately from the definition that eigenvectors associated with a specific eigenvalue λ form a kernel of an endomorphism $v \mapsto (\lambda I - M) \cdot v$. The dimension of this kernel is called the *geometric multiplicity* of the eigenvalue λ. In particular, λ is an eigenvalue of M if and only if this endomorphism is not injective (is not an automorphism). From the course of linear algebra we know that the later condition is equivalent to $\det(\lambda I - M) = 0$.

Recall (see Definition 3.4.10), that the *characteristic polynomial* of a square matrix M is given by the formula $\chi_M := \det(xI_n - M)$, where n is the size of M. Using the notion of the characteristic polynomial, we may summarize the above discussion as follows:

Observation 3.6.1. *An element $\lambda \in K$ is an eigenvalue of M if and only if it is a root of the characteristic polynomial χ_M.*

The multiplicity of λ as the root of χ_M is called the *algebraic multiplicity* of the eigenvalue λ. Since the degree of χ_M equals n, it follows that the algebraic multiplicities of the eigenvalues of M sum up to its size.

Let $M = \operatorname{diag}(a_1, \ldots, a_n)$ be a diagonal matrix, it is easy to notice that scalars $a_1, \ldots, a_n$ are eigenvalues of M and the associated eigenvectors are just those of the canonical basis of K^n. The following four results are well known. The reader is welcome to prove them himself or to check the proofs in any linear algebra textbook.

Proposition 3.6.2. *If M is a square matrix and χ_M is its characteristic polynomials, then $\chi_M(M) = 0$.*

Proposition 3.6.3. *Eigenvectors associated with distinct eigenvalues are linearly independent.*

Proposition 3.6.4. *If a matrix M of size $n \times n$ has n linearly independent eigenvectors $v_1, \ldots, v_n$, then M is similar to a diagonal matrix $D = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$, where $\lambda_1, \ldots, \lambda_n$ are associated eigenvalues. More precisely,*

$$M = V \cdot D \cdot V^{-1},$$

where $V = \big(v_1, \ldots, v_n\big)$ is a change-of-basis matrix whose columns are the eigenvectors associated to $\lambda_1, \ldots, \lambda_n$.

The converse implication is also true.

Proposition 3.6.5. *If a matrix M is similar to some diagonal matrix $D = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$, i.e. if there is an invertible matrix V such that*

$$M = V \cdot D \cdot V^{-1},$$

then $\lambda_1, \ldots, \lambda_n$ are eigenvalues of M and the columns of V are the associated eigenvectors.

Later we will discuss methods for finding actual eigenvalues of a given matrix. However, the sum and product of all the eigenvalues can be obtained very easily. This is, by the way, a special case of Viète's formulæ for the roots of a polynomial (see Theorem 6.1.6).

Definition 3.6.6. Let $M = (m_{ij})$ be a square matrix of size $n \times n$. The sum $m_{11} + m_{22} + \cdots + m_{nn}$ of the entries on the main diagonal is called the *trace* of M and denoted $\operatorname{Tr} M$.

Proposition 3.6.7. *Let M be a square matrix of size $n \times n$ and let $\lambda_1, \ldots, \lambda_n$ be all its eigenvalues, where each eigenvalue is repeated according to its algebraic multiplicity. Then the following two formulas hold:*

$$\det M = \lambda_1 \cdots \lambda_n, \qquad \operatorname{Tr} M = \lambda_1 + \cdots + \lambda_n.$$

Proof. We know that the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ of M are roots of its characteristic polynomial. Hence χ_M factors into linear terms as

$$\chi_M = (x - \lambda_1) \cdot (x - \lambda_2) \cdots (x - \lambda_n). \tag{3.8}$$

In order to prove the first assertion evaluate the characteristic polynomial at zero. On one hand we have

$$\chi_M(0) = (0 - \lambda_1) \cdots (0 - \lambda_n) = (-1)^n \cdot \lambda_1 \cdots \lambda_n.$$

On the other hand, by definition we have

$$\chi_M(0) = \det\big(0 \cdot I_n - M\big) = (-1)^n \cdot \det M.$$

Comparing the right-hand-sides of the above two formulas we obtain the first assertions.

To prove the second assertion we will compute the coefficient at x^{n-1} of χ_M. Write

$$\chi_M = \det\big(x \cdot I_n - M\big) = \det \begin{pmatrix} x - m_{11} & m_{12} & \cdots & m_{1n} \\ m_{21} & x - m_{22} & \cdots & m_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & \cdots & x - m_{nn} \end{pmatrix}.$$

Express the determinant as a sum running over all permutations of the set $\{1, \ldots, n\}$ (see Eq. (3.5)). The only term of the sum that involves more than $n-2$ copies of x is the one that corresponds to the identity. It is the term

$$(x - m_{11}) \cdot (x - m_{22}) \cdots (x - m_{nn}).$$

Thus, this is the only place where x^{n-1} occurs. Expanding the product, we see that the coefficient of x^{n-1} equals

$$(-1) \cdot (m_1 1 + m_{22} + \cdots + m_{nn}) = -\operatorname{Tr} M.$$

On the other hand, if we apply the same expansion to Eq. (3.8), we obtain the coefficient $(-1)\cdot(\lambda_1 + \cdots + \lambda_n)$. Two polynomials are equal if and only if all their coefficients are equal. Therefore we have

$$-\operatorname{Tr} M = (-1) \cdot (\lambda_1 + \cdots + \lambda_n).$$

This proves the second assertion. □

Example 42. Take a matrix over the finite field $\mathbb{F}_{13}$:

$$M = \begin{pmatrix} 12 & 5 & 4 & 8 \\ 11 & 2 & 8 & 9 \\ 4 & 6 & 4 & 8 \\ 2 & 9 & 3 & 12 \end{pmatrix}.$$

We have $\operatorname{Tr} M = 4$ and $\det M = 3$. These must be the sum and the product of the eigenvalues of M, respectively. Indeed, the eigenvalues (computed using methods described below) are: $1, 3, 5, 8$.

Recall that a matrix M is called *symmetric* if $M = M^T$, which means that it is symmetric (in the common sense) with respect to its main diagonal.

Proposition 3.6.8. *If M is a symmetric matrix with real coefficients, then all the eigenvalues of M are real.*

Proof. Let $\lambda \in \mathbb{C}$ be an eigenvalue of M and $v \in \mathbb{C}^n$ be an associated eigenvector. We will show that $\lambda = \overline{\lambda}$, which implies that λ is a real number. To this end, denote by $\overline{v}$

a vector constructed from v by conjugating all its entries and compute the dot product $\overline{v} \bullet v$:

$$\lambda \cdot \overline{v} \bullet v = \overline{v} \bullet (\lambda \cdot v) = \overline{v}^T \cdot A \cdot v = \left(A^T \cdot \overline{v}\right)^T \cdot v$$

now use the fact that A is symmetric and real

$$= \left(A \cdot \overline{v}\right)^T \cdot v = \left(\overline{A} \cdot \overline{v}\right)^T \cdot v = \left(\overline{\lambda} \cdot \overline{v}\right)^T \cdot v = \overline{\lambda} \cdot \overline{v} \bullet v.$$

Since v is a nonzero vector, $\|v\|^2 = \overline{v} \bullet v \neq 0$. It follows that $\overline{\lambda} = \lambda$. □

Eigenvalues and matrix exponentiation

There is close relation between eigenvalues and matrix exponentiation. We begin with a trivial fact.

Proposition 3.6.9. *If $\lambda_1, \ldots, \lambda_m$ are eigenvalues of a matrix M, then $\lambda_1^k, \ldots, \lambda_m^k$ are eigenvalues of M^k for every $k \geq 1$.*

Proof. We use induction on k. The assertion is trivial for $k = 1$. Take an eigenvalue $\lambda \in \{\lambda_1, \ldots, \lambda_m\}$ and let v be an associated eigenvector. Assume that we have already proved that λ^{k-1} is an eigenvalues of M^{k-1} and v is the associated eigenvector. Compute

$$M^k v = M^{k-1} \cdot (Mv) = M^{k-1} \cdot (\lambda v) = \lambda \cdot (M^{k-1} v) = \lambda \cdot \lambda^{k-1} v = \lambda^k v.$$ □

If we know a priori a diagonalization of M, we may compute any power M^k of M much faster than using exponentiation-by-squareing (Algorithm 1.1.10) coupled with matrix multiplication.

Observation 3.6.10. *If $M = V \cdot D \cdot V^{-1}$ and $D = \operatorname{diag}(a_1, \ldots, a_n)$ is a diagonal matrix, then for every $k \geq 1$ one has*

$$M^k = V \cdot D^k \cdot V^{-1} \qquad \text{and} \qquad D^k = \operatorname{diag}(a_1^k, \ldots, a_n^k).$$

In order to prove the correctness of the formula, just expand $(V \cdot D \cdot V^{-1})^k$ and cancel neighboring terms:

$$M^k = \left(V \cdot D \cdot V^{-1}\right)^k = V \cdot D \cdot (V^{-1} \cdot V) \cdot D \cdot V^{-1} \cdots V \cdot D \cdot V^{-1} = V \cdot D^k \cdot V^{-1}.$$

Example 43. Take a matrix M over a finite field $\mathbb{F}_{13}$ of the following form

$$M = \begin{pmatrix} 12 & 5 & 4 & 8 \\ 11 & 2 & 8 & 9 \\ 4 & 6 & 4 & 8 \\ 2 & 9 & 3 & 12 \end{pmatrix}.$$

Using methods described further in this section, one may compute the eigenvalues of M to be $1, 3, 5, 8$. The corresponding eigenvectors are:

$$v_1 = (1, 3, 9, 2), \quad v_2 = (4, 5, 8, 3), \quad v_3 = (2, 6, 0, 1), \quad v_4 = (6, 0, 0, 10).$$

Proposition 3.6.4 asserts that M is similar to $D = \operatorname{diag}(1, 3, 5, 8)$:

$$M = \begin{pmatrix} 1 & 4 & 2 & 6 \\ 3 & 5 & 6 & 0 \\ 9 & 8 & 0 & 0 \\ 2 & 3 & 1 & 10 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 8 \end{pmatrix} \cdot \begin{pmatrix} 1 & 4 & 2 & 6 \\ 3 & 5 & 6 & 0 \\ 9 & 8 & 0 & 0 \\ 2 & 3 & 1 & 10 \end{pmatrix}^{-1}.$$

Compute M^{1938} using the previous observation:

$$\begin{aligned} M^{1938} &= \begin{pmatrix} 1 & 4 & 2 & 6 \\ 3 & 5 & 6 & 0 \\ 9 & 8 & 0 & 0 \\ 2 & 3 & 1 & 10 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 8 \end{pmatrix}^{1938} \cdot \begin{pmatrix} 1 & 4 & 2 & 6 \\ 3 & 5 & 6 & 0 \\ 9 & 8 & 0 & 0 \\ 2 & 3 & 1 & 10 \end{pmatrix}^{-1} \\ &= \begin{pmatrix} 1 & 4 & 2 & 6 \\ 3 & 5 & 6 & 0 \\ 9 & 8 & 0 & 0 \\ 2 & 3 & 1 & 10 \end{pmatrix} \cdot \begin{pmatrix} 1^{1938} & 0 & 0 & 0 \\ 0 & 3^{1938} & 0 & 0 \\ 0 & 0 & 5^{1938} & 0 \\ 0 & 0 & 0 & 8^{1938} \end{pmatrix} \cdot \begin{pmatrix} 7 & 4 & 5 & 1 \\ 10 & 2 & 1 & 7 \\ 12 & 3 & 1 & 11 \\ 10 & 10 & 9 & 11 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 4 & 2 & 6 \\ 3 & 5 & 6 & 0 \\ 9 & 8 & 0 & 0 \\ 2 & 3 & 1 & 10 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 12 & 0 \\ 0 & 0 & 0 & 12 \end{pmatrix} \cdot \begin{pmatrix} 7 & 4 & 5 & 1 \\ 10 & 2 & 1 & 7 \\ 12 & 3 & 1 & 11 \\ 10 & 10 & 9 & 11 \end{pmatrix} \\ &= \begin{pmatrix} 2 & 11 & 5 & 6 \\ 12 & 4 & 1 & 11 \\ 0 & 0 & 1 & 0 \\ 10 & 2 & 0 & 6 \end{pmatrix}. \end{aligned}$$

By the way, since we are working in a finite field, exponentiation of the elements on the diagonal of D is extremely fast when one uses methods described in the next chapter.

Surprisingly we may go the other way round and use exponentiation to find the eigenvalues of a matrix M of size $n \times n$, providing that M has precisely n eigenvalues of distinct absolute values. Let $\lambda_1, \ldots, \lambda_n \in \mathbb{C}$ be the eigenvalues of M and assume that $|\lambda_i| \neq |\lambda_j|$ for $i \neq j$. Without loss of generality, we may assume that they are sorted in a descending order with respect to their absolute values:

$$|\lambda_1| > |\lambda_1| > \cdots > |\lambda_n| \geq 0.$$

It follows from Proposition 3.6.3 that there is a basis consisting of associated eigenvectors $v_1, \ldots, v_n$. Without loss of generality we may assume that each v_i has a unit length:

$$M \cdot v_i = \lambda_i \cdot v_i \quad \text{and} \quad \|v_i\| = 1 \qquad \text{for every} \quad i \in \{1, \ldots, n\}.$$

Pick by random a vector w_0 of length $\|w_0\| = 1$ and express it as a linear combination of the eigenvectors of M:

$$w_0 = a_1 v_1 + a_2 v_2 + \cdots + a_n v_n.$$

This expression is unique, because $v_1, \ldots, v_n$ form a basis of our vector space. Since w_0 is a nonzero vector, at least one a_i is nonzero, too. Set $j := \min\{1 \le i \le n \,|\, a_i \neq 0\}$ and compute $\hat{w}_1 := M \cdot w_0$. We have

$$\begin{aligned}\hat{w}_1 &= M \cdot \big(a_j v_j + a_{j+1} v_{j+1} + \cdots + a_n v_n\big) \\ &= a_j \cdot (M v_j) + a_{j+1} \cdot (M v_{j+1}) + \cdots + a_n \cdot (M v_n) \\ &= a_j \lambda_j v_j + a_{j+1} \lambda_{j+1} v_{j+1} + \cdots + a_n \lambda_n v_n.\end{aligned}$$

If $\hat{w}_1 = 0$, then necessarily $j = n$ and $\lambda_n = 0$. Consequently w_0 is an eigenvector associated to λ_n. Thus, without loss of generality we may assume that $\hat{w}_1$ is nonzero. Normalize it, taking $w_1 := {}^{\hat{w}_1}/_{\|\hat{w}_1\|}$ and write

$$w_1 = \frac{a_j \lambda_j}{\|\hat{w}_1\|} \cdot \Big(v_j + \frac{a_{j+1}\lambda_{j+1}}{a_j \lambda_j} \cdot v_{j+1} + \cdots + \frac{a_n \lambda_n}{a_j \lambda_j} \cdot v_n\Big).$$

Next take $\hat{w}_2 := M \cdot w_2$, $w_2 := {}^{\hat{w}_2}/_{\|\hat{w}_2\|}$ and so on. This way we recursively build a sequence $(w_k)_{k \ge 0}$, where

$$w_k := \frac{M \cdot w_{k-1}}{\|M \cdot w_{k-1}\|} = \frac{a_j \lambda_j^k}{\|\hat{w}_k\|} \cdot \Big(v_j + \frac{a_{j+1}\lambda_{j+1}^k}{a_j \lambda_j^k} \cdot v_{j+1} + \cdots + \frac{a_n \lambda_n^k}{a_j \lambda_j^k} \cdot v_n\Big) \qquad \text{and} \qquad \|w_k\| = 1.$$

By assumption $|\lambda_j| > |\lambda_{j+1}| > \cdots > |\lambda_n|$, therefore $\lim_{k\to\infty} {}^{\lambda_i^k}/_{\lambda_j^k} = 0$ for every $i > j$. Consequently, for k sufficiently large, one has

$$w_k \approx u_k \cdot v_j, \qquad \text{where} \quad u_k = \frac{\lambda_j^k}{|\lambda_j^k|} = \cos(k\varphi) + i \sin(k\varphi).$$

Here, $\varphi = \arg \lambda_j$ is the argument of the complex number λ_j. Thus, in general, the sequence $(w_k)_{k \ge 0}$ is not convergent! In a sense, it "rotates". Yet still, for every sufficiently large index k, an element w_k approximates an eigenvector $u_k v_j$, a linear combination of v_j. If there is an integer m such that $m\varphi = 2\pi$, then the sequence $(w_k)_{k \ge 0}$ consists of m subsequences: $(w_{mk})_{k \ge 0}, (w_{mk+1})_{k \ge 0}, \ldots, (w_{mk+m-1})_{k \ge 0}$. Each one converges to a different multiple of v_j. In a special case, when λ_j is a real number, then $\varphi = \arg \lambda_j$ is either 0 or π. In the former case, $(w_k)_{k \ge 0}$ is simply convergent, while in the later case it consists of two alternating convergent subsequences. Therefore, if we know a priori that all the eigenvalues of a given matrix M are real (e.g. M is symmetric, see Proposition 3.6.8), it suffices to consider every second element of the sequence $(w_k)_{k \ge 0}$. Let us summarize the above discussion in form of an algorithm.

Algorithm 3.6.11. *Given a matrix M of size $n \times n$, that has n real eigenvalues of distinct absolute values, this algorithm returns and approximation (up to a given precision ε) of an eigenvector v of M together with its associated eigenvalue λ.*

1. *Pick a random real vector w_0 of a unit length and set $w_{-2} := -w_0$;*
2. *if $M \cdot w_0$ is the zero vector, then return $v = w_0$ and $\lambda = 0$;*
3. *initialize a counter $i := 0$;*
4. *as long as $\|w_i - w_{i-2}\| > \varepsilon$ repeat the following steps:*
   a) *compute $\hat{w}_{i+2} := M \cdot (M \cdot w_i)$;*
   b) *normalize it taking $w_{i+2} := {}^{\hat{w}_{i+2}}/_{\|\hat{w}_{i+2}\|}$;*
   c) *increment the counter setting $i := i + 2$;*
5. *return $v = w_i$ and $\lambda = v^T \cdot M \cdot v$.*

Example 44. Take a symmetric matrix

$$M = \begin{pmatrix} 8 & 8 & 3 & 11 & 8 \\ 8 & 6 & 3 & 7 & 7 \\ 3 & 3 & 6 & 4 & 6 \\ 11 & 7 & 4 & 6 & 8 \\ 8 & 7 & 6 & 8 & 8 \end{pmatrix}$$

and pick a vector $w_0 = (-2, -6, -1, 2, 0)$. Compute the consecutive approximations of an eigenvector of M:

$$\begin{aligned} w_2 &= (-0.52949, -0.42262, -0.27229, -0.47998, -0.48632) \\ w_4 &= (-0.51913, -0.42224, -0.27531, -0.48844, -0.48769) \\ w_6 &= (-0.51897, -0.42223, -0.27539, -0.48856, -0.48771) \\ w_8 &= (-0.51896, -0.42223, -0.27539, -0.48856, -0.48771) \\ w_{10} &= (-0.51896, -0.42223, -0.27539, -0.48856, -0.48771) \end{aligned}$$

Now, $v := w_{10}$ is a good approximation of an eigenvector of M associated to an eigenvalue $\lambda \approx v^T \cdot M \cdot v \approx 33.975$.

Sylvester theorem to compute $f(M)$ via eigens (?)

Location of eigenvalues

The set $\operatorname{Spec} M$ of all eigenvalues of a given matrix M is called the *spectrum* of M. The maximum of absolute values of eigenvalues of M is called the *spectral radius* of M and denoted $\rho(M)$, i.e.

$$\rho(M) := \max\{|\lambda| \mid \lambda \in \operatorname{Spec} M\}$$

Providing, of course, that the notion of absolute value makes sense, i.e. the entries of M lie in $\mathbb{C}$ or one of its subfields. Below we present a few theorems that allow us

to localize (in the complex plane) eigenvalues of a given matrix. Let $M = (m_{ij})$ be a matrix with complex entries of size $n \times n$. For every index $1 \le i \le n$, by $s_i := \sum_{j=1}^{n} |m_{ij}|$ denote the sum of absolute values of the entries in the i-th row. Further, define

$$D_i := \big\{ z \in \mathbb{C} \,|\, |z - m_{ii}| \le s_i - |m_{ii}| \big\}.$$

The sets $D_1, \ldots, D_n \subset \mathbb{C}$ are closed discs, called *Gershgorin discs* (or *Gershgorin circles*).

Theorem 3.6.12 (Gershgorin, 1931)**.** *If λ is an eigenvalue of M, then $\lambda \in D_i$ for some $1 \le i \le n$.*

Proof. Fix an eigenvalue λ of M and let $v = (x_1, \ldots, x_n)^T$ be a corresponding eigenvector. Since every nonzero multiple of v is again an eigenvector (and λ remains to be the associated eigenvalue), without loss of generality we may assume that the coordinate of v of the maximal absolute value equals 1. Let $i \in \{1, \ldots, n\}$ be the index of this coordinate, hence

$$x_i = 1 \qquad \text{and} \qquad |x_j| \le 1 \quad \text{for } j \ne i.$$

Expand the i-th row of a product $M \cdot v$ and use the fact that $M \cdot v = \lambda \cdot v$ to write

$$\lambda = \lambda x_i = \sum_{1 \le j \le n} m_{ij} x_j = \sum_{j \ne i} m_{ij} x_j + m_{ii}.$$

It follows that

$$|\lambda - m_{ii}| = \left| \sum_{j \ne i} m_{ij} x_j \right| \le \sum_{j \ne i} |m_{ij}| \cdot |x_j| \le \sum_{j \ne i} |m_{ij}| = s_i - |m_{ii}|.$$

This proves the theorem. □

A set of matrices (respectively vectors) of a fixed size admits a partial order in which matrices (vectors) are compared coefficient-wisely. If $A = (a_{ij})$ and $B = (b_{ij})$, we write $A \le B$ if $a_{ij} \le b_{ij}$ for all admissible indices i, j. Likewise, we write $A < B$ if $a_{ij} < b_{ij}$, again for all i's and j's. (Be aware, however, that $A \le B$ and $A \ne B$ does not imply $A < B$.) In what follows, we say that a matrix (respectively a vector) M is *positive* if all its entries are positive, i.e. if $(0) < M$, where (0) denotes a zero matrix of the same size as M. Analogously, a matrix (vector) is *non-negative* if all its entries are non-negative, equivalently if $(0) \le M$. It turns out that a positive matrix always has a single dominant and positive eigenvalue.

Theorem 3.6.13 (Perron, 1907)**.** *Let M be a positive square matrix, then there exists an eigenvalue $\Lambda > 0$ of M such that every eigenvalue $\lambda \in \mathbb{C}$, $\lambda \ne \Lambda$ of M satisfies $|\lambda| < \Lambda$.*

In a more fancy lamguage, Perron theorem says that the spectral radius of a positive matrix is realized by its single eigenvalue. (For an arbitrary matrix, the spectral radius is the maximal absolute value of eigenvalues, but the eigenvalue for which this maximum is attained does not have to be positive or even real.) The case of non-negative matrices is slightly more complicated. First we need to introduce a new term.

Definition 3.6.14. A non-negative matrix M is called *irreducible* if for every pair i, j of indices, there is an exponent $k \geq 1$ such that the corresponding entry of M^k is strictly positive.

Theorem 3.6.15 (Frobenius, 1912)**.** *Let M be a non-negative, irreducible matrix. Then there is an eigenvalue $\Lambda > 0$ such that every eigenvalue $\lambda \in \mathbb{C}$ satisfies, $|\lambda| \leq \Lambda$. Moreover, the set of all eigenvalues of M is invariant under rotation by angle $2\pi/k$, where k is the number of eigenvalues having the absolute values equal Λ.*

Remark 33. Theorems 3.6.13 and 3.6.15 are often together referred to as "Perron-Frobenius Theory".

The eigenvalue Λ, appearing in the two previous theorems, is called the *Perron-Frobenius eigenvalue*. It's value is explicitly given by the following theorem.

Theorem 3.6.16 (Wielandt, 1950)**.** *Let M be a matrix of size $n \times n$ that is either positive or non-negative and irreducible. The Perron–Frobenius eigenvalue of M equals*

$$\Lambda = \min_{v>0} \max_{1 \leq i \leq n} \frac{(M \cdot v)_i}{v_i},$$

here the subscript i stands for the i-th coordinate of a given vector.

Let $M = (m_{ij})$ be a square matrix with complex entries. Denote by $|M| := \left(|m_{ij}|\right)$ a matrix obtained form M by replacing each entry by its absolute value. It is clear that $M \leq |M|$ and $|M|$ is non-negative. If in addition $|M|$ is irreducible, then we may apply Frobenius theorem to it to get the maximal eigenvalue Λ of $|M|$. It turns out that Λ not only dominates the absolute values of the eigenvalues of $|M|$ but also of M itself.

Theorem 3.6.17 (Wielandt, 1950)**.** *Use the above notation. If $|M|$ is irreducible and $\Lambda > 0$ is its eigenvalue of maximal absolute value, then*

$$|\lambda| \leq \Lambda$$

for every eigenvalue λ of M.

In other words, this theorem asserts that the spectrum radius of $|M|$ dominates the spectral radius of M, i.e. $\rho(M) \leq \rho(|M|)$. The proofs of Theorems 3.6.13–3.6.17 exceed the scope of this book. An interested reader can find them in [28, 89].

example

3.7 Short vectors

Lattices

Let V be a (finitely dimensional) vector space over $\mathbb{R}$ with some basis $\mathscr{B} = \{v_1, \ldots, v_m\}$. Among all the possible linear combinations of basis vectors we distinguish those that have integral coefficients.

Definition 3.7.1. If $\mathscr{B} = \{v_1, \ldots, v_m\}$ is a basis of a real vector space V, then the subset

$$L := \mathbb{Z}v_1 + \cdots + \mathbb{Z}v_m = \left\{ n_1 v_1 + \cdots + n_m v_m \,|\, n_1, \ldots, n_m \in \mathbb{Z} \right\}$$

is called a *lattice* with the basis $\mathscr{B}$.

Example 45. Take $V = \mathbb{R}^2$ and let $v_1 = (5, 7)^T$, $v_2 = (6, 8)^T$. Since $\det(v_1, v_2) = -2 \neq 0$, the two vectors form a basis of V. Figure 3.1 shows the corresponding lattice $L = \mathbb{Z}v_1 + \mathbb{Z}v_2$. One can easily notice that, although v_1, v_2 span the lattice, they are not the shortest vectors. We claim that the lattice L can be generated also by $w_1 = (1, 1)^T$ and $w_2 = (-1, 1)^T$. Indeed, we have

$$w_1 = -1v_1 + 1v_2, \qquad w_2 = 7v_1 + -6v_2.$$

On the other hand

$$v_1 = 6w_1 + 1w_2, \qquad v_2 = 7w_1 + 1w_2.$$

Hence, it is clear that the lattices $\mathbb{Z}v_1 + \mathbb{Z}v_2$ and $\mathbb{Z}w_1 + \mathbb{Z}w_2$ coincide. We may express the correspondence between the two lattice bases in matrix notation. Treat v_1, v_2, w_1, w_2 as raw vectors and write

$$\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} -1 & 1 \\ 7 & -6 \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}.$$

Observe that the above change-of-basis matrix is invertible over $\mathbb{Z}$.

The shortest vectors of a lattice are important in many applications (see for example Section 5.6). The previous example shows that they do not have to be the vectors used to generate the lattice in the first place. The problem of finding the shortest vectors of an arbitrary lattice is known to be NP-hard. Thus, there is no hope for an efficient algorithm. Fortunately, in many cases it suffices to find vectors that are only "short enough" (a rigorous notion will be introduced later). This task is manageable and has a polynomial-time (with respect to the bits-size of generators) solution.

Gram–Schmidt orthogonalization

Before we discuss the algorithm for computing short vectors of a lattice, we need to recall a well known procedure that constructs an orthogonal basis of a real vector space equipped with the (standard) dot product, denoted $\bullet$ hereafter.

Let $V \subseteq \mathbb{R}^n$ be a vector space with basis $\mathscr{B} = \{v_1, \ldots, v_m\}$, $m \leq n$. Our task is to construct a new basis $\mathscr{B}^* = \{w_1, \ldots, w_m\}$ of V such that every two distinct vectors are orthogonal and the subspaces generated by $v_1, \ldots, v_k$ and $w_1, \ldots, w_k$ coincide for every k, i.e.:

- $w_i \perp w_j$ for $i \neq j$,
- $\operatorname{lin}\{v_1, \ldots, v_k\} = \operatorname{lin}\{w_1, \ldots, w_k\}$ for $1 \leq k \leq m$.

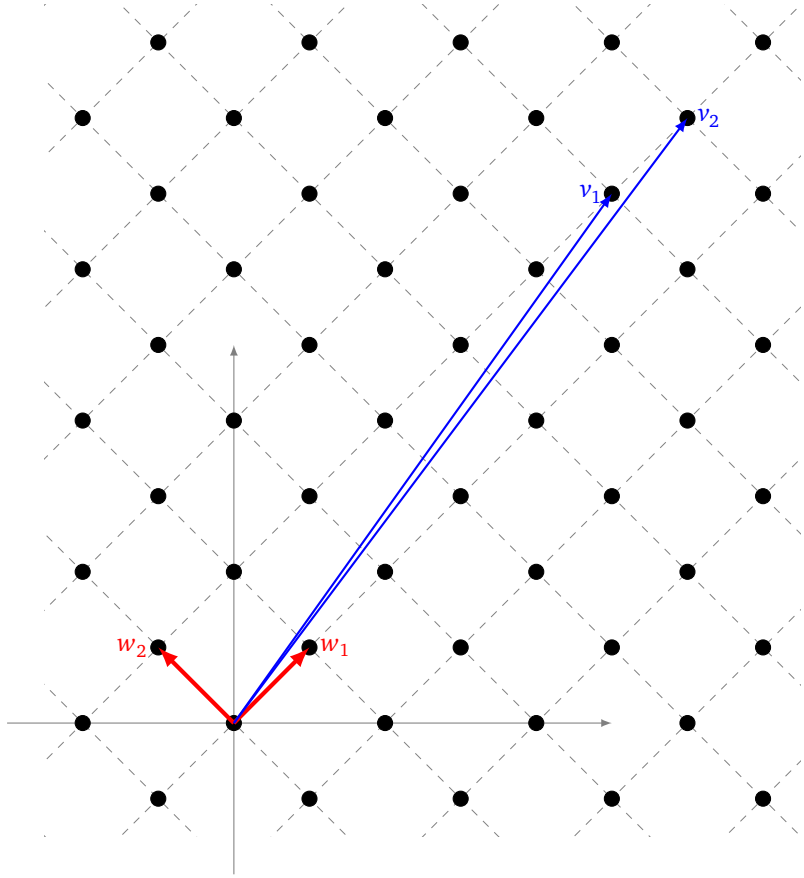


Figure 3.1: Lattice from Example 45

By the second condition, it is clear that w_1 must be a linear combination of v_1. Hence, we just take $w_1 := v_1$. Suppose that we have constructed pairwise orthogonal (thus linearly independent) vectors $w_1, \ldots, w_{k-1}$ that span $V_{k-1} := \mathrm{lin}\{v_1, \ldots, v_{k-1}\}$. Take w_k to be the orthogonal projection of v_k onto the orthogonal completion $V_{k-1}^{\perp}$ of V_{k-1},

$$V_{k-1}^{\perp} := \left\{ v \in V \mid v \perp w \text{ for all } w \in V_{k-1} \right\}.$$

Lemma 3.7.2. *We have*

$$w_k = v_k - \sum_{i=1}^{k-1} \frac{v_k \bullet w_i}{w_i \bullet w_i} \cdot w_i.$$

Proof. The vector v_k decomposes uniquely into a sum

$$v_k = v_{\parallel} + v_{\perp},$$

where $v_{\parallel} \in V_{k-1}$ is the projection of v_k onto V_{k-1} and $v_{\perp} \in V_{k-1}^{\perp}$ is its projection onto the orthogonal completion. Obviously $v_{\perp}$ is just w_k we are looking for. Now, since $v_{\parallel} \in V_{k-1}$, we may write it as a linear combination of the basis vectors $w_1, \ldots, w_{k-1}$:

$$v_{\parallel} = \sum_{i=1}^{k-1} \alpha_i w_i,$$

for some $\alpha_1, \ldots, \alpha_{k-1} \in \mathbb{R}$. Fix $j < k$ and compute the dot product $v_k \bullet w_j$:

$$v_k \bullet w_j = \left(\sum_{i=1}^{k-1} \alpha_i w_i + w_k \right) \bullet w_j = \sum_{i=1}^{k-1} \alpha_i w_i \bullet w_j + w_k \bullet w_j = \alpha_j w_j \bullet w_j.$$

Thus $\alpha_j = \frac{v_k \bullet w_j}{w_j \bullet w_j}$ for every $j < k$. This proves the lemma. □

It follows immediately that the vectors $w_1, \ldots, w_k$ are pairwise orthogonal and they generate V_k. The fact that w_k is a projection of v_k onto $V_{k-1}^{\perp}$ implies that $\|v_k\| \geq \|w_k\|$. Moreover, using matrix matrix notation, we have

$$\begin{pmatrix} v_1 \\ \vdots \\ v_m \end{pmatrix} = \begin{pmatrix} 1 & & & \\ \mu_{2,1} & 1 & & \\ \vdots & \ddots & \ddots & \\ \mu_{m,1} & \cdots & \mu_{m,m-1} & 1 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ \vdots \\ w_m \end{pmatrix},$$

where

$$\mu_{ij} := \frac{v_i \bullet w_j}{w_j \bullet w_j}$$

and $v_1, \ldots, v_m, w_1, \ldots, w_m$ are treated as row vectors. The change-of-basis matrix has determinant one, hence is invertible. Summarizing our discussion, we have proved:

Theorem 3.7.3. *Lat $\mathscr{B} := \{v_1, \ldots, v_m\}$ be a basis of a real vector space V equipped with a standard dot product. Define $w_1, \ldots, w_m$ by a recursive formula*

$$w_1 := v_1, \qquad w_k := v_k - \sum_{i=1}^{k-1} \mu_{ki} \cdot w_i = v_k - \sum_{i=1}^{k-1} \frac{v_k \bullet w_i}{w_i \bullet w_i} \cdot w_i.$$

Then the following conditions hold:

1. *Vectors $w_1, \ldots, w_m$ are pairwise orthogonal.*
2. *For every $1 \leq k \leq m$, the set $w_1, \ldots, w_k$ is a basis of $V_k := \operatorname{lin}\{v_1, \ldots, v_k\}$.*
3. *For every $1 \leq k \leq m$, we have $\|v_k\| \geq \|w_k\|$.*

4. *For every* $1 \le k \le m$, *we have* $v_k \bullet w_k = w_k \bullet w_k$.
5. *For every* $1 \le i < k \le m$, *we have* $v_i \bullet w_k = 0$.

Example 46. We will demonstrate Gram–Schmidt process for vectors

$$v_1 = \begin{pmatrix} 1 \\ 1 \\ -2 \\ 1 \end{pmatrix}, \qquad v_2 = \begin{pmatrix} 3 \\ 2 \\ 1 \\ 1 \end{pmatrix}, \qquad v_3 = \begin{pmatrix} -2 \\ 0 \\ 1 \\ 3 \end{pmatrix}, \qquad v_4 = \begin{pmatrix} -1 \\ -1 \\ 0 \\ 4 \end{pmatrix}.$$

We begin with $w_1 = v_1 = (1, 1, -2, 1)^T$. Then we compute

$$\begin{aligned} w_2 &= v_2 - \mu_{21} w_1 = \left(\frac{17}{7}, \frac{10}{7}, \frac{15}{7}, \frac{3}{7} \right)^T, \\ w_3 &= v_3 - \mu_{31} w_1 - \mu_{32} w_2 = \left(-\frac{141}{89}, \frac{27}{89}, \frac{85}{89}, \frac{284}{89} \right)^T, \\ w_4 &= v_4 - \mu_{41} w_1 - \mu_{42} w_2 - \mu_{43} w_3 = \left(\frac{912}{1219}, -\frac{1653}{1219}, -\frac{57}{1219}, \frac{627}{1219} \right)^T. \end{aligned}$$

It is clear that if we organize our vectors as rows (or columns) of a matrix, then Gram-Schmidt orthogonalization can be performed in situ using elementary row (respectively column) transformations. For vectors from the previous example treated again as rows of a matrix, successive matrices are

$$\begin{pmatrix} 1 & 1 & -2 & 1 \\ 3 & 2 & 1 & 1 \\ -2 & 0 & 1 & 3 \\ -1 & -1 & 0 & 4 \end{pmatrix} \rightsquigarrow \begin{pmatrix} 1 & 1 & -2 & 1 \\ \frac{17}{7} & \frac{10}{7} & \frac{15}{7} & \frac{3}{7} \\ -2 & 0 & 1 & 3 \\ -1 & -1 & 0 & 4 \end{pmatrix}$$

$$\rightsquigarrow \begin{pmatrix} 1 & 1 & -2 & 1 \\ \frac{17}{7} & \frac{10}{7} & \frac{15}{7} & \frac{3}{7} \\ -\frac{141}{89} & \frac{27}{89} & \frac{85}{89} & \frac{284}{89} \\ -1 & -1 & 0 & 4 \end{pmatrix} \rightsquigarrow \begin{pmatrix} 1 & 1 & -2 & 1 \\ \frac{17}{7} & \frac{10}{7} & \frac{15}{7} & \frac{3}{7} \\ -\frac{141}{89} & \frac{27}{89} & \frac{85}{89} & \frac{284}{89} \\ \frac{912}{1219} & -\frac{1653}{1219} & -\frac{57}{1219} & \frac{627}{1219} \end{pmatrix}.$$

The rows of the resulting matrices form the orthogonal basis.

Short vectors in low dimensions

Lagrange-Gauss algorithm, Nguyen–Stehlé algorithm

LLL algorithm

In this subsection we present an algorithm, due to A. Lenstra, H. Lenstra and L. Lovász, that finds "relatively short" vectors in a lattice. Let $V \subset \mathbb{R}^n$ be a vector space with

a (ordered) basis $\mathscr{B} = \{v_1, \ldots, v_m\}$ and let $L := \mathbb{Z}v_1 + \cdots + \mathbb{Z}v_m$ be an associated lattice. Perform Gram–Schmidt orthogonalization on $\mathscr{B}$. Let $\mathscr{B}^* = \{w_1, \ldots, w_m\}$ be the resulting orthogonal basis. Denote

$$\mu_{ij} := \frac{v_i \bullet w_j}{w_j \bullet w_j} \qquad \text{and} \qquad \big\lceil \mu_{ij} \big\rfloor := \big\lfloor \mu_{ij} + \tfrac{1}{2} \big\rfloor.$$

Then $\lceil \mu_{ij} \rfloor$ is the integer nearest to μ_{ij}.

Lemma 3.7.4. *With the above notation, if $v \in L$ is any nonzero vector of the lattice L, then*

$$\|v\| \geq \min\big\{\|w_1\|, \ldots, \|w_m\|\big\}.$$

Proof. Let $v = n_1 v_1 + \cdots + n_m v_m$ with $n_1, \ldots, n_m \in \mathbb{Z}$ not all equal zero. Let k be the highest index such that $n_k \neq 0$. Substitute $\sum_{j=1}^{i} \mu_{ij} w_j$ for v_i and write

$$v = \sum_{i=1}^{k} n_i \sum_{j=1}^{i} \mu_{ij} w_j = n_k w_k + \sum_{j=1}^{k-1} \Big(\sum_{i=1}^{k} n_i \mu_{ij} \Big) \cdot w_j.$$

This yields

$$\|v\|^2 = v \bullet v = \left(n_k w_k + \sum_{j=1}^{k-1} \Big(\sum_{i=1}^{k} n_i \mu_{ij} \Big) \cdot w_j \right) \bullet \left(n_k w_k + \sum_{j=1}^{k-1} \Big(\sum_{i=1}^{k} n_i \mu_{ij} \Big) \cdot w_j \right)$$

Vectors $w_1, \ldots, w_k$ are pairwise orthogonal, hence:

$$= n_k^2 \|w_k\|^2 + \sum_{j=1}^{k-1} \Big(\sum_{i=1}^{k} n_i \mu_{ij} \Big)^2 \cdot \|w_j\|^2 \geq n_k^2 \|w_k\|^2.$$

Now, n_k is a nonzero integer and so $\|v\|^2 \geq \|w_k\|^2$. □

If the Gram–Schmidt orthogonalization of a lattice basis $\mathscr{B}$ produces a basis $\mathscr{B}^*$ that spans the same lattice, then the previous lemma implies that one of the elements of $\mathscr{B}^*$ must be the shortest vector of L. In general, however, the chances that the obtained orthogonal vectors sit in L are infinitesimal. The idea of the LLL algorithm is to construct a basis that is "sufficiently close to orthogonal". One decides whether a basis $\mathscr{B}$ is good enough by considering the variation of lengths of vectors produced from $\mathscr{B}$ by the Gram–Schmidt process. If the lengths do not vary too much, then each vector v_k is "close enough" to its projection $w_k = v_\perp$ (see the previous subsection) onto $V_{k-1}^\perp$. This means that the basis is "close to orthogonal". In order to make the discussion rigorous, we need the following definition.

Definition 3.7.5. Let $\mathscr{B} = \{v_1, \ldots, v_m\}$ be a lattice basis and $\mathscr{B}^* = \{w_1, \ldots, w_m\}$ the corresponding Gram–Schmidt orthogonal basis. We say that the basis $\mathscr{B}$ is *LLL-reduced* if the following two conditions are satisfied:

1. $|\mu_{i,j}| \leq 1/2$ for all $1 \leq j < i \leq m$,
2. $\|w_i + \mu_{i,i-1} w_{i-1}\|^2 \geq 3/4 \cdot \|w_{i-1}\|^2$ for every i.

Remark 34. The constant $3/4$ in the condition 2 of the previous definition look like a "magic number". In fact it can be replaced by any $\delta \in \left(1/4, 1\right)$, but $3/4$ is a typical value used for a description of the algorithm since it results in nicely looking formulas (see the next lemma). In real-life computations, it can be quite advantageous to select a bigger δ.

Lemma 3.7.6. *If $\mathscr{B} = \{v_1, \ldots, v_m\}$ is an LLL-reduced basis and $\mathscr{B}^* = \{w_1, \ldots, w_m\}$ is the corresponding Gram–Schmidt orthogonal basis, then for all $1 \leq j < i \leq m$ we have*

$$\|w_j\|^2 \leq 2^{i-j} \cdot \|w_i\|^2.$$

Proof. Fix $i \leq m$. Vectors $w_1, \ldots, w_m$ are orthogonal, hence condition 2 of the definition yields

$$\begin{aligned}
\frac{3}{4}\|w_{i-1}\|^2 &\leq \|w_i + \mu_{i,i-1} w_{i-1}\|^2 \\
&= (w_i + \mu_{i,i-1} w_{i-1}) \bullet (w_i + \mu_{i,i-1} w_{i-1}) \\
&= w_i \bullet w_i + \mu_{i,i-1}^2 \cdot w_{i-1} \bullet w_{i-1} \\
&= \|w_i\|^2 + \mu_{i,i-1}^2 \|w_{i-1}\|^2 \leq \|w_i\|^2 + \frac{1}{4}\|w_{i-1}\|^2.
\end{aligned}$$

This shows an inequality $\|w_{i-1}\|^2 \leq 2 \cdot \|w_i\|^2$. The assertion follows now by induction. □

We can now explain what we meant saying that we would construct "sufficiently short vectors".

Theorem 3.7.7. *Let $\mathscr{B} = \{v_1, \ldots, v_m\}$ be an LLL-reduced basis for a lattice $L \subset \mathbb{R}^n$ and let λ be the length of the shortest vector in L, then*

$$\|v_1\| \leq 2^{\frac{m-1}{2}} \cdot \lambda.$$

Proof. By Lemmas 3.7.4 and 3.7.6 we have

$$\begin{aligned}
\|\lambda\| &\geq \min\left\{\|w_1\|, \|w_2\|, \ldots, \|w_m\|\right\} \\
&\geq \min\left\{\|w_1\|, 2^{\frac{1-2}{2}} \cdot \|w_1\|, \ldots, 2^{\frac{1-m}{2}} \cdot \|w_1\|\right\} = 2^{\frac{1-m}{2}} \cdot \|w_1\|.
\end{aligned}$$

The result follows now from the fact that $w_1 = v_1$. □

We are now ready to present an algorithm that transforms any lattice basis into an LLL-reduced one. The idea is to iterate over the elements of a given basis and check conditions of Definition 3.7.5. We make vectors to satisfy the first condition by replacing each one by a suitable linear combination with integer coefficients. Then we check the second condition and if it fails for some pair of vectors we swap them and step back.

Algorithm 3.7.8 (Lenstra–Lenstra–Lovász, 1992). *Given a basis $\mathscr{B} = \{u_1, \ldots, u_m\}$ of a lattice L, this algorithm constructs an LLL-reduced basis $\{v_1, \ldots, v_m\}$ of L.*

1. *Initialization:*
   a) *set $v_i := u_i$ for $1 \le i \le m$};*
   b) *construct the Gram–Schmidt orthogonal of basis $\mathscr{B}^* := \{w_1, \ldots, w_m\}$, where $w_1 := v_1$ and*

$$w_i := v_i - \sum_{j=1}^{i-1} \mu_{ij} \cdot w_j, \qquad \mu_{ij} := \frac{v_i \bullet w_j}{\|w_j\|^2}$$

   for all $1 \le i \le m$ and $1 \le j < i$;
   c) *compute and cache the squared norms $\|w_1\|^2, \ldots, \|w_m\|^2$ of the orthogonal vectors.*
2. *Main loop: starting from $k = 2$ repeat the following steps as long as $k \le m$:*
   a) *check the condition 1 of Definition 3.7.5, if for some l one has $|\mu_{k,l}| > 1/2$, then replace v_k by*

$$v'_k := v_k - \sum_{l=1}^{i} \lceil \mu_{k,l} \rfloor \cdot v_l, \qquad \text{where } i = \max\bigl\{l \mid |\mu_{k,l}| > 1/2\bigr\},$$

   and update the Gram–Schmidt coefficients using Lemma 3.7.9;
   b) *check the condition 2 of Definition 3.7.5:*
      i. *if $\|w_k + \mu_{k,k-1} \cdot w_{k-1}\|^2 \ge 3/4 \cdot \|w_{k-1}\|^2$, then increment the index k and reiterate the main loop;*
      ii. *otherwise, swap the vector v_k and v_{k-1} and update the Gram–Schmidt orthogonal basis using Lemma 3.7.10;*
      iii. *update k setting $k := \max\{2, k-1\}$ and reiterate.*
3. *Return the LLL-reduced basis $\{v_1, \ldots, v_m\}$.*

Although it is perfectly possible to reconstruct the Gram–Schmidt basis from scratch every time one of v_k's changes (i.e. in steps 2a and 2(b)ii of the algorithm), such a naïve approach is rather inefficient. The following two lemmas, of quite a technical nature, provide close-form formulas for updating the Gram–Schmidt basis.

Lemma 3.7.9. *Let $\mathscr{B} = \{v_1, \ldots, v_m\}$ be a basis of a lattice L and $\mathscr{B}^* = \{w_1, \ldots, w_m\}$ be a corresponding Gram–Schmidt orthogonal basis. Fix an index k and assume that $\mathscr{B}' = \{v'_1, \ldots, v'_m\}$ is another basis of L, where*

$$v'_k = v_k - \sum_{l=1}^{i} \lceil \mu_{k,l} \rfloor \cdot v_l, \qquad \text{for some } i < k \tag{3.9}$$

and $v'_j = v_j$ for $j \ne k$. Then the following statements hold:

1. *The Gram–Schmidt orthogonalization of $\mathscr{B}'$ coincides with $\mathscr{B}^*$.*

2. *The Gram–Schmidt coefficients $\mu'_{ij} := \frac{v_i \bullet w'_j}{\|w'_j\|^2}$ satisfy:*

$$\begin{aligned}
&\mu'_{ij} = \mu_{ij} && \text{for } i \neq k \text{ and } j \neq k \\
&\mu'_{kj} = \mu_{kj} - \sum_{l=j+1}^{i} [\mu_{kl}]\mu_{lj} - [\mu_{kj}] && \text{for } j \leq i \\
&\mu'_{kj} = \mu_{kj} && \text{for } i < j < k.
\end{aligned}$$

Proof. Let $\mathscr{B}'^* = \{w'_1, \ldots, w'_m\}$ be the Gram–Schmidt orthogonal basis associated to $\mathscr{B}'$. It is obvious that $w'_j = w_j$ for $j \neq k$. We need only to prove that $w'_k = w_k$. We use induction on the upper limit i of the summation in Eq. (3.9). If $i = 1$, then

$$v'_k = v_k - [\mu_{k1}]v_1$$

and by Lemma 3.7.2

$$\begin{aligned}
w'_k &= v'_k - \sum_{j=1}^{k-1} \frac{v'_k \bullet w_j}{\|w'_j\|^2} \cdot w'_j \\
&= v_k - [\mu_{k1}]v_1 - \sum_{j=1}^{k-1} \frac{\left(v_k - [\mu_{k1}]v_1\right) \bullet w_j}{\|w_j\|^2} \cdot w_j \\
&= v_k - [\mu_{k1}]v_1 - \sum_{j=1}^{k-1} \frac{v_k \bullet w_j}{\|w_j\|^2} \cdot w_j + [\mu_{k1}] \cdot \frac{v_1 \bullet w_1}{\|w_1\|^2} \cdot w_1 \\
&= w_1.
\end{aligned}$$

Assume that we have proved the assertion for all natural numbers less than i. We then

have

$$\begin{aligned}
w'_k &= v'_k - \sum_{j=1}^{k-1} \frac{v'_k \bullet w_j}{w_j \bullet w_j} \cdot w_j \\
&= v_k - \sum_{l=1}^{i} [\mu_{kl}] v_l - \sum_{j=1}^{k-1} \left(\frac{1}{w_j \bullet w_j} \cdot \left(v_k - \sum_{l=1}^{i} [\mu_{kl}] v_l \right) \bullet w_j \right) \cdot w_j \\
&= \left(v_k - \sum_{j=1}^{k-1} \frac{v_k \bullet w_j}{w_j \bullet w_j} \cdot w_j \right) - \sum_{l=1}^{i} [\mu_{kl}] v_l + \sum_{j=1}^{k-1} \sum_{l=1}^{i} [\mu_{kl}] \frac{v_l \bullet w_j}{w_j \bullet w_j} \cdot w_j \\
&= w_k - \sum_{l=1}^{i} [\mu_{kl}] \left(v_l - \sum_{j=1}^{k-1} \frac{v_l \bullet w_j}{w_j \bullet w_j} \cdot w_j \right) \\
&= w_k - \sum_{l=1}^{i} [\mu_{kl}] \left(v_l - \sum_{j=1}^{l-1} \frac{v_l \bullet w_j}{w_j \bullet w_j} \cdot w_j - \frac{v_l \bullet w_l}{w_j \bullet w_j} \cdot w_l \right) \\
&= w_k - \sum_{l=1}^{i} [\mu_{kl}] (w_l - w_l) = w_k.
\end{aligned}$$

Now, we prove the second assertion. The only difference between bases $\mathscr{B}$ and $\mathscr{B}'$ is $v_k \neq v'_k$ and we have already shown that $\mathscr{B}^* = \mathscr{B}'^*$. Thus, it is clear that $\mu'_{ij} = \mu_{ij}$ for all i, j distinct from k. Take some $j < k$ and compute μ'_{kj}:

$$\begin{aligned}
\mu'_{kj} = \frac{v'_k \bullet w'_j}{\|w'_j\|^2} &= \frac{\left(v_k - \sum_{l=1}^{i} [\mu_{kl}] v_l \right) \bullet w_j}{\|w_j\|^2} \\
&= \frac{v_k \bullet w_j}{\|w_j\|^2} - \sum_{l=1}^{i} [\mu_{kl}] \frac{v_l \bullet w_j}{\|w_j\|^2}
\end{aligned}$$

If $j > l$, then $v_l \bullet w_j = 0$ and so $\mu'_{kj} = \mu_{kj}$ for $j > i$. On the other hand, for $j \leq i$ we have:

$$\begin{aligned}
&= \mu_{kj} - \sum_{l=j+1}^{i} [\mu_{kl}] \mu_{lj} - [\mu_{kj}] \cdot \frac{v_j \bullet w_j}{\|w_j\|^2} \\
&= \mu_{kj} - \sum_{l=j+1}^{i} [\mu_{kl}] \mu_{lj} - [\mu_{kj}]. \qquad \square
\end{aligned}$$

Lemma 3.7.10. *Let $\mathscr{B} = \{v_1, \ldots, v_m\}$ be a basis of a lattice L and $\mathscr{B}^* = \{w_1, \ldots, w_m\}$ be a corresponding Gram–Schmidt orthogonal basis. Assume that*

$$\mathscr{B}' = \{v'_1 = v_1, \ldots, v'_{k-1} = v_k, v'_k = v_{k-1}, \ldots, v'_m\}$$

is a basis of L obtained from $\mathscr{B}$ by swapping vectors v_{k-1} and v_k. If $\mathscr{B}'^* = \{w'_1, \dots, w'_m\}$ is a corresponding orthogonal basis and $\mu'_{ij} = \frac{v'_i \bullet w'_j}{\|w'_j\|^2}$, then the following statements hold:

1. *for $i < k-1$:*

$$w'_i = w_i, \qquad \|w'_i\|^2 = \|w_i\|^2, \qquad \mu'_{i,j} = \mu_{i,j} \quad \text{for } j < i.$$

2. *for $i = k-1$:*

$$w'_{k-1} = w_k + \mu_{k,k-1} w_{k-1}, \qquad \|w'_{k-1}\|^2 = \|w_k\|^2 + \mu_{k,k-1}^2 \|w_{k-1}\|^2, \qquad \mu'_{k-1,j} = \mu_{k,j} \quad \text{for } j < k-1.$$

3. *for $i = k$:*

$$w'_k = \frac{\|w_k\|^2}{\|w'_{k-1}\|^2} \cdot w_{k-1} - \mu_{k,k-1} \cdot \frac{\|w_{k-1}\|^2}{\|w'_{k-1}\|^2} \cdot w_k, \quad \|w'_k\|^2 = \frac{\|w_{k-1}\|^2 \cdot \|w_k\|^2}{\|w'_{k-1}\|^2},$$
$$\mu'_{k,j} = \mu_{k-1,j} \quad \text{for } j < k-1, \qquad \mu'_{k,k-1} = \mu_{k,k-1} \cdot \frac{\|w_{k-1}\|^2}{\|w'_{k-1}\|^2}.$$

4. *for $i > k$:*

$$w'_i = w_i, \qquad \|w'_i\|^2 = \|w_i\|^2, \qquad \mu'_{i,j} = \mu_{i,j} \quad \text{for } j < k-1 \text{ or } k < j < i,$$
$$\mu'_{i,k-1} = \mu_{k,k-1} \cdot (\mu_{i,k} - \mu_{k,k-1}\mu_{i,k}) + \mu_{i,k}, \quad \mu'_{i,k} = \mu_{i,k-1} - \mu_{i,k}\mu_{k,k-1}.$$

Proof. The first statement is trivial. As for the second one, by Lemma 3.7.2 we have

$$w'_{k-1} = v'_{k-1} - \sum_{j=1}^{k-2} \frac{v'_{k-1} \bullet w'_j}{\|w'_j\|^2} \cdot w'_j = v_k - \sum_{j=1}^{k-2} \frac{v_k \bullet w_j}{\|w_j\|^2} \cdot w_j = w_k + \frac{v_k \bullet w_{k-1}}{\|w_{k-1}\|^2} \cdot w_{k-1} = w_k + \mu_{k,k-1} w_{k-1}.$$

The formula for the squared norm $\|w'_{k-1}\|^2$ follows now by the orthogonality of w_k and w_{k-1}. The equality $\mu'_{k-1,j} = \mu_{k,j}$ is obvious.

Analogously we prove the third assertion:

$$\begin{aligned}
w'_k &= v'_k - \sum_{j=1}^{k-2} \frac{v'_k \bullet w'_j}{\|w'_j\|^2} \cdot w'_j - \frac{v'_k \bullet w'_{k-1}}{\|w'_{k-1}\|^2} \cdot w'_{k-1} \\
&= \left(v_{k-1} - \sum_{j=1}^{k-2} \frac{v_{k-1} \bullet w_j}{\|w_j\|^2} \cdot w_j^2 \right) - \frac{v_{k-1} \bullet (w_k + \mu_{k,k-1} w_{k-1})}{\|w'_{k-1}\|^2} \cdot (w_k + \mu_{k,k-1} w_{k-1}) \\
&= w_{k-1} - \underbrace{\frac{v_{k-1} \bullet w_k}{\|w'_{k-1}\|^2} \cdot (w_k + \mu_{k,k-1} w_{k-1})}_{=0} + \mu_{k,k-1} \bullet \frac{v_{k-1} \bullet w_{k-1}}{\|w'_{k-1}\|^2} \cdot w_k - \mu_{k,k-1}^2 \cdot \frac{v_{k-1} \bullet w_{k-1}}{\|w'_{k-1}\|^2} \cdot w_{k-1}^2 \\
&= \left(1 - \mu_{k,k-1}^2 \cdot \frac{\|w_{k-1}\|^2}{\|w'_{k-1}\|^2} \right) \cdot w_{k-1} - \mu_{k,k-1} \cdot \frac{\|w_{k-1}\|^2}{\|w'_{k-1}\|^2} \cdot w_k.
\end{aligned}$$

We have already proved that $\|w'_{k-1}\|^2 - \mu^2_{k,k-1}\|w_{k-1}\|^2 = \|w_k\|^2$. Divide both sides bu $\|w'_{k-1}\|^2$ and substitute the resulting expression into the above parenthesis to get the desired formula fro w'_k. The squared norm can be now obtained as follows

$$\begin{aligned}\|w'_k\|^2 &= \frac{\|w_k\|^4}{\|w'_{k-1}\|^4}\cdot\|w_{k-1}\|^2 + \mu^2_{k,k-1}\cdot\frac{\|w_{k-1}\|^4}{\|w'_{k-1}\|^4}\cdot\|w_k\|^2\\ &= \frac{\|w_{k-1}\|^2\cdot\|w_k\|^2}{\|w'_{k-1}\|^2}\cdot\Big(\|w_k\|^2 + \mu^2_{k,k-1}\|w_{k-1}\|\Big)\\ &= \frac{\|w_{k-1}\|^2\cdot\|w_k\|^2}{\|w'_{k-1}\|^2}\cdot\|w'_{k-1}\|^2 = \frac{\|w_{k-1}\|^2\cdot\|w_k\|^2}{\|w'_{k-1}\|^2}.\end{aligned}$$

Finally, we prove the last assertion. We need to check only the formulas for $\mu'_{i,k-1}$ and $\mu'_{i,k}$, as the other ones are obvious. We have

$$\begin{aligned}\mu'_{i,k-1} &= \frac{v'_i\bullet w'_{k-1}}{\|w'_{k-1}\|^2} = \frac{v_i\bullet w_k}{\|w'_{k-1}\|^2} + \mu_{k,k-1}\cdot\frac{v_i\bullet w_{k-1}}{\|w'_{k-1}\|^2}\\ &= \mu_{i,k}\cdot\frac{\|w_k\|^2}{\|w'_{k-1}\|^2} + \mu_{k,k-1}\mu_{i,k-1}\cdot\frac{\|w_{k-1}\|^2}{\|w'_{k-1}\|^2}\\ = \mu_{i,k}\cdot\frac{\|w'_{k-1}\|^2 - \mu^2_{k,k-1}\|w_{k-1}\|^2}{\|w'_{k-1}\|^2} &+ \mu_{i,k-1}\mu'_{k,k-1} = \mu_{i,k} - \mu_{i,k}\mu_{k,k-1}\mu'_{k,k-1} + \mu_{i,k-1}\mu'_{k,k-1}.\end{aligned}$$

In a similar way we show the last formula:

$$\begin{aligned}\mu'_{i,k} &= \frac{v'_i\bullet w'_k}{\|w'_k\|^2}\\ &= \frac{v_i\bullet\left(\frac{\|w_k\|^2}{\|w'_{k-1}\|^2}\cdot w_{k-1} - \mu_{k,k-1}\cdot\frac{\|w_{k-1}\|^2}{\|w'_{k-1}\|^2}\cdot w_k\right)}{\|w'_k\|^2}\\ &= \mu_{i,k-1}\cdot\frac{\|w_{k-1}\|^2\cdot\|w_k\|^2}{\|w'_{k-1}\|^2\cdot\|w'_k\|^2} - \mu_{i,k}\mu_{k,k-1}\cdot\frac{\|w_{k-1}\|^2\cdot\|w_k\|^2}{\|w'_{k-1}\|^2\cdot\|w'_k\|^2}.\end{aligned}$$

Now, the already proved statement 3 asserts that both the above fractions evaluate to 1. This concludes the proof. □

Proof of correctness of Algorithm 3.7.8. Let us first notice that, once the Gram–Schmidt basis $\mathscr{B}^*$ is computed in the initialization phase of the algorithm, the orthogonal vectors $w_1,\ldots,w_m$, their (cached) squared norms $\|w_1\|^2,\ldots,\|w_m\|^2$ and the Gram–Schmidt coefficients μ_{ij}, $1\le j<i\le m$ are correct throughout the execution of the algorithm. This is ensured by Lemmas 3.7.9–3.7.10 (and is trivially true if the Gram-Schmidt basis is naïvely recomputed after each alternation of the lattice basis).

Now, we prove that upon the completion of the algorithm, condition 1 of Definition 3.7.5 is satisfied. Suppose, there are indices $i < k$ such that $|\mu_{ki}| > 1/2$. Assume that, for a fixed k, the index i is the highest index for which this happens. Thus, step 2a of the algorithm replaces v_k by v'_k defined as before. For this new vector, we have

$$\mu'_{ki} = \frac{v'_k \bullet w_i}{\|w_i\|^2} = \frac{v_k \bullet w_i}{\|w_i\|^2} - [\mu_{ki}]\frac{v_i \bullet w_i}{\|w_i\|^2} - \sum_{j=1}^{i-1}[\mu_{kj}]\frac{v_j \bullet w_i}{\|w_i\|^2}$$

Now, Theorem 3.7.3 asserts that $v_j \bullet w_i = 0$ for $j < i$ and $v_i \bullet w_i = w_i \bullet w_i$. Therefore the previous formula simplifies to

$$\mu'_{ki} = \mu_{ki} - [\mu_{ki}] \cdot 1 - 0 \leq \frac{1}{2}.$$

Thus condition 1 holds.

It is completely clear that if only the algorithm ever terminates, then the resulting basis satisfies condition 2 of Definition 3.7.5. We need only to show that the algorithm indeed terminates. To this end we should bound the number of iterations of the main loop of the algorithm. This number is m plus the number of swaps in step 2(b)ii. Define

$$D := \|w_1\|^{2m} \cdot \|w_2\|^{2m-2} \cdots \|w_m\|^2.$$

The Gram-Schmidt basis $\mathscr{B}^*$ changes only in step 2(b)ii of the algorithm. Hence this is the only place where D can possibly change. Suppose that the algorithm swaps vectors v_{k-1} and v_k, this means that

$$\frac{3}{4} \cdot \|w_{k-1}\|^2 > \|w_k + \mu_{k,k-1} w_{k-1}\|^2. \tag{3.10}$$

As we did earlier, denote the modified basis by apostrophe:

$$v'_j := \begin{cases} v_j & \text{if } j \notin \{k-1, k\}, \\ v_k & \text{if } j = k-1, \\ v_{k-1} & \text{if } k = k \end{cases}$$

and let $\mathscr{B}'^* = \{w'_1, \ldots, w'_m\}$ be the associated Gram–Schmidt basis. We claim that $D' < 3/4 \cdot D$, where

$$D' := \|w'_1\|^{2m} \cdot \|w'_2\|^{2m-2} \cdots \|w'_m\|^2.$$

Using Lemma 3.7.10 together with Eq. (3.10) we write

$$\begin{aligned} \|w'_{k-1}\|^{2m-2k+2} \cdot \|w'_k\|^{2m-2k} &= \|w'_{k-1}\|^{2m-2k+2} \cdot \frac{\|w_{k-1}\|^{2m-2k} \cdot \|w_k\|^{2m-2k}}{\|w'_{k-1}\|^{2m-2k}} \\ &= \|w'_{k-1}\|^2 \cdot \|w_{k-1}\|^{2m-2k} \cdot \|w_k\|^{2m-2k} \\ &= \|w_k + \mu_{k,k-1} w_{k-1}\|^2 \cdot \|w_{k-1}\|^{2m-2k} \cdot \|w_k\|^{2m-2k} \\ &< \frac{3}{4} \cdot \|w_{k-1}\|^{2m-2k+2} \cdot \|w_k\|^{2m-2k}. \end{aligned}$$

This proves our claim. It follows that each time two vectors are swapped, D is reduced by a factor of at least $3/4$. Now, D is an integer by and so the number of swaps is bounded missing □

Remark 35. It should be noted that, once all the squared norms $\|w_1\|^2, \ldots, \|w_m\|^2$ and all the coefficients μ_{ij} for $j < i$ are computed, there is no need to store the orthogonal vectors $w_1, \ldots, w_m$ themselves. Lemmas 3.7.9–3.7.10 let us update the squared norms and Gram–Schmidt coefficients without referring to the actual vectors.

Example

Schnorr algorithm

3.8 Special matrices

Vandermonde matrix

The following special type of matrices, named after A.-T. Vandermonde[7], has many application. We will encounter some of them in following chapters.

Definition 3.8.1. Let $a_1, \ldots, a_n$ be any elements (of some fixed ring). A square matrix

$$V_{a_1,\ldots,a_n} := \begin{pmatrix} 1 & a_1 & a_1^2 & \cdots & a_1^{n-1} \\ 1 & a_2 & a_2^2 & \cdots & a_2^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & a_n & a_n^2 & \cdots & a_n^{n-1} \end{pmatrix}$$

is called a *Vandermonde matrix*.

First, let us derive a recursive formula for the determinant of a Vandermonde matrix.

Lemma 3.8.2. *For any $a_1, \ldots, a_n$ one has*

$$\det V_{a_1,\ldots,a_n} = \det V_{a_1,\ldots,a_{n-1}} \cdot \prod_{i=1}^{n-1} (a_n - a_i).$$

Proof. Take a Vandermonde matrix $V_{a_1,\ldots,a_n}$. From every column (except the first one) subtract a_n times the column immediately preceding it. This operation does not change

[7] Alexandre-Théophile Vandermonde (born 28 February 1735, died 1 January 1796) was a French violinist, mathematician and chemist.

the determinant, hence

$$\det V_{a_1,\dots,a_n} = \det \begin{pmatrix} 1 & (a_1-a_n) & a_1\cdot(a_1-a_n) & \cdots & a_1^{n-2}\cdot(a_1-a_n) \\ 1 & (a_2-a_n) & a_2\cdot(a_2-a_n) & \cdots & a_2^{n-2}\cdot(a_2-a_n) \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & (a_{n-1}-a_n) & a_{n-1}\cdot(a_{n-1}-a_n) & \cdots & a_{n-1}^{n-2}\cdot(a_{n-1}-a_n) \\ 1 & (a_n-a_n) & a_n\cdot(a_n-a_n) & \cdots & a_n^{n-2}\cdot(a_n-a_n) \end{pmatrix}$$

$$= \det \begin{pmatrix} 1 & (a_1-a_n) & a_1\cdot(a_1-a_n) & \cdots & a_1^{n-2}\cdot(a_1-a_n) \\ 1 & (a_2-a_n) & a_2\cdot(a_2-a_n) & \cdots & a_2^{n-2}\cdot(a_2-a_n) \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & (a_{n-1}-a_n) & a_{n-1}\cdot(a_{n-1}-a_n) & \cdots & a_{n-1}^{n-2}\cdot(a_{n-1}-a_n) \\ 1 & 0 & 0 & \cdots & 0 \end{pmatrix}$$

Compute the determinant using Laplace expansion (Theorem 3.4.2) with respect to the last row:

$$= (-1)^{n+1}\cdot \det \begin{pmatrix} (a_1-a_n) & a_1\cdot(a_1-a_n) & \cdots & a_1^{n-2}\cdot(a_1-a_n) \\ (a_2-a_n) & a_2\cdot(a_2-a_n) & \cdots & a_2^{n-2}\cdot(a_2-a_n) \\ \vdots & \vdots & & \vdots \\ (a_{n-1}-a_n) & a_{n-1}\cdot(a_{n-1}-a_n) & \cdots & a_{n-1}^{n-2}\cdot(a_{n-1}-a_n) \end{pmatrix}$$

$$= (-1)^{n+1}\cdot(a_1-a_n)\cdots(a_{n-1}-a_n)\cdot \det \begin{pmatrix} 1 & a_1 & \cdots & a_1^{n-2} \\ 1 & a_2 & \cdots & a_2^{n-2} \\ \vdots & \vdots & & \vdots \\ 1 & a_{n-1} & \cdots & a_{n-1}^{n-2} \end{pmatrix}$$

$$= (-1)^2 \cdot \prod_{i=1}^{n-1}(a_n-a_i)\cdot \det V_{a_1,\dots,a_{n-1}}. \qquad \square$$

It is now easy to resolve the recursion and write an explicit formula for the derivative of a Vandermonde matrix.

Theorem 3.8.3. *For any $a_1,\dots,a_n$ the determinant of the Vandermonde matrix $V_{a_1,\dots,a_n}$ equals*

$$\det V_{a_1,\dots,a_n} = \prod_{1\le i<j\le n}(a_j-a_i).$$

Proof. The formula is vacuously true for $n=1$ since then the determinant of $V_{a_1} = \begin{pmatrix}1\end{pmatrix}$ is 1 and so equals a product over an empty set of indices. For the sake of clarity, we will check the formula also for $n=2$. In this case

$$\det V_{a_1,a_2} = \det \begin{pmatrix} 1 & a_1 \\ 1 & a_2 \end{pmatrix} = a_2 - a_1$$

and the formula holds.

Assume that we have proved the thesis for a Vandermonde matrix $V_{a_1,\ldots,a_{n-1}}$ of size $(n-1)\times(n-1)$. By the previous lemma:

$$\begin{aligned}\det V_{a_1,\ldots,a_n} &= \det V_{a_1,\ldots,a_{n-1}} \\ cdotprod_{k=1}^{n-1}(a_n-a_k) & \\ &= \prod_{1\le i<j\le n-1}(a_j-a_i)\cdot\prod_{k=1}^{n-1}(a_n-a_k) \\ &= \prod_{1\le i<j\le n}(a_j-a_i).\end{aligned}$$

□

An immediate but extremely important consequence of the previous theorem is:

Corollary 3.8.4. *A Vandermonde matrix $V_{a_1,\ldots,a_n}$, with coefficients in some field, is invertible if and only if elements $a_1,\ldots,a_n$ are distinct.*

In a greater generality we have:

Proposition 3.8.5. *The rank of $V_{a_1,\ldots,a_n}$ equals the number of distinct elements among $a_1,\ldots,a_n$.*

Proof. Let $a_{i_1},\ldots,a_{i_k}$ be all the distinct elements among $a_1,\ldots,a_n$. The previous corollary asserts that $V_{a_{i_1},\ldots,a_{i_k}}$ is invertible, hence

$$k=\operatorname{rank}\bigl(V_{a_{i_1},\ldots,a_{i_k}}\bigr)\le\operatorname{rank}\bigl(V_{a_1,\ldots,a_n}\bigr).$$

On the other hand, if we adjoin any of the omitted elements, say a_j, then a_j is equal to one of the elements $a_{i_1},\ldots,a_{i_k}$. Consequently the matrix $V_{a_{i_1},\ldots,a_{i_k};a_j}$ contains two equal rows and so its determinant vanishes. Consequently $\operatorname{rank}V_{a_1,\ldots,a_n}<k+1$. These two inequalities together prove the assertion. □

Companion matrix

We shall now introduce another special kind of matrices, very useful for polynomial computations.

Definition 3.8.6. If $f\in K[x]$ is a nonzero polynomial (in any representation), then a square matrix C_f is called a *companion matrix* of f, if for some nonzero constant c

$$f=c\cdot\det\bigl(C_f-x\cdot I\bigr), \tag{3.11}$$

where I denotes the identity matrix (of the same size as C_f).

The following very important observation is immediate.

Observation 3.8.7. *Let C_f be a companion matrix of a polynomial f. The eigenvalues of C_f are precisely the roots of f, with all the multiplicities preserved.*

An actual form of a companion matrix (of a given polynomial f) depends on the chosen representation of polynomials. It is particularly simple for polynomials in the power-basis form.

Proposition 3.8.8. *If $f + f_0 + f_1 x + \cdots + f_d x^d$, then*

$$C_f := \begin{pmatrix} 0 & 0 & \cdots & 0 & -f_0/f_d \\ 1 & 0 & \cdots & 0 & -f_1/f_d \\ 0 & 1 & \cdots & 0 & -f_2/f_d \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & 1 & -f_{d-1}/f_d \end{pmatrix}$$

is a companion matrix of f.

Proof. Przerobic na det(xI - Cf) - tak jest prosciej

Compute the determinant of $(C_f - xI)$ using Laplace expansion with respect to the last column:

$$\det\big(C_f - x \cdot I\big) = \det \begin{pmatrix} -x & 0 & \cdots & 0 & 0 & -f_0/f_d \\ 1 & -x & \cdots & 0 & 0 & -f_1/f_d \\ 0 & 1 & \cdots & 0 & 0 & -f_2/f_d \\ \vdots & & \ddots & & \vdots & \\ 0 & 0 & \cdots & 1 & -x & -f_{d-2}/f_d \\ 0 & 0 & \cdots & 0 & 1 & -f_{d-1}/f_d - x \end{pmatrix}$$

$$= (-1)^{d+2} \cdot \frac{f_0}{f_d} \cdot 1 + (-1)^{d+3} \cdot \frac{f_1}{f_d} \cdot (-x) + (-1)^{d+4} \frac{f_2}{f_d} \cdot (-x)^2 + \cdots$$

$$\cdots + (-1)^{2d+1} \cdot \frac{f_{d-1}}{f_d} \cdot (-x)^{d-1} + (-1)^{2d+1} \cdot x \cdot (-x)^{d-1}$$

$$= \frac{(-1)^d}{f_d} \cdot \big(f_0 + f_1 x + f_2 x^2 + \cdots + f_{d-1} x^{d-1} + f_d x^d\big). \qquad \square$$

As the following exercise shows, a companion matrix is not unique.

Exercise 8. The following forms of companion matrices of a polynomial $f_0 + f_1 x + \cdots + f_d x^d$ can be found in various textbooks:

$$\begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & & 1 \\ -f_0/f_d & -f_1/f_d & -f_2/f_d & \cdots & -f_{d-1}/f_d \end{pmatrix}, \quad \begin{pmatrix} -f_{d-1}/f_d & \cdots & -f_2/f_d & -f_1/f_d & -f_0/f_d \\ 1 & & 0 & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & & 1 & 0 & 0 \\ 0 & \cdots & 0 & 1 & 0 \end{pmatrix}, \quad \begin{pmatrix} -f_{d-1}/f_d & 1 & \cdots \\ \vdots & & \ddots \\ -f_2/f_d & 0 & \cdots \\ -f_1/f_d & 0 & \cdots \\ -f_0/f_d & 0 & \cdots \end{pmatrix}$$

Prove that they all satisfy condition Eq. (3.11).

We already know that the roots of f are precisely the eigenvalues of its companion matrix. If the roots are all distinct (the polynomial f is then said to be *square-free*—see Section 5.3 of Chapter 5), then it is easy to find a matrix diagonalizing C_f.

Proposition 3.8.9. *Let f be a nonzero polynomial and $\xi_1, \ldots, \xi_m$ be its roots. Further let C_f be the companion matrix of f in a form as in Proposition 3.8.8. Finally let $V = V_{\xi_1,\ldots,\xi_m}$ be the Vandermonde matrix. If the multiplicity of every root of f is 1, then $V \cdot C_f \cdot V^{-1}$ is a diagonal matrix* $\operatorname{diag}(\xi_1, \ldots, \xi_m)$.

Proof. It is well known that if all the roots of a polynomials are distinct (i.e. multiplicity of each root equals 1), then the degree of the polynomial is the same as the number of roots, hence $\deg f = m$. Say $f = f_0 + f_1 x + \cdots + f_m x^m$. By definition

$$V = V_{\xi_1,\ldots,\xi_m} = \begin{pmatrix} 1 & \xi_1 & \cdots & \xi_1^{m-1} \\ 1 & \xi_2 & \cdots & \xi_2^{m-1} \\ \vdots & \vdots & & \vdots \\ 1 & \xi_m & \cdots & \xi_m^{m-1} \end{pmatrix}.$$

Compute $V \cdot C_f$. The first $m-1$ columns are easy to obtain. Multiplying i-th row of V by j-th column of C_f (with $j < m$) one gets

$$\begin{pmatrix} 1 & \xi_i & \cdots & \xi_i^{m-1} \end{pmatrix} \cdot \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix} = \xi_i^j.$$

Thus the first $m-1$ columns of the product $V \cdot C_f$ are

$$\left(\begin{array}{cccc:c} \xi_1 & \xi_1^2 & \cdots & \xi_1^{m-1} & \star \\ \xi_2 & \xi_2^2 & \cdots & \xi_2^{m-1} & \star \\ \vdots & \vdots & & \vdots & \vdots \\ \xi_m & \xi_m^2 & \cdots & \xi_m^{m-1} & \star \end{array}\right).$$

Let us find out how the last column looks like. The i-th element of the last column equals

$$\begin{aligned} \begin{pmatrix} 1 & \xi_i & \cdots & \xi_i^{m-1} \end{pmatrix} \cdot \begin{pmatrix} -f_0/f_m \\ -f_1/f_m \\ \vdots \\ -f_{m-1}/f_m \end{pmatrix} &= -\frac{f_0}{f_m} - \frac{f_1}{f_m}\xi_i - \cdots - \frac{f_{m-1}}{f_m}\xi_i^{m-1} \\ &= -\frac{1}{f_m}\big(f(\xi_i) - f_m \xi_i^m\big) = \xi_i^m \end{aligned}$$

since ξ_i is a root of f.

Now, compute a product $\operatorname{diag}(\xi_1,\ldots,\xi_m)\cdot V$:

$$\begin{pmatrix}\xi_1 & & \\ & \ddots & \\ & & \xi_m\end{pmatrix}\cdot\begin{pmatrix}1 & \xi_1 & \cdots & \xi_1^{m-1}\\ \vdots & \vdots & & \vdots\\ 1 & \xi_m & \cdots & \xi_m^{m-1}\end{pmatrix}=\begin{pmatrix}\xi_1 & \xi_1^2 & \cdots & \xi_1^m\\ \xi_2 & \xi_2^2 & \cdots & \xi_2^m\\ \vdots & \vdots & & \vdots\\ \xi_m & \xi_m^2 & \cdots & \xi_m^m\end{pmatrix}.$$

Therefore $V\cdot C_f=\operatorname{diag}(\xi_1,\ldots,\xi_m)\cdot V$ and consequently $VC_fV^{-1}=\operatorname{diag}(\xi_1,\ldots,\xi_m)$, providing that V is invertible. This is however ensured by the means of corollary 3.8.4, since we assumed that $\xi_1,\ldots,\xi_m$ are all distinct. □

Further reading

To derive $O(n^{2.807})$ matrix multiplication algorithm (Algorithm 3.1.3), we used Strasses's observation (obtained in [82]), that a product of two 2×2 matrices can be computed with only 7 scalar multiplications. It is not the only formula of this kind. A standard formula for a product of two 3×3 matrices involves 27 scalar products. In 1976 J. Laderman presented algorithm that needs only 23 scalar multiplications (see [49]). Another formulas with the same number of multiplications are obtained in [36, 21]. This result was improved by O. Makarov in [58], who presented a formula that uses only 22 scalar products assuming that the scalar-multiplication is commutative (hence it does not give rise to a divide-and-conquer algorithm for large matrices).

There exist some matrix multiplication algorithms that are asymptotically faster than Strassen's one. A rather well known is algorithm due to D. Coppersmith and S. Winograd (see [20]) with time complexity $O(n^{2.376})$. The two asymptotically fastest algorithms that the author is aware of (at the time of writing) have time complexity $O(n^{2.373})$ and are due to D.V. Zhdanovich (see [93]) and F. Le Gall (see [53])

Algorithms 3.2.8 and 3.4.7 come from [8]. In that paper these algorithms are referred to as single-step elimination algorithms and accredited to C. Jordan. The paper contains a general multi-step approach that we do not present here.

The exposition of Samuelson-Berkowitz algorithm is based mostly on [79]. The recursive formula for the determinant (see Lemma 3.4.16) has its origins in the old paper by P. Samuelson [76]. For discussion on parallel execution of this algorithm see the original article by S. Berkowitz [11]. The combinatorial interpretation and the proof of correctness of this algorithm based on methods of discrete mathematics is presented in [54, 73, 79]. The Bird's method is based on article [12].

In the literature one can find more than a dozen different proofs. A particularly elementary one was published recently in [52].

4

Reconstruction and interpolation

It was like... connecting the dots.

"Proof" (movie, 2005)

4.1 Polynomial interpolation

In this section we consider a problem of finding a polynomial (or a piecewise polynomial function) that takes prescribed values in given points. Strictly speaking the problem we are dealing with is the following one:

Problem 4.1.1. Let K be a field. Given $n+1$ distinct arguments $x_0, x_1, \ldots, x_n \in K$ and corresponding arbitrary values $y_0, y_1, \ldots, y_n \in K$, find a polynomial $f \in K[x]$ of a lowest possible degree (respectively a piecewise polynomial function) such that

$$f(x_0) = y_0,\ f(x_1) = y_1, \ldots,\ f(x_n) = y_n. \tag{4.1}$$

The elements $x_0, \ldots, x_n$; $y_0, \ldots, y_n$ will hereafter be referred as *input data*. In practice, we often work with just $K = \mathbb{R}$, but we sometimes need to extend the co-domain taking $y_0, \ldots, y_n$ from $\mathbb{R}^k$. As we shall see, once we find a solution to Problem 4.1.1, we will be able to use it in any affine space. In case of $\mathbb{R}^k$ it simply boils down to constructing appropriate polynomials (resp. piecewise polynomial functions) coordinate-wisely.

The polynomial interpolation can be viewed as a special case of Chinese remainder problem. Indeed, given distinct $x_0, \ldots, x_n$ and some $y_0, \ldots, y_n$, Eq. (4.1) can be expressed as a system of congruences

$$\begin{cases} f \equiv y_0 \pmod{x - x_0} \\ f \equiv y_1 \pmod{x - x_1} \\ \quad \vdots \\ f \equiv y_n \pmod{x - x_n}. \end{cases}$$

It follows that a polynomial f obtained by the means of Chinese remainder theorem is the sought interpolating polynomial. Nevertheless, in this section we will not relay on Chinese remainder theorem and we will develop all the interpolation formulas from scratch. This makes the section self-contained and possible to be read independently.

Lagrange interpolation formula

Theorem 4.1.2. *For a given input data $x_0, \ldots, x_n$ and $y_0, \ldots, y_n$, there exists a unique polynomial f of degree not exceeding n that satisfies Eq.* (4.1).

We prove only the uniqueness of f. The existence will be obtained by disclosing an explicit formula for f (see Proposition 4.1.3).

Proof (*uniqueness*). The uniqueness follows from an observation that a degree zero (i.e. constant) polynomial is uniquely determined by its values in just one point, degree one (i.e. linear) by its two values, degree two (quadratic) by three values and so on. Indeed, suppose there are two polynomials $f, g \in K[x]$ satisfying Eq. (4.1), both of degree $\leq n$. Take their difference $h := f - g$. Then h has $n+1$ roots, namely $x_0, x_1, \ldots, x_n$, while $\deg h \leq n$. We know that the number of roots of a non-zero polynomial cannot exceed its degree. It follows that $h = 0$ and so $f = g$. This proves the uniqueness. □

The following analytic formula for the interpolating polynomial f is named after J.-L. Lagrange[1], although some source attribute it to E. Waring[2]. It constitutes the existential part of Theorem 4.1.2.

Proposition 4.1.3 (Lagrange interpolation formula)**.** *The interpolating polynomial f is given by an explicit formula*

$$f = \sum_{j=0}^{n} \left(y_j \cdot \prod_{\substack{k=0 \\ k \neq j}}^{n} \frac{x - x_k}{x_j - x_k} \right). \tag{4.2}$$

Proof. Denote

$$\ell_j := \prod_{\substack{k=0 \\ k \neq j}}^{j} \frac{x - x_k}{x_j - x_k}.$$

It is straightforward to show that

$$\ell_j(x_k) = \begin{cases} 1, & \text{if } k = j \\ 0, & \text{if } k \neq j. \end{cases}$$

[1] Joseph-Louis Lagrange (born on 25 January 1736, died on 10 April 1813) was an Italian mathematician and astronomer.

[2] Edward Waring (born around 1736, died on 15 August 1798) was an English mathematician and a member of Royal Society of London for Improving Natural Knowledge.

It follows that $f(x_j) = 0 + \cdots + 0 + y_j \cdot 1 + 0 + \cdots + 0 = y_j$ and so f satisfies Eq. (4.1). Further, by its very definition, the degree of ℓ_j equals n and so does the degree of f. This proves Proposition 4.1.3 and consequently also Theorem 4.1.2. □

Example 47. We illustrate the method, over a finite field $\mathbb{F}_{13}$, with the following input data:

$$x_0 = 1, \quad x_1 = 3, \quad x_2 = 5, \quad x_3 = 7, \quad x_4 = 11 \qquad \text{and} \qquad y_0 = 3, \quad y_1 = 0, \quad y_2 = 1, \quad y_3 = 7, \quad y$$

Using Proposition 4.1.3, construct the interpolating polynomial:

$$f = 3 \cdot \frac{(x-3)(x-5)(x-7)(x-11)}{(1-3)(1-5)(1-7)(1-11)} + 0 \cdot \frac{(x-1)(x-5)(x-7)(x-11)}{(3-1)(3-5)(3-7)(3-11)} + 1 \cdot \frac{(x-1)(x-3)(x-7)}{(5-1)(5-3)(5-7)(}$$
$$= x^4 + x^2 + 1.$$

Remark 36. Observe that for $y_0 = \cdots = y_n = 1$, the unique solution to Problem 4.1.1 is given by a constant polynomial $f = 1$. It follows that

$$\sum_{j=0}^{n} \prod_{\substack{k=0 \\ k \neq j}}^{n} \frac{x - x_k}{x_j - x_k} \equiv 1.$$

Consequently, if A is an affine space over the field K and $P_0, \ldots, P_n \in A$, then for every $t \in K$ the point $\sum_{j=0}^{n} P_j \ell_j(t)$ is a centroid (affine combination) of $P_0, \ldots, P_n$ with weights $\ell_0(t), \ldots, \ell_n(t)$. It follows that f is a well defined function from K to A. Thus, the interpolation problem can be considered over arbitrary affine K-spaces.

Remark 37. The reader should observe that Lagrange interpolation formula is just a special case of Algorithm 2.2.3, that is sometimes referred to as Lagrange algorithm for Chinese remainder problem.

Interpolation through linear algebra

We shall now introduce three methods for constructing the interpolating polynomials, alternative to the explicit formula Eq. (4.2). Of course, all of them lead to the same polynomial due to its uniqueness. One solution comes from a realm of linear algebra. Additionally, it provides a different proof for Theorem 4.1.2.

Suppose that we are given $x_0, \ldots, x_n$ and $y_0, \ldots, y_n$ as in Problem 4.1.1. We are looking for a polynomial f such that $f(x_j) = y_j$ for every $0 \leq j \leq n$. We already know that such f exists and its degree does not exceed n. Write it as $f = f_0 + f_1 x + \cdots + f_n x^n$. Our interpolating problem can be expressed as a system of equations:

$$\begin{cases} f_0 + f_1 x_0 + f_2 x_0^2 + \cdots + f_n x_0^n = y_0 \\ f_0 + f_1 x_1 + f_2 x_1^2 + \cdots + f_n x_1^n = y_1 \\ \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ f_0 + f_1 x_n + f_2 x_n^2 + \cdots + f_n x_n^n = y_n. \end{cases}$$

Beware however, that the unknowns here are not the usual $x_0, \ldots, x_n$ (those are given), but $f_0, \ldots, f_n$. The matrix of this linear system is just the Vandermonde's matrix $V_{x_0,\ldots,x_n}$. All the x_i's are distinct by assumption. Corollary 3.8.4 asserts that the matrix is invertible, ans so the system has a unique solution by Cramer's[3] rule. This way we have obtained a new prove of Theorem 4.1.2 and a method of finding the interpolating polynomial.

Algorithm 4.1.4. *Given distinct elements $x_0, \ldots, x_n \in K$ and arbitrary $y_0, \ldots, y_n \in K$, this algorithm constructs the interpolating polynomial $f \in K[x]$.*

1. *Construct the Vandermonde's matrix $V_{x_0,\ldots,x_n}$;*
2. *solve the system of liner equations*

$$V_{x_0,\ldots,x_n} \cdot \begin{pmatrix} f_0 \\ \vdots \\ f_n \end{pmatrix} = \begin{pmatrix} y_0 \\ \vdots \\ y_n \end{pmatrix};$$

3. *return a polynomial $f = f_0 + f_1 x + \cdots + f_n x^n$.*

Unfortunately the Vandermonde's matrices are known to be notoriously ill-conditioned and so this method is not very well suited for floating point computations.

Example 48 (continued). Consider the same interpolation problem as in the previous example:

$$x_0 = 1, \quad x_1 = 3, \quad x_2 = 5, \quad x_3 = 7, \quad x_4 = 11 \qquad \text{and} \qquad y_0 = 3, \quad y_1 = 0, \quad y_2 = 1, \quad y_3 = 7, \quad y$$

where $x_0, \ldots, x_n, y_0, \ldots, y_n \in \mathbb{F}_{13}$. Build the Vandermonde's matrix and augment it with a column of y_i's:

$$\left(\begin{array}{ccccc|c} 1 & 1 & 1 & 1 & 1 & 3 \\ 1 & 3 & 9 & 1 & 3 & 0 \\ 1 & 5 & 12 & 8 & 1 & 1 \\ 1 & 7 & 10 & 5 & 9 & 7 \\ 1 & 11 & 4 & 5 & 3 & 8 \end{array}\right).$$

Using row reduction, transform the matrix into a reduced row echelon form:

$$\left(\begin{array}{ccccc|c} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{array}\right).$$

It follows that

$$f_0 = 1, \quad f_1 = 0, \quad f_2 = 1, \quad f_3 = 0, \quad f_4 = 1$$

and so $f = x^4 + x^2 + 1$.

[3] Gabriel Cramer (born 31 July 1704, died 4 January 1752) was a Swiss mathematician

Neville's method

Observe that the method described in the previous subsection returns just the coefficients of the interpolating polynomial. If we are not quite interested in the polynomial itself, but rather in its value at some fixed intermediate point a, we still have to compute f first and only then we may evaluate it at a, using Hroner's rule. This is very different from Lagrange interpolation formula, where one may simply plug a into Eq. (4.2) in place of x to directly obtain $f(a)$. The next algorithm, we discuss, is due to E.H. Neville[4] and is more in a flavor of Proposition 4.1.3, in the sense that it can produce either a polynomial (when we use an indeterminate x) or just its value (when we replace x by a fixed argument).

The main idea of Neville's algorithm is to recursively build "layers" of interpolating polynomials of increasing degrees. We start from a layer zero, consisting of constant polynomials, each interpolating a single value y_i. It is clear that the polynomials are

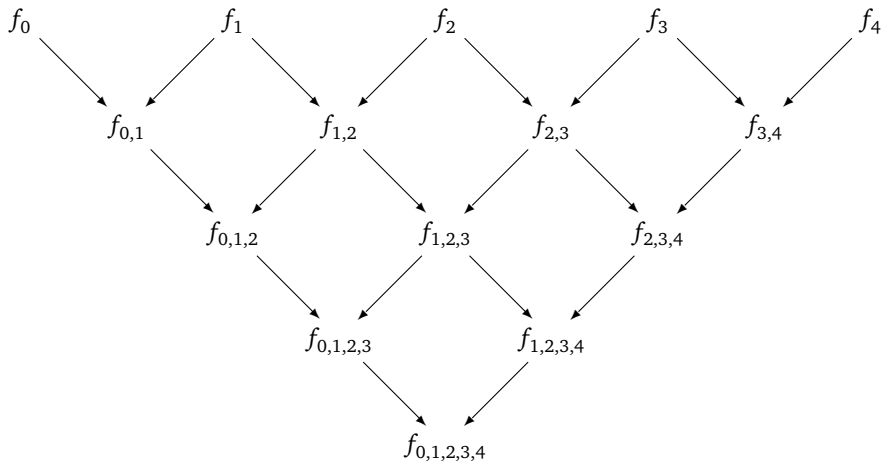
$$f_0 = y_0,\ f_1 = y_1, \ldots,\ f_n = y_n.$$

In the next layer, we construct linear polynomials $f_{0,1}, \ldots, f_{n-1,n}$, interpolating the input data at two points. This means that $f_{i,i+1}(x_i) = y_i$ and $f_{i,i+1}(x_{i+1}) = y_{i+1}$. We continue this process, so that a layer k contains polynomials $f_{i,i+1,\ldots,i+k}$ for $i \in \{0, \ldots, n-k\}$ satisfying

$$f_{i,i+1,\ldots,i+k}(x_j) = y_j, \qquad \text{for } i \le j \le i+k.$$

Finally, the last layer consists of just a single polynomial $f_{0,\ldots,n}$, interpolating all the input data, so that $f(x_i) = y_i$ for every $i \in \{0, \ldots, n\}$.

The polynomials computed in the intermediate steps of Neville's algorithm can be organized in a diagram of a form, which is not uncommon to interpolating schemes (compare it for example with de Boor's algorithm in). For $n = 4$ the diagram has a form: missing



[4] Eric Harold Neville (born 1 January 1889, died 22 August 1961) was an English mathematician.

As said, the k-th level function $f_{i,\ldots,i+k}$ has to satisfy $f_{i,\ldots,i+k}(x_j) = y_j$ for all $j \in \{i,\ldots,i+k\}$. We shall prove inductively, that a recursive formula for it is

$$f_{i,i+1,\ldots,i+k-1,i+k}(x) = \frac{1}{x_{i+k}-x_i} \cdot \det\begin{pmatrix} f_{i,i+1,\ldots,i+k-1}(x) & x_i - x \\ f_{i+1,\ldots,i+k-1,i+k}(x) & x_{i+k}-x \end{pmatrix}. \tag{4.3}$$

Indeed, for $j = i$ it follows from the inductive hypothesis that $f_{i,i+1,\ldots,i+k-1}(x_i) = y_i$, hence

$$f_{i,\ldots,i+k}(x_i) = \frac{1}{x_{i+k}-x_i}\cdot\det\begin{pmatrix} f_{i,\ldots,i+k-1}(x_i) & x_i - x_i \\ f_{i+1,\ldots,i+k}(x_i) & x_{i+k}-x_i \end{pmatrix} = \frac{1}{x_{i+k}-x_i}\cdot\det\begin{pmatrix} y_i & 0 \\ f_{i+1,\ldots,i+k}(x_i) & x_{i+k}-x_i \end{pmatrix} =$$

Analogously, for $j = i+k$ we have $f_{i+1,\ldots,i+k}(x_{i+k}) = y_{i+k}$ and so

$$\begin{aligned} f_{i,\ldots,i+k}(x_{i+k}) &= \frac{1}{x_{i+k}-x_i} \cdot \det\begin{pmatrix} f_{i,\ldots,i+k-1}(x_{i+k}) & x_i - x_{i+k} \\ f_{i+1,\ldots,i+k}(x_{i+k}) & x_{i+k}-x_{i+k} \end{pmatrix} \\ &= \frac{1}{x_{i+k}-x_i} \cdot \det\begin{pmatrix} f_{i,\ldots,i+k-1}(x_{i+k}) & x_i - x_{i+k} \\ y_{i+k} & 0 \end{pmatrix} = y_{i+k}. \end{aligned}$$

Finally, for $i < j < i+k$, both $f_{i,\ldots,i+k-1}(x_j)$ and $f_{i+1,\ldots,i+k}(x_j)$ are equal y_j. Consequently:

$$\begin{aligned} f_{i,\ldots,i+k}(x_j) &= \frac{1}{x_{i+k}-x_i} \cdot \det\begin{pmatrix} f_{i,\ldots,i+k-1}(x_j) & x_i - x_j \\ f_{i+1,\ldots,i+k}(x_j) & x_{i+k}-x_j \end{pmatrix} \\ &= \frac{1}{x_{i+k}-x_i} \cdot \det\begin{pmatrix} y_j & x_i - x_j \\ y_j & x_{i+k}-x_j \end{pmatrix} = y_j \cdot \frac{x_{i+k}-x_j-x_i+x_j}{x_{i+k}-x_i} = y_j. \end{aligned}$$

This proves our claim as well as the correctness of the following algorithm:

Algorithm 4.1.5 (Neville). *Let K be a field and x either a free variable or an element of K. For $n+1$ distinct elements $x_0,\ldots,x_n \in K$ and corresponding values $y_0,\ldots,y_n$ denote the unique interpolating polynomial of degree $\leq n$ by f. This algorithm computes $f(x)$.*

1. *Set $f_j := y_j$ for all $j \in \{0,\ldots,n\}$;*
2. *for every $k \in \{1,\ldots,n\}$ do the following:*
   a) *for every $i \in \{0,\ldots,n-k\}$ construct $f_{i,\ldots,i+k}(x)$ using Eq. (4.3);*
3. *return $f_{0,\ldots n}(x)$.*

Example 49 (continued). Consider once again the field $\mathbb{F}_{13}$ and the input data from Example 47:

$$x_0 = 1, \quad x_1 = 3, \quad x_2 = 5, \quad x_3 = 7, \quad x_4 = 11 \qquad \text{and} \qquad y_0 = 3, \quad y_1 = 0, \quad y_2 = 1, \quad y_3 = 7, \quad y$$

The first iteration of Neville's algorithm produces:

$$f_{(0,1)} = 5x + 11, \quad f_{(1,2)} = 7x + 5, \quad f_{(2,3)} = 3x + 12, \quad f_{(3,4)} = 10x + 2.$$

Next, we compute the second layer:

$$f_{(0,1,2)} = 7x^2 + 3x + 6, \quad f_{(1,2,3)} = 12x^2 + 2x + 3, \quad f_{(2,3,4)} = 12x^2 + 2x + 3,$$

then the third:

$$f_{(0,1,2,3)} = 3x^3 + 6x^2 + 7x, \quad f_{(1,2,3,4)} = 12x^2 + 2x + 3$$

and finally we have

$$f_{0,\ldots,4} = x^4 + x^2 + 1.$$

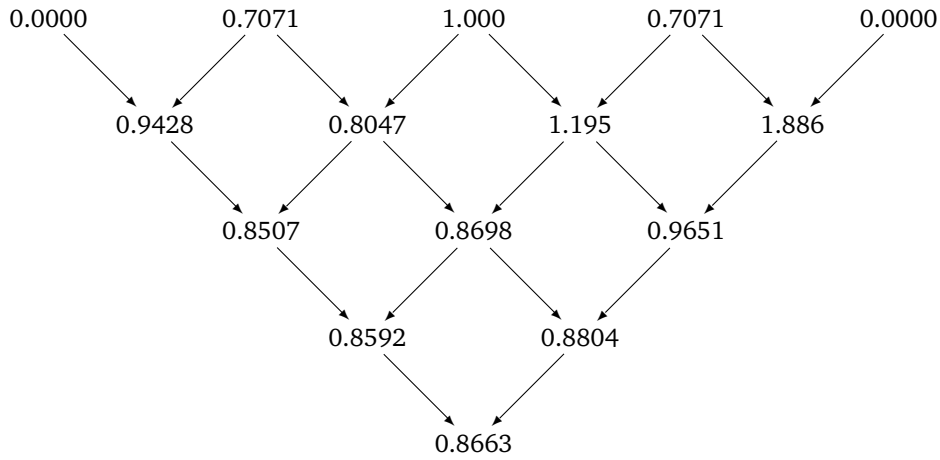
Example 50. As we have already mentioned, Algorithm 4.1.5 can be used not only for computing the interpolating polynomial, but also for directly obtaining its value at a specified point. Let us numerically calculate a value at $a := 1.047 \approx \pi/3$ of a polynomial approximation of sinus, where sinus is sampled at regular intervals of length $\pi/4$. We perform the computations in a floating point numbers with 16 bits of precision, hence the machine epsilon in our case equals 2^{-15}. Starting from the input data:

$$x_0 = 0.0000, \quad x_1 = 0.7854, \quad x_2 = 1.571, \quad x_3 = 2.356, \quad x_4 = 3.142$$

and

$$y_0 = 0.0000, \quad y_1 = 0.7071, \quad y_2 = 1.000, \quad y_3 = 0.7071, \quad y_4 = 0.0000,$$

the successive steps of Neville's algorithm are:



and so the final answer is 0.8663 with a relative error:

$$\frac{\left|\frac{1}{2}\sqrt{3} - 0.8663\right|}{\left|\frac{1}{2}\sqrt{3}\right|} \approx 9.330 \cdot \varepsilon.$$

Remark 38. An incremental nature of Neville's algorithm is its huge advantage. Suppose we have constructed an interpolation polynomial from some input data $x_0, \ldots, x_m$; $y_0, \ldots, y_m$ and after that we appended the data with some extra points $x_{m+1}, \ldots, x_n$; $y_{m+1}, \ldots, y_n$. In Neville's algorithm we can reuse the previous computations (providing that all the intermediate polynomials/values $f_{i,\ldots,i+k}(x)$ are available) and only "extend" the result to encompass the new points. This is not possible with the two methods described earlier, where we have to start everything from scratch.

Newton divided differences method

The next method we discuss is due to I. Newton and, like Neville's method, it is incremental in nature, hence new points can be freely added to the input data when needed. Recall that in Algorithm 4.1.4 we constructed the interpolation polynomial f by solving a system of linear equations. Observe that the coefficients of f (i.e. the unknowns in the system of equations) are nothing else but the coordinates of this polynomial with respect to a basis $\{1, x, \ldots, x^n\}$ of a linear space $K[x]_n = \{f \in K[x] \,|\, \deg f \leq n\}$. This is the so called *power basis* of $K[x]_n$. But the power basis is not the only basis of this linear space, nor its is an optimal choice of a basis in our case.

Let $x_0, \ldots, x_n \in K$ be some distinct elements. For $k \in \{0, \ldots, n\}$ denote

$$\varpi_k := (x - x_0) \cdots (x - x_{k-1}) \in K[x_k].$$

By convention $\varpi_0 := 1$ as a product over an empty set of indices.

Lemma 4.1.6. *For any fixed integer $n \in \mathbb{N}$ and distinct elements $x_0, \ldots, x_n \in K$, the set*

$$\mathscr{N}_n = \{\varpi_0, \ldots, \varpi_n\}$$

is a basis of $K[x]_n$.

Proof. Proceed by induction on n. It is obvious that $\{\varpi_0\} = \{1\}$ is a basis of $K[x]_0 = K$. Assume that the assertion holds for $k-1$ and so $\{\varpi_0, \ldots, \varpi_{k-1}\}$ is a basis of $K[x]_{k-1}$. Since the number of elements of $\mathscr{N}_n$ is $k+1$, and so equals the dimension $\dim K[x]_k$, it suffices to show that the set $\mathscr{N}_k$ is linearly independent. Suppose a contrario that it is not. Then there are coefficients $c_0, \ldots, c_k \in K$ not all equal zero and such that

$$c_0\varpi_0 + \cdots + c_{k-1}\varpi_{k-1} + c_k\varpi_k = 0.$$

The first k terms are linearly independent by the inductive hypothesis, hence it is c_k that must be nonzero. But then

$$\varpi_k = \frac{c_0}{c_k}\varpi_0 + \cdots + \frac{c_{k-1}}{c_k}\varpi_{k-1} \in K[x]_{k-1}.$$

This is clearly impossible, since $\deg \varpi_k = k$. This contradiction follows from our supposition that $\mathscr{N}_k$ was linearly dependent. Thus, it is linearly independent and consequently forms a basis of $K[x]_k$. □

Let us look again at our interpolation problem from the point of view of linear algebra, but this time use the new basis. We are searching for a polynomial f interpolating a given input data $x_0, \ldots, x_n$ and $y_0, \ldots, y_n$. Let

$$f = c_0\varpi_0 + c_1\varpi_1 + \cdots + c_n\varpi_n,$$

where $c_0, \ldots, c_n \in K$ are the coordinates of f with respect to $\mathscr{N}_n$. The interpolation problem may be expressed as a system of linear equations

$$\begin{cases} c_0 \cdot \varpi_0(x_0) + c_1 \cdot \varpi_1(x_0) + \cdots + c_n \cdot \varpi_n(x_0) = y_0 \\ c_0 \cdot \varpi_0(x_1) + c_1 \cdot \varpi_1(x_1) + \cdots + c_n \cdot \varpi_n(x_1) = y_1 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ c_0 \cdot \varpi_0(x_n) + c_1 \cdot \varpi_1(x_n) + \cdots + c_n \cdot \varpi_n(x_n) = y_n. \end{cases}$$

Observe that, directly by definition, ϖ_k vanishes on x_j for all $j < k$. Therefore the system we consider is in fact triangular and has a form

$$\begin{cases} c_0 = y_0 \\ c_0 + c_1 \cdot \varpi_1(x_1) = y_1 \\ c_0 + c_1 \cdot \varpi_1(x_2) + c_1 \cdot \varpi_1(x_2) = y_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ c_0 + c_1 \cdot \varpi_1(x_n) + \cdots + c_n \cdot \varpi_n(x_n) = y_n. \end{cases}$$

Solve it starting from $c_0 = y_0$. Next we have

$$\begin{aligned} c_1 &= \frac{y_1 - c_0}{x_1 - x_0} = \frac{y_1 - y_0}{x_1 - x_0} \\ c_2 &= \frac{\frac{y_2 - c_0}{x_2 - x_0} - c_1}{x_2 - x_1} = \frac{\frac{y_2 - y_0}{x_2 - x_0} - \frac{y_1 - y_0}{1_2 - x_0}}{x_2 - x_1} \\ &\vdots \end{aligned} \tag{4.4}$$

The above formulas can be compactly expressed using notion of *iterated divided differences*. Let us denote

$$y[i,j] := \frac{y_j - y_i}{x_j - x_i}.$$

Composing the operator with itself k-times define

$$y[i_1, \ldots, i_k] := \frac{y[i_2, \ldots, i_k] - y[i_1, \ldots, i_{k-1}]}{x_{i_k} - x_{i_1}}.$$

For the sake of consistency we let also $y[i] := y_i$. This way we can rewrite Eq. (4.4) in a form

$$c_j = y[0, \ldots, j] \qquad \text{for } j \in \{0, \ldots, n\}.$$

It follows from the above discussion that

Corollary 4.1.7 (Newton interpolation formula)**.** *The* (*unique*) *polynomial f satisfying Eq.* (4.1) *can be written as*

$$f = y[0]\cdot\varpi_0 + y[0,1]\cdot\varpi_1 + \cdots + y[0,1,\ldots,n]\cdot\varpi_n.$$

The above formula is sometimes—not quite rigorously—referred as *Newton interpolation polynomial*. For a given input data there is only one interpolation polynomial, hence "Newton interpolation polynomial" is just the same as "Lagrange interpolation polynomial" and can be obtained by any of the means described in this section.

The resulting algorithm is very similar to Neville's one:

Algorithm 4.1.8. *Let K be a field and x be either a free variable or an element of K. Given distinct elements $x_0,\ldots,x_n \in K$ and arbitrary $y_0,\ldots,y_n$ßK, this algorithm computes $f(x)$, where f is the unique polynomial of degree $\leq n$, satisfying $f(x_i) = y_i$ for $i \in \{0,\ldots,n\}$.*

1. *For $i \in \{0,\ldots,n\}$ denote $y[i] := y_i$;*
2. *set $\varpi_0 := 1$;*
3. *for $k \in \{1,\ldots,n\}$ do the following:*
   a) *for every $i \in \{0,\ldots,n-k\}$ construct*
   $$y[i,\ldots,i+k] := \frac{y[i+1,\ldots,i+k] - y[i,\ldots,i+k-1]}{x_{i+k} - x_i};$$
   b) *set $\varpi_k := \varpi_{k-1}\cdot(x - x_{k-1})$;*
4. *return the sum*
   $$\sum_{k=0}^{n} y[0,\ldots,k]\cdot\varpi_k.$$

Remark 39. As with Neville's method, if one adds new data points to the data set, one can build a new interpolation polynomial reusing old intermediate steps.

Example 51. We will one more time consider the same input data from the field $\mathbb{F}_{13}$:

$$x_0 = 1, \quad x_1 = 3, \quad x_2 = 5, \quad x_3 = 7, \quad x_4 = 11 \qquad \text{and} \qquad y_0 = 3, \quad y_1 = 0, \quad y_2 = 1, \quad y_3 = 7, \quad y$$

The polynomials $\varpi_0,\ldots,\varpi_4$ are computed as follows:

$$\begin{aligned}
\varpi_0 &= 1\\
\varpi_1 &= x - 1 = x + 12\\
\varpi_2 &= (x-1)(x-3) = x^2 + 9x + 3\\
\varpi_3 &= (x-1)(x-3)(x-5) = x^3 + 4x^2 + 10x + 11\\
\varpi_4 &= (x-1)(x-3)(x-5)(x-7) = x^4 + 10x^3 + 8x^2 + 6x + 1
\end{aligned}$$

The divided differences of the first order equal respectively

$$y[0,1] = \frac{0-3}{3-1} = 5 \qquad y[1,2] = \frac{1-0}{5-3} = 7$$
$$y[2,3] = \frac{7-1}{7-5} = 3 \qquad y[3,4] = \frac{8-7}{11-7} = 10.$$

Next we compute the divided differences of the second order

$$y[0,1,2] = \frac{7-5}{5-1} = 7, \qquad y[1,2,3] = \frac{3-7}{7-3} = 12, \qquad y[2,3,4] = \frac{10-3}{11-5} = 12,$$

then of the third order

$$y[0,1,2,3] = \frac{12-7}{7-1} = 3, \qquad y[1,2,3,4] = \frac{12-12}{11-3} = 0$$

and finally of the fourth order:

$$y[0,1,2,3,4] = \frac{0-3}{11-1} = 1.$$

Consequently, the sought interpolating polynomial can be obtained using Corollary 4.1.7:

$$\begin{aligned} f &= y[0]\cdot\varpi_0 + y[0,1]\cdot\varpi_1 + y[0,1,2]\cdot\varpi_2 + y[0,1,2,3]\cdot\varpi_3 + y[0,1,2,3,4]\cdot\varpi_4 \\ &= 3\cdot 1 + 5\cdot(x+12) + 7\cdot(x^2+9x+3) + 3\cdot(x^3+4x^2+10x+11) + 1\cdot(x^4+10x^3+8x^2+6x+1 \\ &= x^4 + x^2 + 1. \end{aligned}$$

Remark 40. Th reader should notice that Newton's method compromises features from all three methods discussed earlier. We have already noticed that it has an incremental characteristic similar to Neville's algorithm. Moreover, the iterated divided differences, computed during algorithm execution, can be organized in form of a diagram resembling the one we encountered when discussing Neville's algorithm. On the other hand, the derivation of Newton's interpolation formula follows from the same principles as the derivation of Algorithm 4.1.4. Last but not least, it is easy to notice that the numerators in Lagrange interpolation formula are of a form $\varpi_{k+1}/(x-x_k)$.

Remark 41. Suppose that we are given a real-valued function φ and we sample it at some points $x_0, x_1, \ldots$ setting

$$y_i := \varphi(x_i)$$

(see Section 4.2 for a discussion on how to smartly select abscissæ x_i). The divided differences of the first order can be treated as approximations of a derivative of φ. In fact, this is precisely the way one defines a derivative:

$$\varphi'(x_0) = \lim_{x_1 \to x_0} \frac{\varphi(x_1) - \varphi(x_0)}{x_1 - x_0} = \lim_{x_1 \to x_0} y[0,1].$$

Likewise, iterated divided differences of order k are discrete analogues of k-th order derivatives. If $x_0 < x_1 < \cdots < x_k$ and $x_k \to x_0$, then $y[0,\ldots,k]$ tends to $\varphi^{(k)}(x_0)$. Thus, the Newton's interpolation formula

$$f = y[0] + y[0,1]\cdot(x-x_0) + \cdots + y[0,1,\ldots,n]\cdot(x-x_0)\cdot(x-x_1)\cdots(x-x_{n-1})$$

can be treated as a discrete analogue of a truncated Taylor expansion series of φ at x_0:

$$\varphi \approx \varphi(x_0) + \varphi'(x_0)(x-x_0) + \frac{\varphi''(x_0)}{2}(x-x_0)^2 + \cdots + \frac{\varphi^{(n)}(x_0)}{n!}(x-x_0)^n.$$

Hermite's interpolation

TO DO

Applications of polynomial interpolation

The polynomial interpolation has numerous applications in both numerical and symbolic computations. In numerical methods, we often use interpolation either to model some sampled process by a polynomial function or to substitute a polynomial interpolation (which may be better behaved) for a given function. We will use this approach for instance in Chapter 7 to develop quadrature formulæ for numerical integration.

In the realm of symbolic computations, we often use interpolation in polynomial algorithms, when it is easier to find values of the sought polynomial than explicitly build the result. Let us illustrate this approach by presenting an algorithm for composing polynomials, alternative to Horner's rule (cf. Algorithm).

missing

Algorithm 4.1.9. *Let f, g be two polynomials with coefficients in some field K. Assume that the number of elements of K is greater or equal to the product of the degrees of f and g. One computes the polynomial composition $h := f \circ g$ as follows:*

1. *Denote by n the product $\deg(f)\cdot\deg(g)$;*
2. *find n distinct points $x_0,\ldots,x_{n-1}$ of K;*
3. *for every $0 \le i < n$ let*

$$z_i := g(x_i) \quad \text{and} \quad y_i := f(z_i);$$

4. *using any of the algorithms described earlier in this chapter, find an interpolating polynomial h such that*

$$h(x_0) = y_0,\ h(x_1) = y_1, \ldots,\ h(x_{n-1}) = y_{n-1};$$

5. *return h.*

4.2 Runge's phenomenon and Chebyshev nodes

In the previous section we learned how to build a polynomial that interpolates a given set of data. Now suppose that the interpolated data is obtained by sampling some real-valued function $\varphi : \mathbb{R} \to \mathbb{R}$. For some arguments $x_0, \ldots, x_n \in \mathbb{R}$, let

$$y_0 := \varphi(x_0),\ y_1 := \varphi(x_1), \ldots,\ y_n := \varphi(x_n).$$

Construct a polynomial $f \in \mathbb{R}[x]$ such that

$$f(x_0) = y_0 = \varphi(x_0),\ f(x_1) = y_1 = \varphi(x_1), \ldots,\ f(x_n) = y_n = \varphi(x_n).$$

We can say that f approximates φ by interpolating it in $n+1$ points. It is obvious that increasing the number of points, we force the graph of f to intersect the graph of φ in more places. Does it mean that f approximates φ "better"? Surprisingly it is not necessarily so. In fact, the "quality of approximation" strongly depends on how we position x_i's. A naïve approach of placing them at equal distances can lead to severe errors.

Example 52. Before we dig into details, let us first consider the following example due to C. Runge[5]. Take a function $\varphi : [-1, 1] \to \mathbb{R}$ defined by a formula:

$$\varphi := \frac{\frac{1}{25}}{x^2 + \frac{1}{25}}$$

and build a sequence of polynomials $f_1, f_2, \ldots$, where f_n interpolates the values of φ at $n+1$ equally spaced arguments. We have

$$\begin{aligned}
f_1 &= \frac{1}{26} \\
f_2 &= -\frac{25}{26}x^2 + 1 \\
f_3 &= -\frac{225}{884}x^2 + \frac{259}{884} \\
&\vdots \\
f_5 &= \frac{125}{104}x^4 - \frac{45}{26}x^2 + \frac{59}{104} \\
f_{10} &= -\frac{390625}{1768}x^{10} + \frac{109375}{221}x^8 - \frac{51875}{136}x^6 + \frac{54525}{442}x^4 - \frac{3725}{221}x^2 + 1 \\
&\vdots
\end{aligned}$$

[5]Carl David Tolmé Runge (born 30 August 1856, died 3 January 1927) was a German mathematician and physicist.

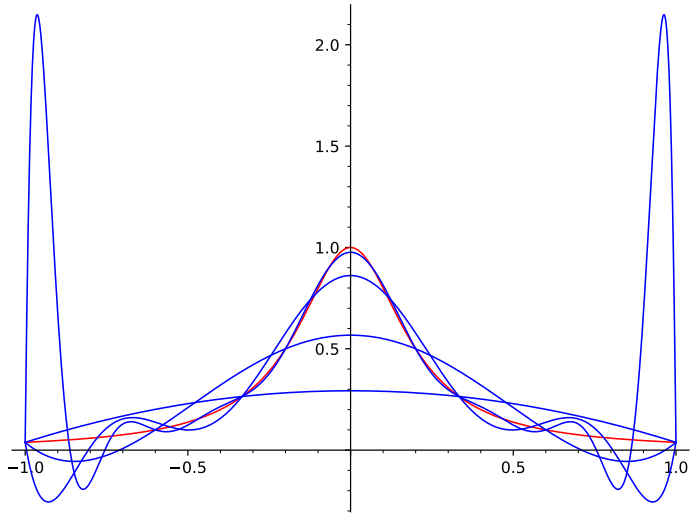


Figure 4.1: Wild oscillations of interpolation polynomials. Red line: the graph of φ, blue lines the graphs of f_3, f_5, f_{10}, f_{20}.

Figure 4.1 shows the graphs of φ and f_3, f_5, f_{10}, f_{20}. It is evident that for large n, the graph of f_n oscillates wildly. In fact it can be proved that

$$\lim_{n\to\infty}\left(\sup_{x\in[-1,1]}|\varphi(x)-f_n(x)|\right)=\infty,$$

likewise

$$\lim_{n\to\infty}\int_{-1}^{1}|\varphi(x)-f_n(x)|\,dx=\infty.$$

Both above formulas can be considered as measures of the "quality" of approximation.

Weierstrass[6] approximation theorem asserts that every real-valued function φ, defined on a compact interval, can be uniformly approximated by polynomials. But of course not every sequence of polynomials converges to φ. Runge's phenomenon, illustrated by the above example, shows that even the sequence of interpolating polynomials, matching φ in more and more points, does not have to be the correct one. Our aim in this section is to improve the approximation by smartly selecting the points x_i.

First recall that a *supremum norm* of a bounded real valued function $g:[a,b]\to\mathbb{R}$ is defined by a formula

$$\|g\|_\infty := \sup_{x\in[a,b]}|g(x)|.$$

[6]Karl Theodor Wilhelm Weierstraß (born 31 October 1815, died 19 February 1897) was a German mathematician.

A distance between two functions f and g is measured as $\|f-g\|_\infty$. We want to minimize the distance between a given function $\varphi : [a,b] \to \mathbb{R}$ and a polynomial $f_n \in \mathbb{R}[x]$ interpolating it at $x_0, \dots, x_n \in [a,b]$. Define an auxiliary function $\delta : [a,b] \to \mathbb{R}$ putting

$$\delta(t) := \big(\varphi(t) - f_n(t)\big) + d \cdot \prod_{i=0}^{n} (t - x_i),$$

for some constant parameter $d \in \mathbb{R}$. Denote $\varpi(t) := (t-x_0)\cdots(t-x_n)$. Assume that φ is of class C^{n+1}, hence so is δ. Observe that $\delta(x_i) = 0$ for every $i \in \{0,\dots,n\}$. We want it to have one more zero $x \in [a,b]$, distinct from $x_0, \dots, x_n$. The condition $\delta(x) = 0$ implies that

$$d = \frac{\varphi(x) - f_n(x)}{\varpi(x)}. \tag{4.5}$$

Rolle's[7] theorem says that for every two consecutive roots $t_i, t_{i+1} \in \{x, x_0, \dots, x_n\}$ of δ, $t_i < t_{i+1}$, there is $\xi_i \in (t_i, t_{i+1})$ such that $\delta'(\xi_i) = 0$. Since δ has $n+2$ zeros in $[a,b]$, its derivative has at least $n+1$ roots in this interval. Inductively, δ'' has at least n roots, δ''' ast least $n-1$,... and so on. Finally we arrive at δ^{n+1} that must have at least one zero $\xi \in [a,b]$.

Now, f_n is a polynomial of degree n, therefore $f_n^{(n+1)} = 0$. On the other hand, $\varpi^{(n+1)} = (n+1)!$. It follows that

$$\delta^{(n+1)}(t) = \varphi^{(n+1)}(t) + d \cdot (n+1)!.$$

Substituting ξ for t we have

$$d = -\frac{\varphi^{(n+1)}(\xi)}{(n+1)!}. \tag{4.6}$$

Using Eq. (4.5) and (4.6) we obtain

$$\varphi(x) - f_n(x) = \varpi(x) \cdot d = -\varpi(x) \cdot \frac{\varphi^{(n+1)}(\xi)}{(n+1)!}.$$

This gives us a formula to estimate a distance between φ and the interpolating polynomial f_n.

Observation 4.2.1. *Denote by M the norm $\|\varphi^{(n+1)}\|_\infty$ of the $(n+1)$-th derivative of φ, then*

$$\|\varphi - f_n\|_\infty \le M \cdot \frac{\|\varpi\|_\infty}{(n+1)!}.$$

We have no control over M, as φ is a function given. But we may minimize $\|\varpi\|_\infty$ by a careful placement of arguments $x_0, \dots, x_n$. To this end we need to find a polynomial of a fixed degree with the lowest possible norm. The existence and uniqueness of such a polynomial is provided by the following theorem due to P. Chebyshev[8]:

[7] Michel Rolle (born 21 April 1652, died 8 November 1719) was a French mathematician

[8] Pafnuty Lvovich Chebyshev (born 16 May 1821, died 8 December 1894) was a Russian mathematician.

Theorem 4.2.2 (Chebyshev). *There is a unique monic polynomial $T \in \mathbb{R}[x]$ of degree n such that*

$$\|T\|_\infty = \min\{\|f\|_\infty \mid f \text{ is a monic polynomial, } \deg f = n\}.$$

Moreover the polynomial T is given by a formula

$$T = \frac{1}{2^{n-1}} \cdot \cos\big(n \cdot \arccos(x)\big)$$

and $\|T\|_\infty = 1/2^{n-1}$, where all the norms are computed over the interval $[-1,1]$.

Proof

The polynomial $2^{n-1} \cdot T = \cos\big(n \cdot \arccos(x)\big)$ is called the *Chebyshev polynomial of the first kind* of degree n and denoted T_n. The first ten Chebyshev polynomials are listed in Table 4.1. There is a simple recursive formula for constructing these polynomials.

Algorithm 4.2.3. *Let x be either a free variable or a real number. Given $k \in \mathbb{N}$ this algorithm computes $T_k(x)$.*

1. *If $k = 0$, then return 1 and quit;*
2. *if $k = 1$, then return x and quit;*
3. *otherwise recursively compute $T_{k-1}(x)$ and $T_{k-2}(x)$;*
4. *return $2x \cdot T_{k-1}(x) - T_{k-2}(x)$.*

Proof of correctness. The two special cases $T_0 = 1$ and $T_1(x) = x$ are obvious. Start from a trigonometric identity

$$\cos(\alpha+\beta) = \cos(\alpha)\cos(\beta) - \sin(\alpha)\sin(\beta).$$

Use it to express $\cos(k\alpha)$ as

$$\begin{aligned}
\cos(k\alpha) &= \cos(2\alpha)\cos\big((k-2)\alpha\big) - \sin(2\alpha)\sin\big((k-2)\alpha\big) \\
&= \big(2\cos^2(\alpha) - 1\big)\cos\big((k-2)\alpha\big) - 2\sin(\alpha)\cos(\alpha)\sin\big((k-2)\alpha\big) \\
&= 2\cos(\alpha)\cdot\Big(\cos(\alpha)\cos\big((k-2)\alpha\big) - \sin(\alpha)\sin\big((k-2)\alpha\big)\Big) - \cos\big((k-2)\alpha\big) \\
&= 2\cos(\alpha)\cos\big((k-1)\alpha\big) - \cos\big((k-2)\alpha\big).
\end{aligned}$$

Substituting $\arccos(x)$ for α we have

$$\cos\big(k \cdot \arccos(x)\big) = 2x \cdot \cos\big((k-1)\cdot\arccos(x)\big) - \cos\big((k-2)\cdot\arccos(\alpha)\big).$$

The left-hand-side is T_k, while the right-hand-side equals $2x \cdot T_{k-1}(x) - T_{k-2}(x)$. □

Basic trigonometry let us find the roots of T_n. Recall that cos vanishes at odd multiples of $\pi/2$. Hence we need to have

$$n \cdot \arccos(x) = (2j+1)\cdot\frac{\pi}{2}.$$

Solve it for x to get:

Observation 4.2.4. *A Chebyshev polynomial T_n has n distinct roots $x_0, \dots x_{n-1} \in [-1,1]$, with*

$$x_j = \cos\Big(\frac{2j+1}{2n} \cdot \pi\Big).$$

The roots of a Chebyshev polynomial are called *Chebyshev nodes*. Table 4.2 contains approximations of Chebyshev nodes up to degree 9. Observe that

$$x \mapsto \frac{2x-a-b}{b-a}$$

is an order-preserving bijection from $[a,b]$ onto $[-1,1]$. We can use now Theorem 4.2.2 to minimize the bound obtained in Observation 4.2.1.

Corollary 4.2.5. *Take a function $\varphi : [a,b] \to \mathbb{R}$ of class C^{n+1} and denote*

$$M := \|\varphi^{(n+1)}\|_\infty = \sup_{x\in[a,b]} |\varphi^{(n+1)}(x)|.$$

Let $\overline{x}_0, \dots, \overline{x}_n \in [-1,1]$ be the roots of T_{n+1}. If f_n is a polynomial interpolating φ at

$$x_0 = \frac{2\overline{x}_0 - a - b}{b-a},\ x_1 = \frac{2\overline{x}_1 - a - b}{b-a}, \dots\ x_n = \frac{2\overline{x}_n - a - b}{b-a},$$

then the distance $\|\varphi - f_n\|_\infty$ between φ and f_n is bounded from above by

$$\frac{1}{2^n \cdot (n+1)!} \cdot M.$$

Example 53 (continued). We can experience the strength of using Chebyshev nodes by considering the same function $\varphi : [-1,1] \to \mathbb{R}$,

$$\varphi(x) = \frac{\frac{1}{25}}{x^2 + \frac{1}{25}},$$

that we have encountered in the previous example. Take x_i's from Table 4.2 and construct the interpolating polynomials. For example f_3 is

$$-2.08166817117217\times10^{-17}x^3 - 0.240096038415366x^2 - 1.21430643318376\times10^{-17}x + 0.2496998$$

while f_5 equals

$$6.24500451351651\times10^{-17}x^5 + 0.711566513679867x^4 - 7.63278329429795\times10^{-16}x^3 - 1.0958124$$

Figure 4.2 shows the graphs of f_3, f_5, f_{10} and f_{20}. The wild oscillations seems to be gone. Indeed, it can be shown that this time

$$\lim_{n\to\infty} \|\varphi - f_n\|_\infty = \lim_{n\to\infty} \Big(\sup_{x\in[-1,1]} |\varphi(x) - f_n(x)| \Big) = 0.$$

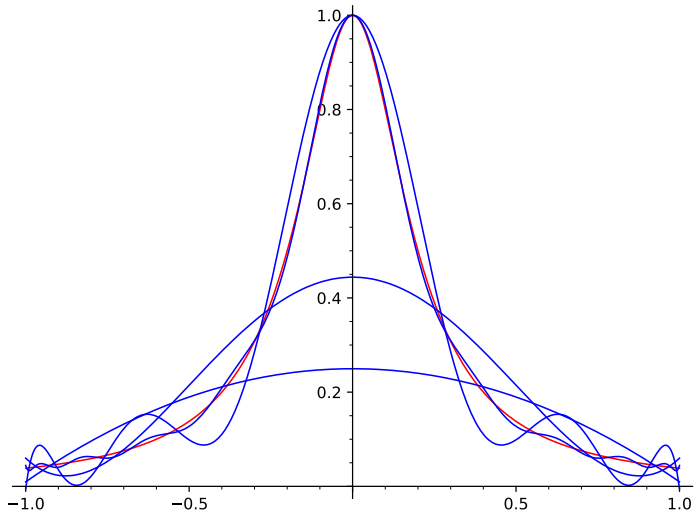


Figure 4.2: Interpolation of Runge's function using Chebyshev nodes. The red line is φ, the blue lines are: f_3, f_5, f_{10}, f_{20}.

Table 4.1: First 10 Chebyshev polynomials

$$
\begin{aligned}
f_0 &= 1\\
f_1 &= x\\
f_2 &= 2x^2 - 1\\
f_3 &= 4x^3 - 3x\\
f_4 &= 8x^4 - 8x^2 + 1\\
f_5 &= 16x^5 - 20x^3 + 5x\\
f_6 &= 32x^6 - 48x^4 + 18x^2 - 1\\
f_7 &= 64x^7 - 112x^5 + 56x^3 - 7x\\
f_8 &= 128x^8 - 256x^6 + 160x^4 - 32x^2 + 1\\
f_9 &= 256x^9 - 576x^7 + 432x^5 - 120x^3 + 9x
\end{aligned}
$$

Table 4.2: Roots of the first ten Chebyshev polynomials.

0				
1	0.00000000			
2	−0.70710678	0.70710678		
3	−0.86602540	0.00000000	0.86602540	
4	−0.92387953	−0.38268343	0.38268343	0.92387953
5	−0.95105652	−0.58778525	0.00000000	0.58778525
	0.95105652			
6	−0.96592583	−0.70710678	−0.25881905	0.25881905
	0.70710678	0.96592583		
7	−0.97492791	−0.78183148	−0.43388374	0.00000000
	0.43388374	0.78183148	0.97492791	
8	−0.98078528	−0.83146961	−0.55557023	−0.19509032
	0.19509032	0.55557023	0.83146961	0.98078528
9	−0.98480775	−0.86602540	−0.64278761	−0.34202014
	0.00000000	0.34202014	0.64278761	0.86602540
	0.98480775			

4.3 Piecewise polynomial interpolation

Interpolation of a certain input data by a single polynomial function has some limitations and can lead to undesired artifacts as Runge's phenomenon observed in the previous section. We already know one remedy—select the interpolation nodes at roots of Chebyshev polynomials. But what if we cannot change the nodes (for example we are modeling some physical or financial process, that we may sample only at fixed time intervals)? The way to go is to use piecewise polynomial functions.

Definition 4.3.1. Let $a = x_0 < x_1 < \cdots < x_n = b$ be a finite, strictly increasing sequence of real numbers. We say that a function $f : [a, b] \to \mathbb{R}$ is *piecewise polynomial* of degree $\leq d$ if for every $i \in \{0, \ldots, n-1\}$ its restriction $f\big|_{[x_i, x_{i+1}]}$ is a polynomial function of degree not exceeding d.

Piecewise polynomial function are also called *splines*[9]. Since $f\big|_{[x_i, x_{i+1}]}$ is a polynomial function for every i, it follows that f is of class C^∞ everywhere but in $x_1, \ldots, x_{n-1}$. Thus, smoothness of f depends entirely on its behavior at nodes.

Hermite's splines

TO DO

[9] Some textbooks define splines to be piecewise polynomial functions of a certain class, typically C^{d-1}. Here we do not enforce this additional condition.

Natural cubic splines

TO DO

B-splines

TO DO

4.4 Fourier transform and fast multiplication

This section must be completely reworked

Fourier series and Fourier transform

Fourier series is a method of expressing a given periodic function f as an infinite sum of sines and cosines with certain coefficients

$$f(t) \approx \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos\Big(nt \cdot \frac{2\pi}{L}\Big) + b_n \sin\Big(nt \cdot \frac{2\pi}{L}\Big),$$

where $L > 0$ is a period.

Theorem 4.4.1. *Let $f : \mathbb{R} \to \mathbb{R}$ be a periodic function with a period L and assume that f is absolutely integrable on $(-L/2, L/2)$. Then the functional series*

$$\mathscr{F}(t) := \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos\Big(nt \cdot \frac{2\pi}{L}\Big) + b_n \sin\Big(nt \cdot \frac{2\pi}{L}\Big),$$

where

$$a_n = \frac{2}{L}\int_{-L/2}^{L/2} f(T)\cos\Big(nT \cdot \frac{2\pi}{L}\Big)dT \qquad b_n = \frac{2}{L}\int_{-L/2}^{L/2} f(T)\sin\Big(nT \cdot \frac{2\pi}{L}\Big)dT$$

is point-wise convergent to f. If additionally f is C^2-smooth, then the series is uniformly convergent.

Example 54. Take a function $f(x) = \operatorname{sgn}\big(\sin(x)\big)$ and compute partial sums of its Fourier series:

$$\begin{aligned}
f_1 &= \frac{4\sin(x)}{\pi} \\
f_5 &= \frac{4\sin(x)}{\pi} + \frac{4\sin(5x)}{5\pi} + \frac{4\sin(3x)}{3\pi} \\
f_{10} &= \frac{4\sin(x)}{\pi} + \frac{4\sin(9x)}{9\pi} + \frac{4\sin(7x)}{7\pi} + \frac{4\sin(5x)}{5\pi} + \frac{4\sin(3x)}{3\pi} \\
f_{30} &= \frac{4\sin(x)}{\pi} + \frac{4\sin(29x)}{29\pi} + \frac{4\sin(27x)}{27\pi} + \cdots + \frac{4\sin(3x)}{3\pi}
\end{aligned}$$

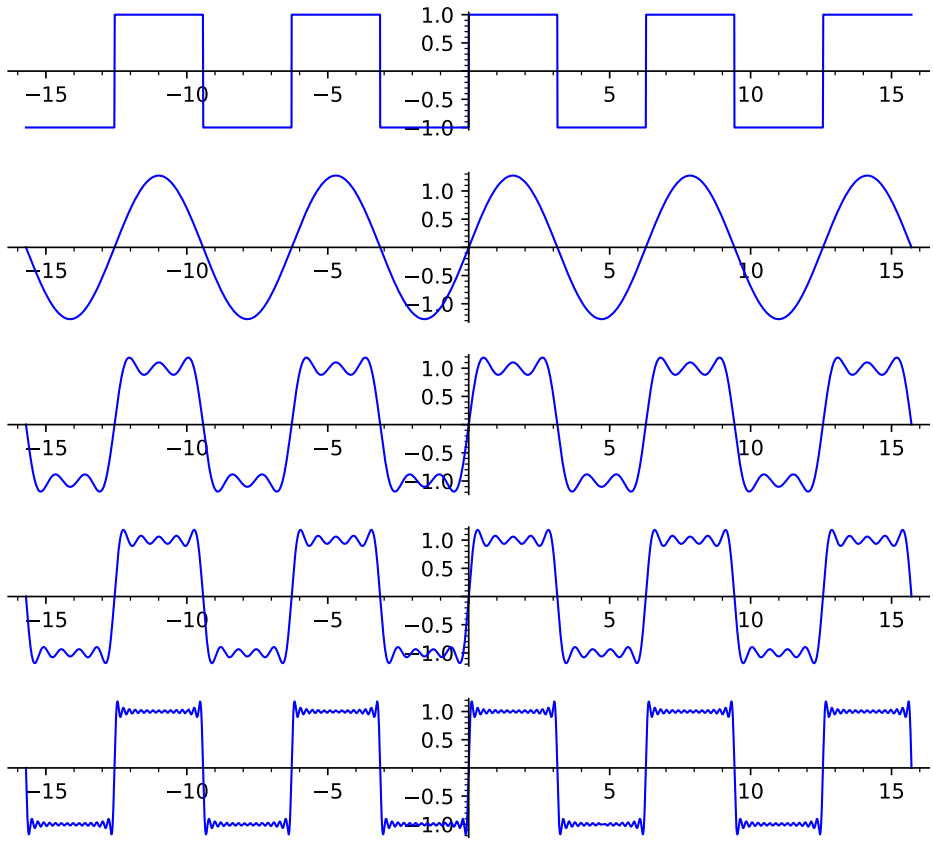


Figure 4.3: Graphs of $f(x) = \operatorname{sgn}\big(\sin(x)\big)$ and partial sums of its Fourier series.

The graphs of f and f_1, f_5, f_{10}, f_{30} are presented in Figure 4.3.

Using a well known identity

$$e^{i\alpha} = \cos(\alpha) + i\sin(\alpha),$$

we can write a Fourier series in a alternative form as

$$f \approx \sum_{n=-\infty}^{\infty} c_n \cdot e^{int\frac{2\pi}{L}},$$

where c_n is given by the formula

$$c_n = \frac{1}{L}\int_{-L/2}^{L/2} f(T)\cdot e^{-inT\frac{2\pi}{L}}\,dT.$$

Now, informally speaking, a Fourier transform is a "continuous cousin" of a Fourier series, where a sequence $(c_n)_{n\in\mathbb{N}}$ is replaced by a function $c\colon \nu \mapsto c(\nu)$, traditionally denoted $\hat{f}$:

Definition 4.4.2. Let $f : \mathbb{R} \to \mathbb{C}$ be an integrable function, the *Fourier transform* of f is a function $\hat{f} : \mathbb{R} \to \mathbb{C}$ defined by the formula

$$\hat{f}(T) := \int_{-\infty}^{\infty} f(t)e^{-i2\pi tT}\,dt$$

Definition 4.4.3. If $F : \mathbb{R} \to \mathbb{C}$ is an integrable function, then the *inverse Fourier transform* of F is defined as

$$\check{F}(t) := \int_{-\infty}^{\infty} F(T)e^{i2\pi tT}\,dT$$

The next theorem presents the fundamental identity which bounds the forward and the inverse Fourier transforms.

Theorem 4.4.4 (Fourier)**.** *Let $f : \mathbb{R} \to \mathbb{C}$ be a continuous and absolutely integrable function such that its Fourier transform $\hat{f}$ is absolutely integrable, too. Then*

$$f = (f\hat{})\check{}.$$

Example 55. Take again $f(x) = \operatorname{sgn}\big(\sin(x)\big)$, then its Fourier transform is

$$F(T) = \hat{f}(T) = -\frac{\left(e^{\left(8\pi^2T\right)} - 2\,e^{\left(4\pi^2T\right)} + 1\right)e^{\left(-4\pi^2T\right)}}{4\pi T}.$$

In Figure 4.4 we present the graph of the imaginary part of F (the real part of F is identically zero — exercise: why?) together with superimposed coefficients of the Fourier series of f. So that the reader may visually convince himself (or herself, as it may be), that the Fourier transform is indeed a "continuous cousin" of the Fourier series.

Discrete Fourier transform

Discretizing the domains of f and F, we define the discrete Fourier transform and discrete inverse Fourier transform. If $f = (f_0, \ldots, f_{N-1})$ is a finite sequence, then the *discrete Fourier transform* of f is a sequence $\hat{f} = (\hat{f}_0, \ldots, \hat{f}_{N-1})$, where

$$\hat{f}_n := \sum_{k=0}^{N-1} f_k e^{-i2\pi kn/N}.$$

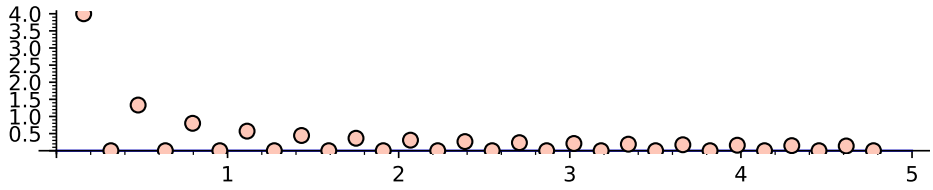


Figure 4.4: The (imaginary part of the) Fourier transform of $f(x) = \operatorname{sgn}\big(\sin(x)\big)$. The dots represent the coefficients of the Fourier series of f.

Analogously, we define *discrete inverse Fourier transform* of a finite sequence $F = (F_0, \ldots, F_{N-1})$ by the formula

$$\check{F}_k := \frac{1}{N} \sum_{n=0}^{N-1} F_n e^{i2\pi kn/N} \quad \text{for } k \in \{0, \ldots, N-1\}.$$

Now, take $\omega = e^{i \cdot \frac{2\pi}{N}} \in \mathbb{C}$ and observe that $\omega^N = e^{2\pi i} = 1$, while $\omega^k \neq 1$ for all $0 < k < N$. Such an element is called a *primitive root of unity* of degree N. This let us generalize the notion of the discrete Fourier transform to an arbitrary ring providing that a primitive root of unity is available.

Definition 4.4.5. Let R be a ring and $\omega \in R$ be a primitive root of unity of degree N. The *discrete Fourier transform* (DFT) of a finite sequence $f = (f_0, \ldots, f_{N-1})$ of length N consisting of elements from R is a sequence $\hat{f} = (\hat{f}_0, \ldots, \hat{f}_{N-1})$ defined by the formula

$$\hat{f}_n := \sum_{k=0}^{N-1} f_k \omega^{-kn}.$$

The *discrete inverse Fourier transform* (DIFT) of a sequence $F = (F_0, \ldots, F_{N-1})$ is a sequence $\check{F} = (\check{F}_0, \ldots, \check{F}_{N-1})$ given by

$$\check{F}_k := \frac{1}{N} \sum_{n=0}^{N-1} F_n \omega^{kn}.$$

Analogously to Theorem 4.4.4 we have:

Proposition 4.4.6. *Let R be a ring and $\omega \in R$ be a primitive root of unity of degree N. Then for any sequence f of length N the following identity holds*

$$f = (f\hat{})\check{}.$$

Take now a polynomial $f = f_0 + f_1 t + \cdots + f_n t^n \in R[t]$ and treat it as a sequence of coefficients $f \sim (f_0, \ldots, f_n)$. The the DFT $\hat{f}$ of f is a sequence $(\hat{f}_0, \ldots, \hat{f}_n)$, where

$$\hat{f}_j = \sum_{k=0}^{n} f_k \omega^{-kj} = f_0 + f_1 \cdot \Big(\frac{1}{\omega^j}\Big) + \cdots + f_n \cdot \Big(\frac{1}{\omega^j}\Big)^n = f\Big(\frac{1}{\omega^j}\Big).$$

It follows that $\hat{f} = \big(f(1), f(1/\omega), \ldots, f(1/\omega^n)\big)$. Now, since ω is a primitive root of unity of degree $n+1$, then so is $1/\omega$. Thus, we have proved:

Observation 4.4.7. *The discrete Fourier transform of* (*a sequence of coefficients of*) *a polynomial f, with respect to a primitive root of unity ω of degree $1+\deg f$, is a sequence of values of f at all the consecutive powers of $1/\omega$.*

Example 56. Take a polynomial $f = 1 + 2t + 3t^2 + 4t^3 \in \mathbb{Z}[t]$. The primitive root of unity of degree four is $i = \sqrt{-1} \in \mathbb{C}$. Using the above observation, we compute DFT of f:

$$\hat{f} = \big(f(1), f(-i), f(-1), f(i)\big) = (10, -2+2i, -2, -2-2i).$$

Convolution and polynomial multiplication

Recall that the *convolution* of two integrable functions g and h is a new function $f * g$ defined by the formula

$$(f * g)(x) := \int_{-\infty}^{\infty} f(\xi) \cdot g(x - \xi)\, d\xi$$

Again, we may define a discrete version of the convolution replacing functions by sequences and integrals by sums. Let $f, g : \mathbb{Z} \to \mathbb{R}$ be two two sequence of reals and assume that they are absolutely summable (i.e. $\sum |f|, \sum |g| < \infty$). The sequence $(f \hat{*} g)_{k \in \mathbb{Z}}$ defined by the formula

$$(f \hat{*} g)_k := \sum_{j=-\infty}^{\infty} f_j \cdot g_{k-j}$$

is called the *discrete convolution* of f and g. Once again, we wish to generalize the notion to an arbitrary ring. But in a general setting we do not have the reasonable notion of a convergence and consequently of infinite sums. Nevertheless, finite sequences are always summable and they suffice to our needs.

Definition 4.4.8. Let R be a ring. If $f = (f_0, \ldots, f_m)$ and $g = (g_0, \ldots, g_n)$ are two finite sequences of elements of R, then the sequence $(f \hat{*} g)_{0 \le k \le m+n}$ defined by the formula

$$(f \hat{*} g)_k := \sum_{j=\max\{0, k-n\}}^{\min\{k, m\}} f_j \cdot g_{k-j}$$

is called the *discrete convolution* of f and g.

Now, let f and g be two polynomials, treated as sequences of coefficients. Observe that discrete convolution of f and g is just the sequence of coefficients of their product as polynomials. Hence $\hat{*}$ is nothing else than just a polynomial multiplication. And here is the important point—the Fourier transport (either "continuous" or discrete) translates the convolution to a point-wise multiplication.

Theorem 4.4.9. *The following identities hold:*

$$(f * g)\hat{} = \hat{f} \cdot \hat{g} \qquad (f \cdot g)\hat{} = \hat{f} * \hat{g}$$
$$(F \cdot G)\check{} = \check{F} * \check{G} \qquad (F * G)\check{} = \check{F} \cdot \check{G}$$

Here $*$ denotes the convolution and $\hat{}, \check{}$ the forward and inverse Fourier transforms when f, g, F, G are function, and they denote respectively the discrete convolution and discrete forward and inverse Fourier transforms when f, g, F, G are sequences. In both cases $\cdot$ denotes the point-wise multiplication.

Corollary 4.4.10. *If $f, g \in K[t]$ are two polynomials, then*

$$f * g = (\hat{f} \cdot \hat{g})\check{}.$$

Here $*$ is the polynomial multiplication and $\cdot$ is the point-wise multiplication of their Fourier transforms.

Since the point-wise multiplication of two finite sequences of length n requires only n operations. Hence, the complexity of the polynomial multiplication using the above corollary is

$$O(n + \text{complexity of discrete Fourier transform})$$

Thus, if we devise a fast method for computing the Fourier transform we can cut down the complexity of the polynomial multiplication. We present one such an algorithm due to Cooley and Tukey (in fact these two authors rediscover in 1965 an algorithm invented by Gauss in 1805). The algorithm presented below is a special version of the Cooley-Tukey algorithm when one assumes that the sequence f has length $N = 2^n$. Take a sequence $(f_0, \dots, f_{N-1})$ and compute its DFT $\hat{f} = (\hat{f}_0, \dots, \hat{f}_{N-1})$. Split the formula for $\hat{f}_j$ into "even" and "odd" terms as follows:

$$\begin{aligned}\hat{f}_j = \sum_{k=0}^{2^n-1} f_k \omega^{-kj} = \sum_{k=0}^{2^{n-1}-1} f_{2k} \omega^{-2kj} + \sum_{k=0}^{2^{n-1}-1} f_{2k+1} \omega^{-(2k+1)j} \\ = \sum_{k=0}^{2^{n-1}-1} f_{2k} \big(\omega^2\big)^{-kj} + \omega^{-j} \cdot \sum_{k=0}^{2^{n-1}-1} f_{2k+1} \big(\omega^2\big)^{-kj}.\end{aligned}$$

Observe that the first term is nothing else but the j-th coefficient of DFT of the sequence $(f_0, f_2, \dots, f_{N-2})$ and the sum in the second term is the j-th coefficient of the discrete

Fourier transform of $(f_1, f_3, \ldots, f_{N-1})$. Moreover, $\omega^{N/2} = -1$ and so, for $j \geq N/2$ we have $\omega^j = -\omega^{j-N/2}$. Hence

$$\hat{f}_j := \begin{cases} \hat{e}_j + \omega^{-j}\hat{o}_j, & \text{for } j < N/2 \\ \hat{e}_{j-N/2} - \omega^{-j+N/2}\hat{o}_{j-N/2}, & \text{for } j \geq N/2. \end{cases}$$

The above discussion can be summarized in form of the following algorithm.

Algorithm 4.4.11 (Cooley-Tukey)**.** *Let R be a ring and $\omega \in R$ be a primitive root of unity of degree N, where N is a power of 2. Given $f = (f_0, \ldots, f_{N-1})$ a finite sequence of length N, this algorithm computes the discrete Fourier transform $\hat{f}$ of f.*

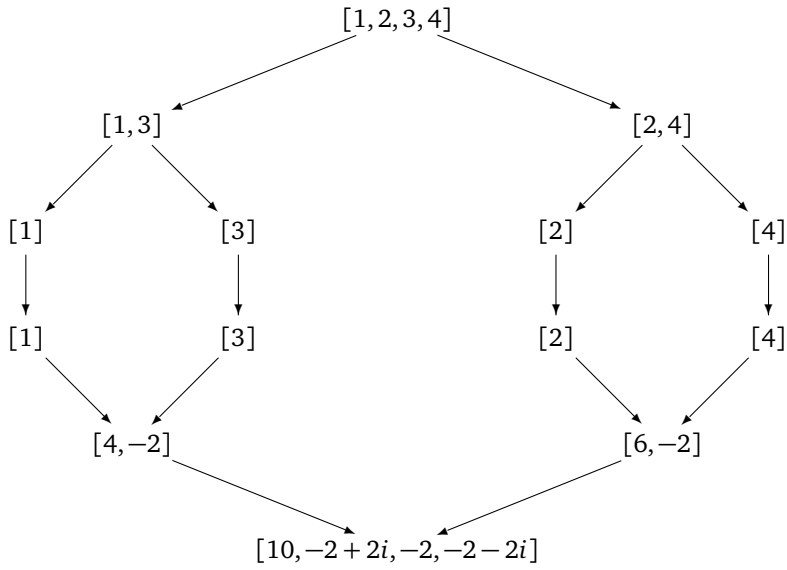
1. *If $n = 1$, then return $\hat{f} = f$;*
2. *otherwise take $f_{even} := (f_0, f_2, \ldots, f_{N-2})$ and $f_{odd} := (f_1, f_3, \ldots, f_{N-1})$ the subsequences of f consisting of elements with respectively even and odd indices;*
3. *compute recursively Fourier transforms $\hat{f}_{even} = (\hat{e}_0, \ldots, \hat{e}_{N/2-1})$ and $\hat{f}_{odd} = (\hat{o}_0, \ldots, \hat{o}_{N/2-1})$ of f_{even} and f_{odd};*
4. *combine the results taking*

$$\hat{f}_j := \hat{e}_j + \omega^{-j}\hat{o}_j$$
$$\hat{f}_{j+N/2} := \hat{e}_j - \omega^{-j}\hat{o}_j$$

   for $j \in \{0, \ldots, N/2 - 1\}$;
5. *return $\hat{f} = (\hat{f}_0, \ldots, \hat{f}_{n-1})$.*

Corollary 4.4.12. *The time complexity of Cooley-Tukey algorithm is $O(N \log N)$.*

Example 57. Take again (see Example 56) a sequence $[1, 2, 3, 4]$ of integers. The consecutive steps of DFT are presented in the following diagram:



Using Corollary 4.4.10, we devise a method of multiplying polynomials.

Algorithm 4.4.13. *Let $f = f_0 + f_1 t + \cdots + f_m t^m$ and $g = g_0 + g_1 t + \cdots + g_n t^n$ be two polynomials with coefficients in a ring R. Let $N \geq m + n + 1$ be a power of 2 and assume that R contains a primitive root of unity ω of degree N. Then one computes the product of f and g as follows:*

1. *let $\mathfrak{f}$, $\mathfrak{g}$ be the sequences of coefficients of f and g padded with zeros, so that they both have length N:*
$$\mathfrak{f} := (f_0, f_1, \ldots, f_m, 0, \ldots, 0),$$
$$\mathfrak{g} := (g_0, g_1, \ldots, g_n, 0, \ldots, 0);$$
2. *compute the discrete Fourier transforms $\hat{\mathfrak{f}} = (\hat{f}_0, \hat{f}_1, \ldots, \hat{f}_{N-1})$ and $\hat{\mathfrak{g}} = (\hat{g}_0, \hat{g}_1, \ldots, \hat{g}_{N-1})$ of $\mathfrak{f}$, $\mathfrak{g}$;*
3. *multiply the lists $\hat{\mathfrak{f}}$ and $\hat{\mathfrak{g}}$ coefficient-wisely, taking*
$$H := (\hat{f}_0 \cdot \hat{g}_0, \ldots, \hat{f}_{N-1} \cdot \hat{g}_{N-1});$$
4. *compute the discrete inverse Fourier transform $h := \check{H}$;*
5. *return h.*

The time complexity of the above algorithm (providing that the dominating operation is a scalar multiplication) is again $O(n \log n)$, where n is the maximum of degrees of f and g. Hence, this is a fast method of multiplying dense polynomials of large degrees. For polynomials of smaller degrees, the constant factors of time complexity of this algorithm causes it to be often slower than Karatsuba or long multiplication. For sparse polynomials, algorithms capable of omitting the zero coefficients tend to be faster, since in the above algorithm always all the coefficients are involved.

Example 58. Consider the following two polynomials $f = 1 + 2t + 3t^2 + 4t^3$ and $g = 2 + t + 4t^2 + 3t^3$. Write their lists of coefficients padded with zeros:

$$\mathfrak{f} = (1, 2, 3, 4, 0, 0, 0, 0)$$
$$\mathfrak{g} = (2, 1, 4, 3, 0, 0, 0, 0).$$

Compute DFT's of both getting:

$$\hat{\mathfrak{f}} = \big(10, 1 - 4\omega - 3\omega^2 - 2\omega^3, -2 + 2\omega^2, 1 - 2\omega + 3\omega^2 - 4\omega^3\big)$$
$$\big(-2, 1 + 4\omega - 3\omega^2 + 2\omega^3, -2 - 2\omega^2, 1 + 2\omega + 3\omega^2 + 4\omega^3\big),$$
$$\hat{\mathfrak{g}} = \big(10, 2 - 3\omega - 4\omega^2 - \omega^3, -2 + 2\omega^2, 2 - \omega + 4\omega^2 - 3\omega^3\big)$$
$$\big(2, 2 + 3\omega - 4\omega^2 + \omega^3, -2 - 2\omega^2, 2 + \omega + 4\omega^2 + 3\omega^3\big).$$

Multiply coefficient-wisely:

$$H = \big(100, -20 - 22\omega + 20\omega^3, -8\omega^2, -20 + 20\omega - 22\omega^3\big)$$
$$\big(-4, -20 + 22\omega - 20\omega^3, 8\omega^2, -20 - 20\omega + 22\omega^3\big).$$

Finally, compute the discrete inverse Fourier transform getting $h = 2 + 5t + 12t^2 + 22t^3 + 22t^4 + 25t^5 + 12t^6$.

The disadvantage of the above algorithm lays in the assumption that R contains the primitive root of unity ω of degree N. Of course, we may always pass to a bigger field, containing ω. Unfortunately for polynomials with exact coefficients (say integers or rational numbers), the cost of this operation may turn out to be prohibitive. For example, treating ω as a "symbolic entity" subject to a relation $\omega^N = 1$, de facto means working in a finite field/ring extension $R[\omega]$. But then, a multiplication of elements of this field/ring is equivalent to a multiplication of polynomials of degrees equal to the degree of the ring extension. All in all, in order to multiply to polynomials of degree n, we would end up multiplying twice as many 'polynomials in variable ω' of degree $\geq 2n+2$! Fortunately, there are ways to overcome this obstacle. In particular, when the polynomials in question have floating-point coefficients, we just use a floating-point complex roots of unity. This is the case where this algorithm really excels.

reference to a section on finite field extension arithmetic

As another option, if the polynomials f and g have integer coefficients, we may compute their product in a sufficiently large quotient ring $\mathbb{Z}_m = \mathbb{Z}/m$. The existence of a primitive root of unity is then guarantied by the following proposition.

Proposition 4.4.14. *Let $N, \omega > 1$ be integer powers of 2. Take $m := \omega^{N/2} + 1$. Then ω is a primitive root of unity in $\mathbb{Z}_m$ of degree N.*

For the proof of this proposition refer to [91, Theorem 3.3.9]. Coefficients of the product $f \cdot g$ are exactly retrievable from the quotient ring $\mathbb{Z}_m$, when elements of this ring are expressed as integers from the set $\{-m+1/2, \ldots, m-1/2\}$ and m is greater than

$$2 \cdot \max\{f_i : 0 \leq i \leq \deg f\} \cdot \max\{g_j : 0 \leq j \leq \deg g\} \cdot \big(\deg f + \deg g\big).$$

Therefore, this method is applicable, when working with large polynomials having small coefficients.

Binomial coefficients

As promised in Chapter 1 we will now present an application of the fast Fourier transform to computation of binomial coefficients in bulk. The key is the following lemma.

Lemma 4.4.15. *Let $v = (1,1)$. For $n \geq 1$ by $v^{(n)}$ denote the n-fold convolution $v * \cdots * v$. Then $v^{(n)}$ is the nth row of the Pascal triangle, i.e.:*

$$v^{(n)} = \left(\binom{n}{0}, \binom{n}{1}, \ldots, \binom{n}{n}\right).$$

Proof. We use induction on n. For $n = 1$ the assertion holds since $v^{(1)} = v = \left(\binom{1}{0}, \binom{1}{1}\right)$. Assume that the lemma is true for $n-1$ and compute $v^{(n)}$. We have

$$\begin{aligned} v^{(n)} = v * v^{(n-1)} &= (1,1) * \left(\binom{n-1}{0}, \binom{n-1}{1}, \dots, \binom{n-1}{n-1}\right) \\ &= \left(\dots, 1 \cdot \binom{n-1}{k} + 1 \cdot \binom{n-1}{k+1}, \dots\right) = \left(\dots, \binom{n}{k+1}, \dots\right). \end{aligned}$$

Here the last equality follows from Observation 1.1.22. □

In view of the above lemma, it is clear how to compute binomial coefficients:

1. Start with a list $v = (1,1)$.
2. Pad it with zeros to length $2^k > n$.
3. Compute FFT.
4. Raise every element of the result to n-th power.
5. Compute the inverse transform.

The calculations can be done using complex floating-point numbers with final result rounded to the nearest integers. However, to be fully safe from rounding errors it is desirable to use modular arithmetic. To this end we need to determine the size of a modulus. The biggest element of a row of the Pascal triangle is the middle one $\binom{n}{n/2}$. Proposition 1.1.28 asserts that $\binom{n}{n/2} \leq (2e)^{n/2}$. All involved integers are non-negative hence it suffices to take the modulus $m \geq 6^{n/2} \geq \lceil (2e)^{n/2} \rceil$.

Algorithm 4.4.16. *Given a positive integer n, this algorithm computes the binomial coefficients* $\binom{n}{0}, \dots, \binom{n}{n}$.

1. *If* $n = 1$*, output* $(1,1)$ *and terminate.*
2. *Let* $N = 2^k$ *be a power of* 2 *greater than or equal to* $n+1$.
3. *Let* $m \in \mathbb{Z}$ *be such that* $m \geq 6^{n/2}$ *and* $\mathbb{Z}_m$ *contains a primitive root of unity of degree* N (*use e.g. Proposition 4.4.14*).
4. *Take a vector* $v = (1,1,0,\dots,0)$ *of size* N *with coordinates in* $\mathbb{Z}_m$.
5. *Compute the Fourier transform* V *of* v.
6. *Compute the coordinate-wise n-th power* $W := V^n$.
7. *Compute the inverse Fourier transform* w *of* W.
8. *Output the first* $(n+1)$ *coordinates of* w *lifted to the ring of integers.*

Time complexity analysis. The fast Fourier transform and its inverse need $O(N \cdot \lg N) = O(n \cdot \lg n)$ arithmetic operations in $\mathbb{Z}_m$. The computation of the n-th power in step (6) by successive squaring has time complexity $O(n \cdot \lg n)$, too. All in all the asymptotic complexity of Algorithm 4.4.16 is

$$O\big(n \cdot \lg n \cdot M(6^{n/2})\big) = O\big(2^n \cdot n \cdot \lg n\big),$$

where M is the complexity of multiplication of integers.

Example 59. Suppose we ne need to compute the 6th row of the Pascal triangle. We have $n = 6$, hence $k = \lceil \lg 7 \rceil = 3$ and $N = 8$. Take $\omega = 2^{k-1} = 4$. Proposition 4.4.14 asserts that ω is a primitive root of unity of degree N in $\mathbb{Z}_{257}$. We have $v = (1, 1, 0, 0, 0, 0, 0, 0)$ and its Fourier transform $V = (2, 194, 242, 254, 0, 65, 17, 5)$. Consequently $W = V^6 = (64, 196, 128, 215, 0, 99, 129, 205)$. Computing the inverse transform and restricting ourselves to the first $(n + 1)$ entries we obtained the sought 6th row of the Pascal triangle: $1, 6, 15, 20, 15, 6, 1$.

Schönhage–Strassen algorithm, Stehle-Zimmermann GCD algorithm

Bibliographic notes and further reading

5

Factorization and irreducibility tests

> God does not play dice with the universe... but something is going on with the primes.
>
> Carl Pomerance, 1997

The fundamental theorem of arithmetic states that every integer (except ± 1) factors uniquely into a product of primes. Take for example $n = 8738366$. It can be factorized into

$$n = 2 \cdot 7^2 \cdot 13 \cdot 19^3.$$

The problem of finding prime factors of a given integer n — or proving that n is prime — has been prevalent in mathematics for centuries. In 1801, Carl Friedrich Gauss wrote:

> *The problem of distinguishing prime numbers from composite numbers and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic. It has engaged the industry and wisdom of ancient and modern geometers to such an extent that it would be superfluous to discuss the problem at length.... Further, the dignity of the science itself seems to require that every possible means be explored for the solution of a problem so elegant and so celebrated.*

We have often emphasized similarities between integers and univariate polynomials. In particular every monic polynomial can be uniquelly written as a product of monic irreducible polynomials. Hence the question about factorization is equally reasonable for polynomials. In fact, as we will see, the factorization of polynomials is substantially easier than factorization of integers. Nevertheless, we start with integers respecting their precedence.

5.1 Primality tests

In this section we examine some classical algorithms for testing if a given integer $n \in \mathbb{N}$ is a prime. Of course one can always divide n by $2, 3, \ldots$ and so on till $\lfloor \sqrt{n} \rfloor$. Then n is a prime number if and only if all these divisions produce nonzero remainders. It is clear that the time complexity of this approach is $O(\sqrt{n})$ and so is exponential in bit-size of n. It was a long standing problem weather there exists a deterministic primality test which has polynomial time with respect to the bit-size of the argument. This question was eventually affirmatively answered in 2002 by M. Agrawal, N. Kayal and N. Saxena. We present their algorithm at the end of this section, but earlier we discuss some classical probabilistic methods that are easier to digest first.

The general strategy behind the next few algorithms is to consider a property known to hold for elements of (finite) fields and test it on elements of a quotient ring $\mathbb{Z}_n := \mathbb{Z}/n\mathbb{Z}$. If we find an element $a \in \mathbb{Z}_n$ for which the property fails, then n is necessarily composite. We will usually pick a by random. Hence if our property does hold for a given element, then we don't really know whether n is prime or composite. All that we can say is that *n can possibly be a prime*, with a probability depending on the property of our choice.

Solovay–Strassen primality test

Let n be an odd positive integer. For any $a \in \mathbb{Z}$ by $\left(\frac{a}{n}\right)$ we denote the Jacobi symbol (see Definition 2.4.8). Recall that Euler's criterion (Theorem 2.4.3) asserts that if n is prime, then

$$\left(\frac{a}{n}\right) \equiv a^{\frac{n-1}{2}} \pmod{n}, \tag{5.1}$$

for every $a \in \mathbb{Z}$. Therefore, if we manage to find an integer a for which Eq. (5.1) does not hold, then we know for sure that n is a composite number. Otherwise we may suspect that n is possibly a prime.

Definition 5.1.1. Let $n > 0$ be an odd composite integer and a be a number relatively prime to n.

- If Eq. (5.1) holds for a and n, then n is called an *Euler pseudo-prime* to base a, while a is said to be an *Euler liar*.
- If Eq. (5.1) does not hold, then a is called an *Euler witness* for the compositeness of n.

The following observation is immediate, nevertheless still worth being written explicitly.

Observation 5.1.2. *If $a \equiv b \pmod{n}$, then a is an Euler witness* (*respectively Euler liar*) *if an only if b is an Euler witness* (*respectively Euler liar*).

Assume now that n is a composite number. Our aim is to estimate how likely is to find an Euler witness by a hit-an-miss method. Observation 5.1.2 tells us that it is

enough to restrict the search to the range $\{1,\dots,n-1\}$. In fact it suffices to look at the numbers between 2 and $n-2$, since 1 and $-1 \equiv n-1 \pmod{n}$ are always Euler liars. First we show that at least one Euler witness must exist.

Lemma 5.1.3. *Let $n > 2$ be an odd composite number. If n is square-free, then there exists an Euler witness for the compositeness of n.*

Proof. The integer n is square-free, hence is a product of distinct primes. Fix a prime p that divides n and let $m := {}^{n}/_{p}$. Then p and m are co-prime. Take any quadratic nonresidue b modulo p. The Chinese reminder theorem asserts that there is (exactly one) integer $a < n$ such that

$$a \equiv b \pmod{p} \qquad \text{and} \qquad a \equiv 1 \pmod{m}.$$

Compute the Jacobi symbol $\left(\frac{a}{n}\right)$:

$$\left(\frac{a}{n}\right) = \left(\frac{a}{pm}\right) = \left(\frac{a}{p}\right)\cdot\left(\frac{a}{m}\right) = \left(\frac{b}{p}\right)\cdot\left(\frac{1}{m}\right) = (-1)\cdot 1 = -1.$$

On the other hand $a^{(n-1)/2} \equiv 1^{(n-1)/2} \equiv 1 \pmod{m}$. In particular $a^{(n-1)/2}$ cannot be congruent to -1 modulo m, hence also modulo n. This shows that a is an Euler witness. □

The next step is to get rid of the assumption that n is square-free.

Proposition 5.1.4. *If $n > 2$ is an odd composite number, then there exists an Euler witness for the compositeness of n.*

Proof. If n is square-free, then the assertion follows from the lemma. Hence without loss of generality we may assume that we have a prime number p such that p^2 divides n. Denote $m := {}^{n}/_{p}$ and $a := 1 + m$. We claim that a is an Euler witness we are looking for. To this end compute the Jacobi symbol:

$$\left(\frac{a}{n}\right) = \left(\frac{1+m}{pm}\right) = \left(\frac{1+m}{p}\right)\cdot\left(\frac{1+m}{m}\right).$$

Now $1 + m \equiv 1 \pmod{m}$ and $1 + m \equiv 1 \pmod{p}$, since $p \mid m$. Hence

$$\left(\frac{a}{n}\right) = \left(\frac{1}{m}\right)\cdot\left(\frac{1}{m}\right) = 1.$$

In particular a is invertible modulo n.

On the other hand we have

$$a^p = (1+m)^p = 1 + \binom{p}{1}\cdot m + \binom{p}{2}\cdot m^2 + \cdots + m^p.$$

All the summands, except the first one, are divisible by $pm = n$, thus $a^p \equiv 1 \pmod{n}$. It follows that the order of a in the multiplicative group $U(\mathbb{Z}_n)$ divides p. But p is a prime number and $a \not\equiv 1 \pmod{n}$, therefore the order of a is precisely p. Now, p divides n, hence it cannot divide $n-1$, the more it cannot divide ${}^{(n-1)}/_{2}$. Consequently $a^{(n-1)/2} \not\equiv 1 \pmod{n}$. This shows that a is an Euler witness. □

Recall (see Definition 2.3.4) that $\varphi(n)$ is the number of elements in the unit group $U(\mathbb{Z}_n)$ or in other words a number of distinct remainders modulo n co-prime to n.

Theorem 5.1.5. *If $n > 2$ is an odd composite number, then in the set $\{2, \ldots, n-2\}$ there are at least $\varphi(n)/2$ Euler witnesses.*

Proof. If every integer relatively prime to n is an Euler witness, then the theorem is trivially true. Assume that there are some Euler liars out there. We will show that every Euler liar can be coupled with an Euler witness. To this end fix an Euler witness a. The previous proposition shows that such a exists. Suppose that b is an Euler liar. Take $c := a \cdot b$ and compute both sides of the test Eq. (5.1) for c:

$$c^{\frac{n-1}{2}} = (a \cdot b)^{\frac{n-1}{2}} = a^{\frac{n-1}{2}} \cdot b^{\frac{n-1}{2}} \not\equiv \left(\frac{a}{n}\right) \cdot \left(\frac{b}{n}\right) = \left(\frac{a \cdot b}{n}\right) = \left(\frac{c}{n}\right).$$

This shows that c is an Euler witness. Now, if $b_1, \ldots, b_k \in U(\mathbb{Z}_n)$ are all Euler liars for n, then $ab_1, \ldots, ab_k$ are distinct Euler witnesses. This shows that the number of Euler witnesses is at least $1/2 \cdot \sharp U(\mathbb{Z}_n) = \varphi(n)/2$. □

Remark 42. The reader familiar with basic group theory may observe that (for a fixed n) the set of Euler liars

$$\left\{a \in U\mathbb{Z}_n \mid \left(\frac{a}{n}\right) \equiv a^{(n-1)/2} \pmod{n}\right\}$$

is a subgroup of the multiplicative group $U\mathbb{Z}_n$ of units modulo n. It is proper subgroup by Proposition 5.1.4. Now, Lagrange theorem asserts that the cardinality of a subgroup divides the cardinality of the group. Thus the number of Euler liars is a proper divisor of $\varphi(n) = \sharp U\mathbb{Z}_n$. In particular the number cannot exceed $\varphi(n)/2$.

The above theorem proves the correctness of the following randomized polynomial time primality test.

Algorithm 5.1.6 (Solovay–Strassen primality test, 1977)**.** *Given an odd integer $n > 2$ this algorithm returns either 'composite' or 'probably prime'. If it returns 'composite', then n is definitely a composite number. Otherwise n is a prime with probability $> 1/2$.*

1. *Pick a random integer $a \in \{2, \ldots, n-2\}$;*
2. *if $\gcd(a, n) \neq 1$, then return 'composite' (here we actually know a non-trivial factor of n, namely $\gcd(a, n)$);*
3. *compute the Jacobi symbol $\left(\frac{a}{n}\right)$;*
4. *if $\left(\frac{a}{n}\right) \not\equiv a^{(n-1)/2} \pmod{n}$, then return 'composite', otherwise return 'probably prime'.*

Remark 43. In practice the above test would be executed up to k times in a loop. If it returns 'probably prime' for k uniformly distributed random elements $a \in \{2, \ldots, n-2\}$, then the probability that n is a genuine prime is greater than $1-2^{-k}$. Given some $\varepsilon > 0$, in order to achieve confidence greater than $1-\varepsilon$, it suffices to take $k > \lceil -\lg \varepsilon \rceil$.

Miller–Rabin primality test

Recall that if p is a prime number, then Fermat little theorem (Theorem 2.3.2) asserts that, a^{p-1} is congruent to 1 modulo p for every a relatively prime to p. This property of prime numbers is used in Fermat test. Like with Solovay–Strassen test, if we find some a for which the property does not hold, then the number we are examining is definitely composite.

Definition 5.1.7. Let $n > 2$ be an odd composite integer and $a \in \mathbb{Z}$ be a number relative prime to n.

- If $a^{n-1} \equiv 1 \pmod{n}$, then n is called a *Fermat pseudo-prime* to base a and a is said to be a *Fermat liar*.
- If $a^{n-1} \not\equiv 1 \pmod{n}$, then a is called a *Fermat witness* for the compositeness of n.

Analogously to Observation 5.1.2 we have:

Observation 5.1.8. *If a is co-prime to n and $b \equiv a \pmod{n}$, then a, b are either simultaneously Fermat liars or simultaneously Fermat witnesses.*

We now have two notions of pseudo-primality: Euler's and Fermat's ones. We should explain how they are related.

Proposition 5.1.9. *Every Euler pseudo-prime (to some base) is a Fermat pseudo-prime (to the same base).*

Proof. Assume that n is an Euler pseudo-prime to base a, then

$$\left(\frac{a}{n}\right) \equiv a^{\frac{n-1}{2}} \pmod{n}.$$

Compute the squares of both sides of the congruency to obtain

$$1 \equiv a^{n-1} \pmod{n}.$$

This shows that n is a Fermat pseudo-prime to base a. □

It is tempting to expect that and integer n satisfying the condition $a^{n-1} \equiv 1 \pmod{n}$ for sufficiently many a's is likely a prime. Unfortunately there are integers that are Fermat pseudo-primes to **all** co-prime bases.

Definition 5.1.10. A composite integer n is called a *Carmichael*[1] *number* if

$$a^{n-1} \equiv 1 \pmod{n}$$

for every a relatively prime to n.

Table 5.1: First 30 Carmichael numbers

i	C_i	i	C_i	i	C_i
1	561	11	41041	21	172081
2	1105	12	46657	22	188461
3	1729	13	52633	23	252601
4	2465	14	62745	24	278545
5	2821	15	63973	25	294409
6	6601	16	75361	26	314821
7	8911	17	101101	27	334153
8	10585	18	115921	28	340561
9	15841	19	126217	29	399001
10	29341	20	162401	30	410041

The Carmichael numbers are extremely rare. In fact they are substantially rarer than prime numbers, yet they still exist and there are still infinitely many of them. For the proofs of these two facts we refer the reader to . Table 5.1 collects the first 30 Carmichael numbers. Now, we will estimate the probability that a composite integer n passes Fermat primality test, providing that n is not a Carmichael number.

missing ref

Lemma 5.1.11. *Let n be an odd composite integer and a be an integer relatively prime to n. Then n is a Fermat pseudo-prime to base a if and only if the order of the class of a in the multiplicative group $U\mathbb{Z}_n$ divides $n-1$.*

Proof. The integer a is relatively prime to n, hence it is invertible in the quotient ring $\mathbb{Z}_n = \mathbb{Z}/n\mathbb{Z}$. Assume that a is a Fermat liar and let d be the order of a in the multiplicative group $U\mathbb{Z}_n$. Compute the quotient q and remainder r of $n-1$ modulo d. We have

$$n-1 = q\cdot d + r \qquad \text{and} \qquad 0 \le r < d.$$

By assumption $1 \equiv a^{n-1} \equiv a^{dq+r} \equiv 1^q \cdot a^r \pmod{n}$. Now, d is the smallest positive integer such that $a^d \equiv 1 \pmod{n}$, hence $r = 0$, which means that d divides $n-1$. Conversely, if $n-1 = q\cdot d$, then clearly $a^{n-1} = (a^d)^q \equiv 1 \pmod{n}$. □

Lemma 5.1.12. *Let a, b be two integers relatively prime to n and let c be an integer such that $a\cdot c \equiv 1 \pmod{n}$. If n is a Fermat pseudo-prime to bases a and b simultaneously, then it is also a pseudo-prime to bases ab and c*

Proof. By assumption we have

$$a^{n-1} \equiv 1 \pmod{n} \qquad \text{and} \qquad b^{n-1} \equiv 1 \pmod{n}.$$

Multiplying both congruences side-by-side we prove the first assertion. In order to show the other one, denote by $\overline{a}, \overline{c}$ the classes of a and c in the multiplicative group

[1] Robert Daniel Carmichael (born March 1, 1879, died May 2, 1967) was an American mathematician.

$U\mathbb{Z}_n$. We then have $\overline{c}^{n-1} = (\overline{a}^{-1})^{n-1} = (\overline{a}^{n-1})^{-1} = 1^{-1} = 1$. This means that $c^{n-1} \equiv 1 \pmod{n}$. □

Theorem 5.1.13. *Let $n > 0$ be a composite integer which is not a Carmichael number. Then at least half of all the elements of $U\mathbb{Z}_n$ are Fermat witnesses for the compositeness of n.*

Proof. Let S be a set of all Fermat liars:

$$S := \left\{ b \in U\mathbb{Z}_n \mid b^{n-1} \equiv 1 \pmod{n} \right\}.$$

Without loss of generality we may assume that S is not empty, since otherwise the assertion holds trivially. By assumption n is not a Carmichael number, hence there is at least one element $a \in U\mathbb{Z}_n$ which is not in S. Consider all the products of a form $a \cdot b$ for $b \in S$. We claim that they are all Fermat witnesses. Indeed, suppose that n is a Fermat pseudo-prime to some base ab. Let c be the inverse of b modulo n, that is $bc \equiv 1 \pmod{n}$. The previous lemma states that n is a pseudo-prime to base $(a \cdot b) \cdot c \equiv a \pmod{n}$, which is a contradiction. This shows that every Fermat liar b can be coupled with a Fermat witness ab. Thus the number of Fermat witnesses is greater or equal $1/2 \cdot \sharp U\mathbb{Z}_n = \varphi(n)/2$. □

If we know a priori that n is not a Carmichael number, we can test whether it is a prime in the following way:

Algorithm 5.1.14 (Fermat primality test)**.** *Given an integer $n > 1$, which is not a Carmichael number, this algorithm returns either 'composite' if it finds a Fermat witness for n or 'probably prime' if n is a prime with probability $> 1/2$.*

1. *Pick a random integer $a \in \{2, \ldots, n-2\}$;*
2. *if $a^{n-1} \equiv 1 \pmod{n}$, then return 'probably prime', otherwise return 'composite'.*

Notice that we do not have to test whether a is relatively prime to n before performing the check in step 2. Indeed, if $\gcd(a, n) \neq 1$, then $a^{n-1} \equiv 0 \pmod{n}$ by Theorem 2.3.8, since $\varphi(n) < n-1$. Remark 43 after Solovay–Strassen test applies also to Fermat test. The beauty of Fermat test lies in its remarkable simplicity. However, the serious limitation of the test is the existence of Carmichael numbers. In order to remove this obstacle, we need to investigate these numbers more scrutinously.

Observation 5.1.15. *Every Carmichael number is odd.*

Indeed, take a base $a = -1$, then for $n > 2$ we have $(-1)^{n-1} \equiv 1 \pmod{n}$ if and only if n is odd. Before we continue, we need to introduce the notion of *square-free* integers.

Definition 5.1.16. An element of a ring is said to be *square-free*, if it is not divisible by a non-unit square. In other words, $a \in R$ is square-free if and only if $b^2 \mid a$ implies that b is invertible in R.

In particular an integer is square free if and only if it is a product of distinct primes. The following criterion for a composite integer to be a Carmichael number was proved in 1899 by A. Korselt[2] about 10 years before the original work of Carmichael. The reader may wish to compare its proof with the proofs of Lemma 5.1.3 and Proposition 5.1.4.

Theorem 5.1.17 (Korselt, 1899)**.** *An odd composite integer n is a Carmichael number if and only if n is square-free and $p-1$ divides $n-1$ for every prime divisor p of n.*

Proof. Let n be a Carmichael number. Suppose that there is a prime number p such that p^2 divides n. Write n as a product $n = p^2 m$. We do not assume here that m is co-prime to p. Take a base $a := 1 + p \cdot m$ and compute

$$a^p = 1 + \binom{p}{1} \cdot pm + \binom{p}{2} \cdot p^2 m^2 + \cdots + p^p m^p.$$

Observe that all the terms, except the first one, are divisible by $n = p^2 m$. It follows that $a^p \equiv 1 \pmod{n}$. Hence, the order of a in the group of units of $\mathbb{Z}_n$ divides p and so it actually equals p, since p is prime. On the other hand, $a^{n-1} \equiv 1 \pmod{n}$ as n is a Carmichael number, therefore the order of a divides $n-1$, too. This is impossible as p cannot divide n and $n-1$ at the same time. This shows that n is square-free.

Now, take any prime divisor p of n and set $m := {}^{n}/_{p}$. It follows from the previous paragraph that p does not divide m. Theoremasserts that there is an integer $b \in \mathbb{Z}$ of order $p-1$ in the multiplicative group $U\mathbb{F}_p$. It follows from the Chinese remainder theorem that there is an integer a such that

missing

$$a \equiv b \pmod{p} \qquad \text{and} \qquad a \equiv 1 \pmod{m}.$$

In particular a is relatively prime to n and so $a^{n-1} \equiv 1 \pmod{n}$ by the definition of a Carmichael number. Consequently $a^{n-1} \equiv 1 \pmod{p}$ hence the order of a in $U\mathbb{F}_p$, which is the same as the order of b, divides $n-1$. Thus $p-1$ divides $n-1$, as claimed.

Conversely assume that n is a square-free integer such that $p-1 \,|\, n-1$ for every prime divisor p of n. Say $n = p_1 \cdots p_k$, where $p_1, \ldots, p_k$ are all distinct. Fix a base a relatively prime to n. Then by little Fermat theorem $a^{p_j - 1} \equiv 1 \pmod{p_j}$ for every $j \leq k$. Consequently also $a^{n-1} \equiv 1 \pmod{p_j}$ since $p_j - 1$ divides $n-1$ by assumption. Chinese remainder theorem asserts now that $a^{n-1} \equiv 1 \pmod{n}$. □

Corollary 5.1.18. *If n is a Carmichael number, then it is a product of at least three distinct primes.*

Proof. Let n be a Carmichael number, then n is not a prime. But it is square-free by Korselt theorem, hence it has at least two distinct prime divisors. In contradiction to the assertion suppose that n has precisely two prime divisors, say $n = p \cdot q$ with $p < q$. Then $q-1$ divides $n-1 = pq-1 = p \cdot (q-1) + (p-1)$. Therefore $q-1$ divides $p-1$, but this is impossible since $p < q$. □

[2] Alwin Reinhold Korselt (born 17 March 1864, died 4 February 1947) was a German mathematician.

Our goal is to develop a (randomized polynomial time) primality test that preserves the simplicity of Fermat test but does not share its vulnerability. In Fermat test we look just at a single expression, namely a^{n-1} and compute its remained modulo n. In Solovay–Strassen test, we also look at a single exponent but this time $a^{(n-1)/2}$. In Miller–Rabin test we are going to successively check $s+1$ successive exponents.

Let n be a positive odd integer. Write the $n-1$ as a product $n-1 = 2^s \cdot m$, where m is odd. For any element $a \in U\mathbb{Z}_n$ define a sequence $M_n(a) := (b_0, b_1, \ldots, b_s)$ setting

$$b_i := a^{2^i m} \qquad \text{for } i \in \{0, \ldots, s\}.$$

We have $b_0 = a^m$ and every other term of the sequence is a square of its predecessor. Moreover, if n is a Fermat pseudo-prime to base a (in particular if n is a Carmichael number), then b_s is necessarily equal 1.

Recall that in a field of an odd characteristic each square has precisely two square roots. In particular the square roots of 1 are ± 1. Thus, if we stumble upon a root of 1 (modulo n), which is neither equal 1 nor -1, we are sure that n cannot be a prime number. Start from the last element of the sequence. If b_s is not equal 1, then n is composite by Fermat condition. Otherwise, look at its predecessor, for n to be prime, $a^{2^{s-1}m}$ must be either 1 or -1. If it is not, we are sure that n is composite. If this element is also equal 1, then go back another step,... and so on. This way we have proved the following criterion:

Observation 5.1.19. *Let n be a positive odd integer and $a \in U\mathbb{Z}_n$. Let further $M_n(a) = (b_0, \ldots, b_s)$. If either $b_s \neq 1$ or $M_n = (\ldots, b, 1, \ldots, 1)$ for some $b \neq -1$, then n is a composite number.*

Since Fermat test is a special case of the above condition, we say that n is a *strong Fermat pseudo-prime* to base a if $M_n(a) = (\ldots, -1, 1, \ldots, 1)$ or $M_n(a) = (1, \ldots, 1)$. If it is the case, then a is called a *strong Fermat liar*. Otherwise we say that a is a *strong Fermat witness*. We will now show that for this new, stronger test Carmichael numbers no longer cause problems.

Fix a Carmichael number n. It is an odd integer by Observation 5.1.15. Now n, being a Carmichael number, is a Fermat pseudo-prime to every base co-prime to it. Thus, for every $a \in U\mathbb{Z}_n$ we have $b_s = 1$. Take a minimal index for which all sequences $M_n(a)$ stabilize:

$$k := \min\left\{ j \in \mathbb{N} \mid a^{2^j m} \equiv 1 \pmod{n}, \quad \text{for every } a \text{ co-prime to } n \right\}.$$

It is clear that $k \leq s$. On the other hand $k > 0$ because n is odd and taking a base $a = -1$ we have $(-1)^{2^0 m} = -1 \not\equiv 1 \pmod{n}$.

Lemma 5.1.20. *For every odd $n > 2$, there exists a strong Fermat witness.*

Proof. We may assume that n is a Carmichael number, since otherwise the assertion is trivial — we would find a such that $M_n(a) = (\ldots, b)$ with $b \neq 1$. The index k is minimal, hence there is an integer $b \in \mathbb{Z}$, relatively prime to n and such that $b^{2^{k-1}m} \not\equiv 1$

(mod n). Now n, being a Carmichael number, is square-free by Theorem 5.1.17, thus there is a prime divisor p of n for which $b^{2^{k-1}m} \not\equiv 1 \pmod{p}$. By the Chinese remainder theorem there is $a \in \mathbb{Z}$ satisfying congruences

$$a \equiv b \pmod{p} \qquad \text{and} \qquad a \equiv 1 \pmod{n/p}.$$

Then $a^{2^{k-1}m} \equiv b^{2^{k-1}m} \not\equiv 1 \pmod{p}$ and so $a^{2^{k-1}m}$ is not congruent to 1 modulo n. On the other hand $a^{2^{k-1}m} \equiv 1 \not\equiv -1 \pmod{n/p}$, therefore $a^{2^{k-1}m}$ cannot be congruent modulo n to -1, either. □

Theorem 5.1.21. *If n is a Carmichael number, then in the set $\{2, \ldots, n-2\}$ there are at least $\varphi(n)/2$ strong Fermat witnesses for compositeness of n.*

Proof. The proof of this theorem mimics the proof of Theorem 5.1.5. If all the elements in the set are strong witnesses, then there is nothing to do. Assume that $b_1, \ldots, b_l$ are all the strong liars. The previous lemma asserts that there is at least strong witness $a \in U\mathbb{Z}_n$. We then have

$$(ab_j)^{2^{k-1}m} = a^{2^{k-1}m} \cdot n^{2^{k-1}m}.$$

The second terms is congruent to ± 1 while the first one is not, hence the whole product cannot be congruent to ± 1. Thus every strong liar b_j can be paired with a strong witness ab_j. This proves the theorem. □

Remark 44. The reader may wish to notice that Remark 42 after Theorem 5.1.5 applies also to Theorem 5.1.21.

We are now ready to present a primality test that it is no longer fooled by Carmichael numbers.

Algorithm 5.1.22 (Miller–Rabin primality test, 1976/1980)**.** *Given a positive integer n this algorithm returns either 'composite' or 'probably prime'. If it returns 'composite', then n is definitely a composite number. Otherwise n is a prime with probability $> 1/2$.*

1. *Pick a random integer $a \in \{2, \ldots, n-2\}$;*
2. *find $s > 0$ and an odd integer m such that $n - 1 = 2^s \cdot m$;*
3. *set $b_0 := a^m \bmod n$;*
4. *if $b_0 = 1$ or $b_0 = -1$, then return 'probably prime';*
5. *repeat the following steps for $j \in \{1, \ldots, s\}$:*
   a) *compute $b_j := b_{j-1}^2 \bmod n$;*
   b) *if $b_j = 1$, then return 'composite';*
   c) *if $b_{j-1} = -1$, then return 'probably prime';*
6. *return 'composite'.*

Remark 45. Although we have proved the probability of a failure of this test to be less than $1/2$, it can be shown that this probability is in fact less than $1/4$. For the details we refer the reader e.g. to [43, §5.1] or to the original article by M. Rabin [70, Theorem 1].

Table 5.2: Thresholds for a deterministic Miller–Rabin test with successive prime bases.

k	bases a_k	thresholds T_k
1	2	2 047
2	2, 3	1 373 653
3	2, 3, 5	25 326 001
4	2, 3, 5, 7	3 215 031 751
5	2, 3, 5, 7, 11	2 152 302 898 747
6	2, 3, 5, 7, 11, 13	3 474 749 660 383
7	2, 3, 5, 7, 11, 13, 17	341 550 071 728 321
8	2, 3, 5, 7, 11, 13, 17, 19	341 550 071 728 321
9	2, 3, 5, 7, 11, 13, 17, 19, 23	3 825 123 056 546 413 051
10	2, 3, 5, 7, 11, 13, 17, 19, 23, 29	3 825 123 056 546 413 051
11	2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31	3 825 123 056 546 413 051
12	2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37	318 665 857 834 031 151 167 461
13	2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41	3 317 044 064 679 887 385 961 981

Miller–Rabin test is stochastic, as it depend on a random choice of a base a. Nevertheless, for small values of n it can be made deterministic. Table 5.2 summarizes thresholds T_k (for $k \leq 13$) such that if $n < T_k$, then it suffices to use the first k primes numbers as bases a in the test. For example, if $n < 2^{11} - 1 = 2047$ then it suffices to test a single base $a = 2$. If $n < 3\,215\,031\,751$ it suffices to successively check $a = 2$, 3, 5 and 7. A word of caution, taking first k successive primes not necessarily leads to a smallest set of bases.

Remark 46. Let n be a composite number. If the so called General Riemann Hypothesis (GRH) is true, which virtually everybody believes to be so (yet noone has been able to prove it so far), then it can be shown, that there is a strong witness a strictly less than $2 \cdot (\ln n)^2$. The reader may find a prove in either [7] or [23]. In particular, under GRH Algorithm 5.1.22 can be turned into a fully deterministic method running in a polynomial time with respect to the bit-size of n. To this end it suffices to check all possible bases form 2 to $\lfloor 2 \cdot (\ln n)^2 \rfloor$. This is in fact the original version of the test presented by G. Miller in his Ph.D. thesis [64].

Deterministic polynomial-time primality test

5.2 Integer factorization

Pollard ρ, Fermat's factorization -> Shank's method / Dixon's method -> quadratic sieve, Lenstra elliptic curves method (?)

5.3 Square-free decomposition of polynomials

In Section 5.1 we introduced a notion of square-freeness. Recall that an element a of a ring R is *square-free* (see Definition 5.1.16) if it is not divisible by a square of a non-unit element. In this section we show how to factor a polynomial into a product of square-free polynomials.

We first deal with the case when the field of coefficients contains $\mathbb{Q}$. In other words, it has *characteristic* zero. From now on let K be the field of coefficients and $\mathbb{Q} \subseteq K$. The ring $R = K[x]$ of polynomials over a field is Euclidean. In particular it is a unique factorization domain, hence every polynomial $f \in K[x]$ factors into a product of powers of irreducible polynomials

$$f = c \cdot p_1^{e_1} \cdots p_m^{e_m}, \tag{5.2}$$

where $c \in K$ and $p_1, \ldots, p_m \in K[x]$ are monic and irreducible. It is clear that f is square-free if and only if all the exponents $e_1, \ldots, e_k$ are equal 1. Equivalently, f is square-free if and only if it has only simple roots in an algebraic closure of K. For example take a polynomial $f = x^3 - 1 \in \mathbb{Q}[x]$. It is square-free, since $f = (x-1) \cdot (x^2 + x + 1)$.

It is a simple observation that every irreducible polynomial is necessarily square-free. The previous example shows that the opposite implication is not true. Nevertheless, there is an easy criterion to verify whether a polynomial is square-free.

Lemma 5.3.1. *Let ξ be a root of f (in an algebraic closure of the field of coefficients) of multiplicity $k \geq 1$. Then ξ is a root of the derivative f' of f if and only if $k > 1$.*

Proof. Write f as

$$f = (x - \xi)^k \cdot g,$$

where $g(\xi) \neq 0$. Compute the derivative of f using the product rule

$$f' = (x - \xi)^{k-1} \cdot \big(k \cdot g + (x - \xi) \cdot g'\big).$$

The expression in parenthesis does not vanish at ξ since $g(\xi) \neq 0$. Consequently, $f'(\xi) = 0$ if and only if $k > 1$. □

Theorem 5.3.2. *Let f be a polynomial over a field of characteristic 0 and let f' denote its (formal) derivative, then f is square-free if and only if f and f' are relatively prime.*

Proof. If f is not square-free then it has a root ξ of multiplicity strictly greater than 1 and it follows from the previous part of the proof that $(x - \xi)$ is a common divisor of f and f'. Conversely, suppose that f and f' are not relatively prime, let h be their nontrivial common divisor and ξ be any root of h. Then ξ is a common root of both f and f' and so, as a root of f, it has multiplicity $k > 1$. Consequently f is not square-free. □

This theorem may be directly converted into an algorithm.

Algorithm 5.3.3. *Given a polynomial $f \in K[x]$ over a field K with* $\operatorname{char} K = 0$*, this algorithm returns true if and only if f is square-free and false otherwise.*

1. *Compute the derivative f' of f using Algorithm 1.1.30;*
2. *compute the greatest common divisor $g := \gcd(f, f')$;*
3. *return true if g is invertible and false otherwise.*

Remark 47. The assumption that the characteristic of K is 0 is indispensable as the following example shows. Let p be a prime number and take $K := \mathbb{F}_p(t)$. Let $f \in K[x]$ be given by the formula $f = x^p - t$. It can be checked (e.g using Eisenstein criterion) that f is irreducible, hence square-free. Nevertheless, we have $F' = 0$ thus f and f' are not relatively prime. In fact $\gcd(f, f') = f$.

Let us return to the factorization Eq. (5.2) of f. Multiplication is commutative so we can rearrange the terms in the product, collating the irreducible polynomials with equal exponents. For j running from 1 up to $k := \max\{e_1, \ldots, e_m\}$ denote

$$g_j := \prod_{\substack{1 \le i \le m \\ e_i = j}} p_i$$

the product of all irreducible factors of f with multiplicities equal j. Then f can be expressed as a product

$$f = c \cdot g_1^1 \cdot g_2^2 \cdots g_k^k, \tag{5.3}$$

where $c \in K$ and all g_i are monic, square-free and pairwise co-prime. We allow some of them to be trivial (i.e. equal 1). In particular the roots of g_j for $1 \le j \le k$ are precisely the roots of f with multiplicity j. Eq. (5.3) is called the *square-free decomposition* of f.

Example 60. Take $f = (x-1) \cdot (x-2)^3 \cdot (x-3)^2 \cdot (x-4)^3 \cdot (x-5)^2 \cdot (x-6)$, then its square-free decomposition is

$$\begin{aligned} f &= ((x-6) \cdot (x-1)) \cdot ((x-5) \cdot (x-3))^2 \cdot ((x-4) \cdot (x-2))^3 \\ &= (x^2 - 7x + 6) \cdot (x^2 - 8x + 15)^2 \cdot (x^2 - 6x + 8)^3. \end{aligned}$$

As we will see, the square-free decomposition of a polynomial is quite easy to compute and does not require the prior knowledge of the irreducible factors of f. In fact, the square-free decomposition is often used as the initial step in polynomial factorization. Below we present a few algorithms from the square-free decomposition.

Tobey-Horowitz algorithm

In order to compute the square-free decomposition of a given polynomial f, we define three finite recursive sequences: $(a_i)_i$, $(b_i)_i$ and $(g_i)_i$, where g_i's are the desired square-free factors of f. Let $a_0 := f$ and $a_1 := \gcd(a_0, a_0')$. Assuming that the sought decomposition of f is $f = c \cdot g_1^1 \cdot g_2^2 \cdots g_k^k$, we have

$$a_1 = \gcd\big(c \cdot g_1 \cdot g_2^2 \cdots g_k^k, c \cdot g_1' \cdot g_2^2 \cdots g_k^k + \cdots + ka \cdot g_1 \cdot g_2^2 \cdots g_k^{k-1} g_k'\big).$$

Now, the polynomials $g_1, \ldots, g_k$ are pairwise co-prime and so are g_1 and g_1' since each g_i is square-free. It follows that

$$a_1 = c \cdot g_2 \cdot g_3^2 \cdots g_k^{k-1}.$$

Next, we define $a_2 := \gcd(a_1, a_1')$ and the analogous computation shows that $a_2 = c g_3 g_4^2 \cdots g_k^{k-2}$. In general, the sequence $(a_i)_{0 \le i \le k}$ is defined recursively as

$$a_0 := f \qquad \text{and} \qquad a_i := \gcd\big(a_{i-1}, a_{i-1}'\big) \quad \text{for} \quad i > 0.$$

Simple induction yields

$$a_i = c \cdot g_{i+1} g_{i+2}^2 \cdots g_k^{k-i}. \tag{5.4}$$

The second auxiliary sequence is obtained by taking quotients of consecutive a_i's. We set $b_i := {}^{a_{i-1}}/_{a_i}$. It follows from the previous part that b_i is a polynomial. In fact we have

$$b_i = \frac{a_{i-1}}{a_i} = \frac{c \cdot g_i \cdot g_{i+1}^2 \cdot g_{i+2}^3 \cdots g_k^{k-i+1}}{c \cdot g_{i+1} \cdot g_{i+2}^2 \cdots g_k^{k-i}} = g_i \cdots g_k, \quad \text{for } i \le k. \tag{5.5}$$

In particular the element b_1 is a product of all the square-free factors of f. Hence it is a square-free polynomial having precisely the same roots as f. It is called the *radical* of f and denoted $\operatorname{rad}(f)$ for reasons that will become clear in Chapter 9. The radical of a polynomial is important on its own right and often needed, thus substituting the values for a_0 and a_1, we shall explicitly note down the following:

Observation 5.3.4. *The radical of a polynomial f (with coefficients in a field of characteristic zero) is computed by the formula*

$$\operatorname{rad} f := \frac{f}{\gcd(f, f')}.$$

The formula (5.5) let us compute the square-free factors $g_1, \ldots, g_k$, we are after. We now have

$$g_i = \frac{b_i}{b_{i+1}}.$$

The above discussion proves the correctness of the following algorithm, discovered independently by R. Tobey and E. Horowitz.

Algorithm 5.3.5 (Tobey–Horowitz, 1967/1969)**.** *Given a monic polynomial f with coefficients in a field K of characteristic 0, this algorithm returns the square-free factors $(g_1, \ldots, g_k)$ of f.*

1. *Set $a_0 := f$;*
2. *initialize $i := 1$, $a_1 := \gcd(a_0, a_0')$ and $b_1 := {}^{a_0}/_{a_1}$;*
3. *as long as $b_i \neq 1$ repeat the following steps:*
   a) *compute the next term of the first auxiliary sequence setting $a_{i+1} := \gcd(a_i, a_i')$;*

b) compute the next term of the second sequence: $b_{i+1} := {}^{a_i}/_{a_{i+1}}$;

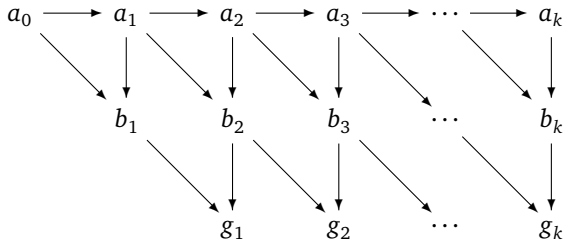
c) compute the i-th square-free factor by the formula

$$g_i := \frac{b_i}{b_{i+1}};$$

d) increment the counter i;

4. *return the list* $(g_1, \ldots, g_k)$.

The order in which various terms a_i, b_i and g_i are computed in Tobey–Horowitz algorithm is explained in the following diagram:



Time complexity analysis. As always, let $M(d)$ be the time complexity of multiplying polynomials of degree d. The number of iterations of the loop in step 3 of the algorithm equals k, the index of the last non-trivial square-free factor of f. It is clear that, for a given degree d, k is maximal when f is a k-th power of a square-free polynomial that is $f = g_k^k$. In such a case $\deg f = k \cdot \deg g_k$ and so $k = O(d)$. The four operations performed inside the loop are (in order of time complexity):

- Computation of a polynomial derivative. This has time complexity $O(d)$.
- Two (exact) polynomial divisions. They have time complexity $O\big(M(d)\big)$.
- A single gcd computation. When using a fast gcd algorithm, this step takes $O\big(d \lg(d)\big)$ time.

Thus it is the gcd computation that dominates. Consequently the time complexity of the whole algorithm is $O\big(d \cdot M(d) \cdot \lg(d)\big)$. With fast polynomial multiplication this is $O\big(d^2 \lg(d)^2\big)$.

Example 61. Take a polynomial $f = x^8 - 14x^7 + 82x^6 - 264x^5 + 513x^4 - 618x^3 + 452x^2 - 184x + 32 \in \mathbb{Q}[x]$. By definition $a_0 = f$ and

$$a_1 = \gcd(a_0, a_0') = x^5 - 7x^4 + 19x^3 - 25x^2 + 16x - 4.$$

This let us compute b_1:

$$b_1 = \frac{a_0}{a_1} = \frac{x^8 - 14x^7 + 82x^6 - 264x^5 + 513x^4 - 618x^3 + 452x^2 - 184x + 32}{x^5 - 7x^4 + 19x^3 - 25x^2 + 16x - 4} = x^3 - 7x^2 + 14x - 8.$$

Analogously, we compute a_2 and b_2:

$$a_2 = \gcd(a_1, a_1') = \gcd(x^5 - 7x^4 + 19x^3 - 25x^2 + 16x - 4, 5x^4 - 28x^3 + 57x^2 - 50x + 16) = x^3 - 4x$$
$$b_2 = \frac{a_1}{a_2} = \frac{x^5 - 7x^4 + 19x^3 - 25x^2 + 16x - 4}{x^3 - 4x^2 + 5x - 2} = x^2 - 3x + 2.$$

Having b_1 and b_2 we may, at last, compute the first square-free factor of f:

$$g_1 = \frac{b_1}{b_2} = \frac{x^3 - 7x^2 + 14x - 8}{x^2 - 3x + 2} = x - 4.$$

Continue the process and compute the remaining factors:

$$g_2 = 1, \qquad g_3 = x - 2, \qquad g_4 = x - 1.$$

All in all, our original polynomial factors into

$$f = g_1 \cdot g_2^2 \cdot g_3^3 \cdot g_4^4 = (x-4) \cdot (x-2)^3 \cdot (x-1)^4.$$

Exercise 9. The radical of a polynomials is sometimes confused with its *square-free part*. The square-free part of a polynomial f is a square-free polynomial g such that $f = g \cdot h^2$ for some polynomial h. Devising an appropriate algorithm, show that the square-free part of a polynomial can be obtained from its square-free decomposition.

Musser algorithm

There is an alternative method of computing the elements a_{i+1} and b_{i+1}. Using Eq. (5.4) and Eq. (5.5) we have

$$\gcd(a_i, b_i) = \gcd\big(g_{i+1} g_{i+2}^2 \cdots g_k^{k-i}, g_i g_{i+1} \cdots g_k\big) = g_{i+1} g_{i+2} \cdots g_k = b_{i+1}.$$

This gives as b_{i+1} but we still need a_{i+1}. Thus, we take

$$\frac{a_i}{b_{i+1}} = \frac{g_{i+1} g_{i+2}^2 \cdots g_k^{k-i}}{g_{i+1} g_{i+2} \cdots g_k} = g_{i+2} g_{i+3}^2 \cdots g_k^{k-i-1} = a_{i+1}.$$

This variation of a square-free factorization was devised by D. Musser and is known to be slightly faster (see below).

Algorithm 5.3.6 (Musser, 1971). *Given a monic polynomial f with coefficients in a field K of characteristic 0, this algorithm returns the square-free factors $(g_1, \ldots, g_k)$ of f.*

1. *Set $a_0 := f$;*
2. *initialize $i := 1$ and set $a_1 := \gcd(a_0, a_0')$, $b_1 := {}^{a_0}/_{a_1}$;*
3. *as long as $b_i \neq 1$ repeat the following steps:*

a) compute the next term of the second sequence taking $b_{i+1} := \gcd(a_i, b_i)$*;*

b) compute the next term of the first auxiliary sequence: $a_{i+1} := {}^{a_i}/_{b_{i+1}}$*;*

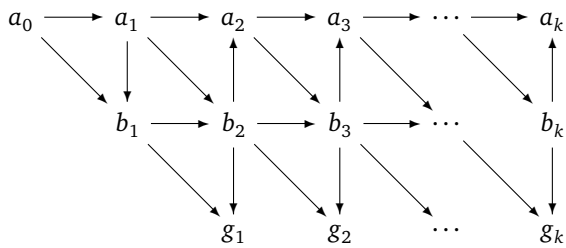
c) compute the i-th square-free factor by the formula

$$g_i := \frac{b_i}{b_{i+1}};$$

d) increment the counter i;

4. *return the list* $(g_1, \ldots, g_i)$.

Observe that in the Musser's algorithm, the element b_{i+1} is computed before a_{i+1}. The order of computations is, thus, different as one may see below:



Time complexity analysis. It is obvious that asymptotically the time complexity of the above algorithm is the same as of Tobey–Horowitz algorithm and equals $O\big(d \cdot M(d) \cdot \lg(d)\big)$. Nevertheless, the constant factors hidden in big-Oh notation are lower here. Thus in a computer implementation, Algorithm 5.3.6 is likely to beat Algorithm 5.3.5. The reasons of this phenomenon are 2-fold. For one, Musser method uses only one derivative. We no longer compute a polynomial derivative inside a loop. Secondly, the degree of b_i never exceeds the degree of a_i' and on average it is strictly lower. Thus the gcd in step 3a is computed for lower degree polynomials and so is faster.

Yun algorithms

Another classical method of square-free decomposition is due to D. Yun. It utilizes a different pair of auxiliary sequences, namely $(b_i)_{1 \le i \le k}$ and $(c_i)_{1 \le i \le k}$. They are again defined recursively. Let f be a monic polynomial with a square-free factorization $f = g_1 \cdot g_2^2 \cdots g_k^k$. Set

$$b_1 := \operatorname{rad}(f) = \frac{f}{\gcd(f, f')} \qquad \text{and} \qquad c_1 := \frac{f'}{\gcd(f, f')} - b_1'.$$

We have

$$f' = \sum_{i=1}^{k} i \cdot g_i' \cdot g_i^{i-1} \cdot \prod_{j \neq i} g_j^j$$

and we know that $\gcd(f,f') = g_2 \cdot g_3^2 \cdots g_k^{k-1}$. It follows that

$$\begin{aligned} c_1 &= \sum_{i=1}^{k} i \cdot g_i' \cdot \prod_{j\neq i} g_j - \sum_{i=1}^{k} g_i' \cdot \prod_{j\neq i} g_j \\ &= g_1 \cdot \sum_{i=1}^{k} (i-1) \cdot g_i' \cdot \prod_{\substack{j\geq 2 \\ j\neq i}} g_j. \end{aligned}$$

Consequently $g_1 = \gcd(b_1, c_1)$. Now we define the remaining terms of the two sequences by a recursive formulas:

$$b_{i+1} := \frac{b_i}{g_i} \qquad \text{and} \qquad c_{i+1} := \frac{c_i}{g_i} - b_{i+1}'.$$

A similar computation to the above one shows that for $i \geq 1$ we have

$$b_i = g_i \cdots g_k \qquad \text{and} \qquad c_i = g_i \cdot \sum_{j=i+1}^{k} (j-i) \cdot g_j' \cdot \prod_{\substack{l\geq i+1 \\ l\neq j}} g_l.$$

Therefore the square-free factors of f are $g_i = \gcd(b_i, c_i$ for $i \leq 1$. All in all, Yun algorithm takes a form:

Algorithm 5.3.7 (Yun, 1976)**.** *Given a monic polynomial f with coefficients in a field K of characteristic 0, this algorithm returns the square-free factors $(g_1, \ldots, g_k)$ of f.*

1. *Initialize $i := 1$ and set*

$$b_1 := {}^{f}\!/_{\gcd(f,f')}, \qquad c_1 := {}^{f'}\!/_{\gcd(f,f')} - b_1';$$

2. *as long as $b_i \neq 1$ repeat the following steps:*
   a) *compute the i-th square-free factor by the formula*

$$g_i := \gcd(b_i, c_i);$$

   b) *compute the next elements of the auxiliary sequences setting:*

$$b_{i+1} := \frac{b_i}{g_i}, \qquad \textit{and next} \qquad c_{i+1} := \frac{c_i}{g_i} - b_{i+1}';$$

   c) *increment the counter i;*
3. *return the list $(g_1, \ldots, g_i)$.*

The asymptotic complexity of Yun algorithm is again $O\big(dM(d)\lg(d)\big)$. Nevertheless its average running time outperforms Musser algorithm. This is due to the fact that the gcd in step 2a is computed for lower degree polynomials. For an in-depth comparison of running times of the three algorithms described above we refer the reader to [92].

Example 62. Take again the polynomial

$$f = x^8 - 14x^7 + 82x^6 - 264x^5 + 513x^4 - 618x^3 + 452x^2 - 184x + 32$$

The initial terms of the auxiliary sequences $(b_i)_{1\le i\le k}$ and $(c_i)_{1\le i\le k}$ are

$$b_1 = \frac{f}{\gcd(f,f')} = x^3 - 7x^2 + 14x - 8, \qquad c_1 = \frac{f'}{\gcd(f,f')} - b_1' = 5x^2 - 28x + 32.$$

Computing their greatest common divisor one obtains the first square-free factor of f:

$$g_1 = \gcd(b_1, c_1) = x - 4.$$

We then successively compute:

$$\begin{aligned}
b_2 &= \frac{b_1}{g_1} = \frac{x^3 - 7x^2 + 14x - 8}{x-4} = x^2 - 3x + 2\\
c_2 &= \frac{c_1}{g_1} - b_2' = \frac{5x^2 - 28x + 32}{x-4} - (2x-3) = 3x - 5\\
g_2 &= \gcd(b_2, c_2) = \gcd(x^2 - 3x + 2, 3x - 5) = 1\\
b_3 &= \frac{b_2}{g_2} = \frac{x^2 - 3x + 2}{1} = x^2 - 3x + 2\\
c_3 &= \frac{c_2}{g_2} - b_3' = \frac{3x-5}{1} - (2x-3) = x - 2\\
g_3 &= \gcd(b_3, c_3) = \gcd(x^2 - 3x + 2, x - 2) = x - 2\\
b_4 &= \frac{b_3}{g_3} = \frac{x^2 - 3x + 2}{x-2} = x - 1\\
c_4 &= \frac{c_3}{g_3} - b_4' = \frac{x-2}{x-2} - (1) = 0\\
g_4 &= \gcd(b_4, c_4) = \gcd(x-1, 0) = x - 1\\
b_5 &= \frac{b_4}{g_4} = \frac{x-1}{x-1} = 1\\
c_5 &= \frac{c_4}{g_4} - b_5' = \frac{0}{x-1} - (0) = 0
\end{aligned}$$

Thus $g_1 g_2^2 g_4^4 = (x-4)\cdot(x-2)^3\cdot(x-1)^4$ is the sought square-free decomposition of f.

Guersenzvaig–Szechtman algorithm

We now present yet another approach to square-free decomposition, which is due to N. Guersenzvaig and F. Szechtman. Again consider a nonzero polynomial over a field K

of characteristic zero. Since we are working over a field, without loss of generality we may assume that f is monic. Let $r := \operatorname{rad} f = {}^{f}/_{\gcd(f,f')}$ be the radical of f. It is clearly monic, as well. Consider in addition a polynomial

$$p := \frac{f'}{\gcd(f,f')}.$$

Further, denote by $\mathscr{Z} := \{\xi \in K^{\mathrm{alg}} \mid f(\xi) = 0\}$ the set of all the roots of f. These are also the roots of its radical r. For every root ξ of f, denote its multiplicity by $m(\xi)$, so that

$$f = \prod_{\xi\in\mathscr{Z}} (x-\xi)^{m(\xi)} \qquad \text{and} \qquad r = \prod_{\xi\in\mathscr{Z}} (x-\xi). \tag{5.6}$$

The polynomial r is square-free, hence its degree equals the number of its roots, that is the cardinality of the set $\mathscr{Z}$. Denote this number by s, i.e. $s = \deg r = \sharp\mathscr{Z}$.

Lemma 5.3.8. *With the above notation, for every root $\xi \in \mathscr{Z}$ one has*

$$p(\xi) = m(\xi)\cdot r'(\xi).$$

Proof. Let us first compute the derivative of r. Using Eq. (5.6) write

$$r' = \sum_{\xi\in\mathscr{Z}} \prod_{\zeta\neq\xi} (x-\zeta).$$

It follows that for every root ξ of f we have

$$r'(\xi) = \prod_{\zeta\neq\xi} (\xi-\zeta). \tag{5.7}$$

Now expand the derivative of f:

$$f' = \sum_{\xi\in\mathscr{Z}} \Big(m(\xi)\cdot(x-\xi)^{m(\xi)-1} \cdot \prod_{\substack{\zeta\in\mathscr{Z}\\ \zeta\neq\xi}} (x-\zeta)^{m(\zeta)} \Big). \tag{5.8}$$

Once we substitute the corresponding expressions derived in Eqs. (5.6) and (5.8) for f an f', the formula for p evaluates to

$$p = \frac{f'}{\gcd(f,f')} = \frac{r\cdot f'}{f} = \frac{\displaystyle\sum_{\xi\in\mathscr{Z}} \Big(m(\xi)\cdot(x-\xi)^{m(\xi)-1} \cdot \prod_{\zeta\neq\xi} (x-\zeta)^{m(\zeta)} \Big)}{\displaystyle\prod_{\xi\in\mathscr{Z}} (x-\xi)^{m(\xi)-1}} = \sum_{\xi\in\mathscr{Z}} \Big(m(\xi)\cdot \prod_{\zeta\neq\xi} (x-\zeta) \Big).$$

Consequently, evaluating p at a root ξ, we have

$$p(\xi) = m(\xi)\cdot \prod_{\zeta\neq\xi} (\xi-\zeta)$$

and the thesis follows from Eq. (5.7). □

Lemma 5.3.9. *There are polynomials g and h such that*

$$r'g + rh = 1 \qquad \text{and} \qquad \deg g < s.$$

Proof. The polynomial r is square-free, hence relatively prime to its derivative. Bézout's identity asserts that there are polynomials α, β such that $\alpha r' + \beta r = \gcd(r, r') = 1$. If $\deg \alpha < \deg r$, then $g := \alpha$ and $h := \beta$ satisfy the thesis. Otherwise, let g be the remainder of α modulo r, thus $\deg g < \deg r = s$ and $\alpha = qr + g$ for some $q \in K[x]$. Take $h := r'g + \beta$. We have

$$r'g + rh = r' \cdot (\alpha - qr) + r \cdot (r'q + \beta) = r' \cdot \alpha + r \cdot \beta = 1. \qquad \square$$

Let g be the polynomial of from the previous lemma. Define a polynomial $\mu \in K[x]$ to be the remainder modulo r of a product $p \cdot g$:

$$\mu := \big(p \cdot g \bmod r\big). \tag{5.9}$$

Proposition 5.3.10. *For every root ξ of f, the value $\mu(\xi)$ equals the multiplicity $m(\xi)$ of ξ.*

Proof. Theorem 4.1.2 asserts that there is a unique polynomial of degree $\leq s-1$ that take a value $m(\xi)$ at every root $\xi \in \mathscr{Z}$. Slightly abusing the notation we shall denote it by m, as well. We must show that $m = \mu$. Since $\deg \mu < \deg r = s$, it suffices to show that the two polynomials agree at s distinct points, namely the roots of f (hence also r). Take any root $\xi \in \mathscr{Z}$, then by Lemma 5.3.8 we may write

$$\mu(\xi) = p(\xi) \cdot g(\xi) = m(\xi) \cdot r'(\xi) \cdot g(\xi) = m(\xi) \cdot (1 - r \cdot h)(\xi) = m(\xi).$$

This shows that these two polynomials are identical. $\square$

Example 63. Consider the following polynomial

$$\begin{aligned} f &= (x-4) \cdot (x-2)^3 \cdot (x-1)^4 \\ &= x^8 - 14x^7 + 82x^6 - 264x^5 + 513x^4 - 618x^3 + 452x^2 - 184x + 32, \end{aligned}$$

then:

$$\begin{aligned} r &= \operatorname{rad} f = (x-4) \cdot (x-2) \cdot (x-1) = x^3 - 7x^2 + 14x - 8 \\ r' &= 3x^2 - 14x + 14 \\ p &= 8x^2 - 42x + 46. \end{aligned}$$

We need the polynomials g and h of Lemma 5.3.9. The extended Euclidean algorithm (Algorithm 1.3.6) yields

$$g = \frac{7}{18}x^2 - 2x + \frac{35}{18} \qquad \text{and} \qquad h = -\frac{7}{6}x + \frac{59}{18}.$$

Therefore the polynomial μ has a form

$$\mu = \big(8x^2 - 42x + 46\big) \cdot \Big(\frac{7}{18}x^2 - 2x + \frac{35}{18}\Big) \bmod \big(x^3 - 7x^2 + 14x - 8\big) = -x + 5.$$

It is now easy to derive the square-free decomposition of f. The key is the following result.

Proposition 5.3.11. *Let $f = g_1 g_2^2 \cdots g_k^k$ be the square-free decomposition of f. Then for every $j \in \{1, \ldots, k\}$*

$$g_j = \gcd(\mu - j, r),$$

where $r = \operatorname{rad} f$ is the radical of f and μ is the polynomial defined in Eq. (5.9).

Proof. Proposition 5.3.10 asserts that the value of μ at a root of f equals the multiplicity of this root. It follows that $\xi \in \mathscr{Z}$ has multiplicity $j \leq k$ if and only if $\mu(\xi) = k$ if and only if ξ is a root of $\mu - k$.

The square-free factor g_j of f is the product of $(x - \xi)$ for all roots ξ of f (hence also of $r = \operatorname{rad} f$) of multiplicity j. Thus it is the product of all $(x - \xi)$ running over all common roots of r and $\mu - j$. Therefore g_j is the greatest common divisor of r and $\mu - j$, as desired. □

Example 64. We continue the previous example. Compute

$$\begin{aligned}
g_1 &= \gcd(\mu - 1, r) = \gcd(-x + 4, x^3 - 7x^2 + 14x - 8) = x - 4 \\
g_2 &= \gcd(\mu - 2, r) = \gcd(-x + 3, x^3 - 7x^2 + 14x - 8) = 1 \\
g_3 &= \gcd(\mu - 3, r) = \gcd(-x + 2, x^3 - 7x^2 + 14x - 8) = x - 2 \\
g_4 &= \gcd(\mu - 4, r) = \gcd(-x + 1, x^3 - 7x^2 + 14x - 8) = x - 1
\end{aligned}$$

Now, $1 \cdot \deg g_1 + 2 \cdot \deg g_2 + 4 \cdot \deg g_4 = 8 = \deg f$, therefore $g_1 g_2^2 g_4^4 = (x-4) \cdot (x-2)^3 \cdot (x-1)^4$ is the square-free decomposition of f.

Let us now write the pseudo-code of the square-free decomposition method we have just described.

Algorithm 5.3.12 (Guersenzvaig–Szechtman, 2012/2017)**.** *Given a monic polynomial f over a field K of characteristic 0, this algorithm computes the square-free factors of f.*

1. *Construct polynomials*

$$p := \frac{f'}{\gcd(f, f')}, \qquad r := \operatorname{rad} f = \frac{f}{\gcd(f, f')};$$

2. *call the extended Euclidean algorithm with input: r' and r, to find polynomials $g, h \in K[x]$ such that*

$$r' \cdot g + r \cdot h = 1, \qquad \deg g < \deg r \quad \text{and} \quad \deg h < \deg r';$$

3. *compute the polynomial μ as the remainder modulo r of $p \cdot g$;*

4. *initialize* $k := 1$ *and* $n := 0$*;*
5. *for* $k \in \{1, 2, \dots\}$ *and as long as* $\sum_{j=1}^{k-1}(j \cdot \deg g_j) < \deg f$*, compute the* k*-th square-free factor of* f *by the formula*

$$g_k := \gcd(\mu - k, r);$$

6. *return* $(g_1, \dots, g_k)$*.*

Time complexity analysis. Recall that $s = \deg r$ and let $d := \deg f$. It is clear that $s = O(d)$. If one uses fast gcd, then steps (1)–(3) of the algorithm need $O\big(M(d)\lg d\big)$ arithmetic operations. The same applies to every iteration of the loop in step (5). The number of iterations equals $k = O(d)$, the index of the last non-trivial square-free factor of f. All in all, the time complexity (in terms of arithmetic operations in the field of coefficients) of Algorithm 5.3.12 is $O\big(d \cdot M(d) \cdot \lg(d)\big)$, the same as of the three previously described algorithms.

The following exercise presents an alternative method of constructing the polynomial μ. In fact this is the original construction from [31].

Exercise 10. Let C_r be the companion matrix of the polynomial $r = \operatorname{rad} f$. Further, let V_μ and V_g be column vectors of coefficients of μ and g, zero-padded to length $s = \deg r$, if needed. Prove that $V_\mu = C_r \cdot V_g$.

5.4 Applications of square-free decomposition

An immediate application of the square-free decomposition is an algorithmic test whether there is a polynomial g such that a given polynomial f is its n-th power. Let $f \in K[x]$ be a polynomial and

$$f = c \cdot g_1 g_2^2 \cdots g_k^k$$

be its square-free decomposition. Since all g_i's are monic, square-free and pairwise co-prime, hence, it is clear that f is a n-th power of some polynomial g if and only if c is a n-th power in K and $g_i \neq 1$ only for those i that are divisible by n. If this is the case, then

$$g = c^{1/n} \cdot g_n g_{2n}^2 \cdots g_{mn}^m,$$

where $mn \leq k < (m+1)n$.

Algorithm 5.4.1. *Assume that* K *is a field of characteristic* 0 *and such that* n*-th roots of elements of* K *are computable. Given a polynomial* $f \in K[x]$ *and a positive integer* $n \in \mathbb{N}$*, this algorithm returns a polynomial* g *satisfying* $g^n = f$ *or false if such a polynomial* g *does not exist.*

1. *Let* $c := \operatorname{lc}(f)$ *be the leading coefficient of* f*;*
2. *if* $c \notin K^n$*, then return false and quit; otherwise set* $d := c^{1/n}$*;*
3. *compute the square-free decomposition of* f*:*

$$f = c \cdot g_1 g_2^2 \cdots g_k^k;$$

4. *if there is* $1 \leq i \leq k$ *such that* $n \nmid i$ *and* $g_i \neq 1$*, then return false and quit;*
5. *return* $d \cdot g_n g_{2n}^2 \cdots g_{mn}^m$.

Partial fraction decomposition

A partial fraction decomposition is a technique of transforming a rational expression with a compound denominator into a sum of fractions, with denominators being factors of the original denominator. Suppose, that we are given a rational number or a rational function (or more generally an element of a fraction field of some Euclidean ring) f/g and the denominator is a product $g = a \cdot b$ of two relatively prime elements (integers, polynomials,...). By Bézout's identity (see Theorem 1.3.4), there are such elements α, β that $1 = \gcd(a, b) = \alpha a + \beta b$. Hence, we may decompose our fraction into a sum

$$\frac{f}{g} = \frac{f \cdot 1}{a \cdot b} = \frac{f \cdot (\alpha a + \beta b)}{a \cdot b} = \frac{f\beta}{a} + \frac{f\alpha}{b}.$$

This is called a *partial fraction decomposition* (PFD) of f/g. Of course, if either a or b can again be factored, then we may continue the process and decompose the fraction even further.

Let us now apply the above technique to a square-free decomposition of a the denominator of a given rational function $f/g \in K(x)$. A prori, the numerator may have a higher degree than the denominator (i.e. the fraction may be improper). In such a case, we may extract the "integral part" with a help of polynomial division. Let $f = f_0 \cdot g + r_0$, with $\deg r_0 < \deg g$, then we write

$$\frac{f}{g} = f_0 + \frac{r_0}{g}.$$

Denote by $c = \mathrm{lc}(g)$ the leading coefficient of g, so that $b_0 := 1/c \cdot g$ is a monic polynomial and let $a_0 := 1/c \cdot r_0$. Compute the square-free decomposition of the denominator $g = c \cdot g_1 \cdot g_2^2 \cdots g_k^k$ and write

$$\frac{f}{g} = f_0 + \frac{a_0}{b_0} = f_0 + \frac{a_0}{g_1 \cdot (g_2^2 \cdots g_k^k)}.$$

Set $b_1 := g_2^2 \cdots g_k^k = b_0/g_1$. Since all g_i's are pairwise co-prime, therefore $\gcd(b_1, g_1) = 1$ and as above we find such polynomials $\alpha, \beta \in K[x]$ that

$$1 = \alpha \cdot b_1 + \beta \cdot g_1$$

and $\deg \alpha < \deg g_1$, $\deg \beta < \deg b_1$. It follows that

$$\frac{f}{g} = f_0 + \frac{a_0}{g_1 b_1} = f_0 + \frac{a_0 \alpha}{g_1} + \frac{a_0 \beta}{b_1}.$$

Unfortunately, the two fractions obtained above may turn out to be improper. Perform polynomial division again in order to compute the quotients and remainders. Let

$$a_0\alpha = \hat{q}_1 \cdot g_1 + f_1 \qquad \text{and} \qquad a_0\beta = q_1 b_1 + a_1,$$

where $\deg f_1 < \deg g_1$ and $\deg a_1 < \deg b_1$. Write

$$\frac{f}{g} = f_0 + \frac{f_1}{g_1} + \frac{a_1}{b_1} + \big(\hat{q}_1 + q_1\big).$$

We claim that the last term is null. Indeed, we have

$$a_0 = a_0\alpha b_1 + a_0\beta g_1 = (\hat{q}_1 + q_1)g_1 b_1 + (f_1 b_1 + a_1 g_1),$$

but $\deg a_0 < \deg g = \deg(g_1 b_1)$. Hence, $\hat{q}_1 + q_1 = 0$, as claimed. Consequently

$$\frac{f}{g} = f_0 + \frac{f_1}{g_1} + \frac{a_1}{b_1}$$

and we can continue the process decomposing the last term. This way we have proved:

Observation 5.4.2. *Let $f/g \in K(x)$ be a rational function with the square-free decomposition of the denominator $g = c \cdot g_1 g_2^2 \cdots g_k^k$. Then there are unique polynomials $f_0, f_1, \ldots, f_k \in K[x]$ such that*

$$\frac{f}{g} = f_0 + \frac{f_1}{g_1} + \frac{f_2}{g_2^2} + \cdots + \frac{f_k}{g_k^k} \tag{5.10}$$

and $\deg f_i < i \cdot \deg g_i$ for $1 \le i \le k$.

Equation (5.10) is called the *incomplete square-free partial fraction decomposition* (ISPFD) of f/g. We may formalize the described procedure as:

Algorithm 5.4.3. *Given a rational function f/g over a field K of characteristic zero, with $f, g \in K[x]$ relatively prime, this algorithm returns the incomplete square-free partial fraction decomposition of f/g.*

1. *Using Algorithm 1.1.12 compute the quotient q and the remainder r of f modulo g and set $f_0 := q$;*
2. *let $c := \mathrm{lc}(g)$ be the leading coefficient of the denominator;*
3. *compute the square-free decomposition of the denominator:*

$$g = c \cdot g_1 g_2^2 \cdots g_k^k;$$

4. *set $a_0 := 1/c \cdot r$ and $b_0 := 1/c \cdot g$;*
5. *for $i = 1, \ldots, k-1$ proceed as follows:*
   a) *set $b_i := b_{i-1}/g_i^i$;*

b) using the extended Euclidean algorithm (Algorithm 1.3.6) compute polynomials $\alpha, \beta \in K[x]$ *such that*

$$1 = \gcd(b_i, g_i^i) = \alpha b_i + \beta g_i^i$$

and $\deg \alpha < \deg g_i^i$, $\deg \beta < \deg b_i$;

c) compute the next numerator of the decomposition, setting

$$f_i := (a_{i-1} \cdot \alpha) \bmod g_i^i;$$

d) compute the numerator of the part that remains to be decomposed, setting $a_i := a_{i-1} \cdot \beta \bmod b_i$;

6. *the last, remaining numerator is* $f_k := a_{k-1}$;
7. *return* $f_0, {}^{f_1}/_{g_1}, {}^{f_2}/_{g_2^2}, \ldots, {}^{f_k}/_{g_k^k}$.

Example 65. Take a rational function

$$\frac{f}{g} = \frac{703 + 1533x + 1324x^2 + 697x^3 + 361x^4 + 190x^5 + 68x^6 + 13x^7 + x^8}{108 + 324x + 387x^2 + 238x^3 + 80x^4 + 14x^5 + x^6}.$$

The square-free decomposition of the denominator is

$$g = (1+x) \cdot (2+x)^2 \cdot (3+x)^3$$

and the application of the above algorithm yields

$$\begin{aligned}
\frac{f}{g} &= \frac{703 + 1533x + 1324x^2 + 697x^3 + 361x^4 + 190x^5 + 68x^6 + 13x^7 + x^8}{108 + 324x + 387x^2 + 238x^3 + 80x^4 + 14x^5 + x^6} \\
&= (2 - x + x^2) + \frac{487 + 993x + 766x^2 + 284x^3 + 52x^4 + 4x^5}{(1+x) \cdot (2+x)^2 \cdot (3+x)^3} \\
&= (2 - x + x^2) + \frac{3}{1+x} + \frac{163 + 182x + 71x^2 + 12x^3 + x^4}{(2+x)^2 \cdot (3+x)^3} \\
&= (2 - x + x^2) + \frac{3}{(1+x)} + \frac{5+x}{(2+x)^2} + \frac{7-2x}{(3+x)^3}.
\end{aligned} \tag{5.11}$$

Complete square-free partial fraction decomposition

Eq. 5.10 is called an incomplete decomposition, because it is possible to further split a given rational function ${}^f/_g$ writing

$$\frac{f}{g} = f_0 + \sum_{i=1}^{k} \sum_{j=1}^{i} \frac{f_{ij}}{g_i^j} \qquad \text{with } \deg f_{ij} < \deg g_i.$$

This is called the (*complete*) *square-free partial fraction decomposition* (SPFD) of the rational function ${}^f/_g$. We leave it to the reader to prove the uniqueness of this expression.

Suppose that f_i/g_i^i is one of the summands of ISPFD. Compute the quotient and remainder of a polynomial division:

$$f_i = q \cdot g_i + r, \qquad \text{where } \deg r < \deg g_i.$$

Denote the remainder $f_{ii} := r$. Substituting it to our fraction, we have

$$\frac{f_i}{g_i^i} = \frac{q}{g_i^{i-1}} + \frac{f_{ii}}{g_i^i}.$$

Iterating, we compute the complete decomposition

$$\frac{f_i}{g_i^i} = \frac{f_{i1}}{g_i} + \frac{f_{i2}}{g_i^2} + \cdots + \frac{f_{ii}}{g_i^i}.$$

Formalizing this procedure, we write an algorithm for SPFD.

Algorithm 5.4.4. *Given a rational function f/g over a field K of characteristic zero, with $f, g \in K[x]$ relatively prime, this algorithm returns the complete square-free partial fraction decomposition of f/g.*

1. *Using Algorithm 5.4.3, compute ISPFD of f/g:*

$$\frac{f}{g} = f_0 + \frac{f_1}{g_1} + \frac{f_2}{g_2^2} + \cdots + \frac{f_k}{g_k^k};$$

2. *iterate other the summands and for every $i = 2, 3, \ldots, k$:*
   a) *initialize $j := i$ and $a := f_i$;*
   b) *as long as $j > 1$ repeat the following steps:*
      i. *using Algorithm 1.3.3 compute q and r such that*

$$a = q \cdot g_i + r;$$

      ii. *set $f_{ij} := r$ and update $a := q$;*
      iii. *decrement j;*
3. *return the complete decomposition $f/g = f_0 + \sum_{i=1}^{k} \sum_{j=1}^{i} f_{ij}/g_i^j$.*

Example 66 (continued). Take the rational function f/g from the previous example. We have already computed its ISPFD in Eq. (5.11). Now, the above algorithm returns

$$(2 - x + x^2) + \frac{3}{(1+x)} + \frac{1}{(2+x)} + \frac{3}{(2+x)^2} + \frac{-2}{(3+x)^2} + \frac{13}{(3+x)^3}$$

which is the complete decomposition of f/g.

A basic application of the square-free partial fraction decomposition is an algorithm for symbolic integration of rational functions, which we describe in Section 7.1 of Chapter 7.

5.5 Polynomial factorization over finite fields

In this section we discuss the problem of factoring a polynomial defined over a finite field. Given an arbitrary monic polynomial $f \in \mathbb{F}_q[x]$, on way to achieve its complete factorization is by a three stage process (see Figure). First, find its square-free factors $g_1, \ldots, g_k$ so that $f = g_1 \cdot g_2^2 \cdots g_k^k$. This step has been already covered in Section 5.3. From now on we work just with square-free polynomials. In the second step, we factor a (square-free) polynomial $g = g_j$ into into a product of factors of like degrees $g = h_1 \cdots h_s$. The final step is to factor each $h = h_j$ into the irreducibles. Since, the first step has already been discussed, in the subsequent sections we investigate the remaining two steps.

missing

Distinct-degree factorization

Let g be a monic square-free polynomial with coefficients in some finite field $\mathbb{F}_q$. Here $q = p^m$ is a prime power, with p being the characteristic of the field. Since the ring of polynomials $\mathbb{F}_q[x]$ is a unique factorization domain, we may (uniquely) write g as a product $g = p_1 \cdots p_m$ of monic irreducible polynomials. Group the factors of equal degree, setting

$$h_k := \prod_{\deg p_i = k} p_i \qquad \text{for } k \le s := \max\{\deg p_i \mid i \le m\}.$$

Then g is a product of $h_1, \ldots, h_s$.

Definition 5.5.1. The *distinct-degree factorization* of a (non-constant) monic, square-free polynomial $g \in \mathbb{F}_q[x]$ is an expression

$$g = h_1 \cdots h_s,$$

where $h_s \neq 1$ and each h_k is a product of all the (monic) irreducible factors of g of degree k. The polynomials $h_1, \ldots, h_s$ are called the *distinct-degree factors* of g.

The key to the distinct-degree factorization is provided by the following theorem.

Theorem 5.5.2. *Fix an integer $d \ge 1$ and let $p_1, \ldots, p_t \in \mathbb{F}_q[x]$ be all the monic, irreducible polynomials with coefficients in $\mathbb{F}_q$ such that $\deg p_i$ divides d. Then*

$$p_1 \cdots p_t = x^{q^d} - x.$$

Proof. Take a finite filed extension $\mathbb{F}_{q^d}/\mathbb{F}_q$ of degree d. Corollary 2.5.4 asserts that the polynomial $h := x^{q^d} - x$ splits into linear factors over $\mathbb{F}_{q^d}$:

$$h = \prod_{a \in \mathbb{F}_{q^d}} (x - a).$$

It is clear that h is a square-free polynomial, as it has $q^d = \deg h$ distinct roots.

Take now any monic irreducible polynomial $p \in \mathbb{F}_q[x]$ of degree n dividing d. Take a field $K := \mathbb{F}_q(\alpha)$, where α is any root of p. Then $K \cong {}^{\mathbb{F}_q[x]}/_{\langle p \rangle}$ and so it has q^n elements. Up to an isomorphism, K is a subfield of $\mathbb{F}_{q^n}$, hence $h(\alpha) = 0$ and in particular $x - \alpha$ divides h. It follows that $x - \alpha$ is a nontrivial divisor (in the polynomial ring $K[x]$) of $\gcd(h, p)$. In particular $\gcd(h, p) \neq 1$. On the other hand, both h and p are elements of $\mathbb{F}_q[x]$ and so is their greatest common divisor. It follows that $\gcd(h, p) \in \mathbb{F}_q[x]$ is a nontrivial divisor of the irreducible polynomial p. This may happen only if $\gcd(h, p) = p$, which means that p divides h.

Conversely assume that $p \in \mathbb{F}_q[x]$ is a monic, irreducible divisor of h. Since h splits into linear terms in $\mathbb{F}_{q^d}$, the polynomial p must have a root $\alpha \in \mathbb{F}_{q^d}$. Extend $\mathbb{F}_q$ by α. Set $K := \mathbb{F}_q(\alpha)$. The field K has $q^{\deg p}$ elements. On the other hand, K is a subfield of $\mathbb{F}_{q^d}$, thus $q^d = \sharp\mathbb{F}_{q^d} = (\sharp K)^e$, where e is the degree of the field extension $\mathbb{F}_{q^d}/K$. It follows that $\deg p$ divides d.

All in all, we have proved that h is a square-free polynomial and its irreducible factors are exactly the irreducible polynomials whose degrees divide d. Consequently h is the product of all these polynomials. □

Let $g = h_1 \cdots h_s$ be the distinct degree factorization of g. It follows immediately from the previous theorem that

$$h_1 \cdots h_k = \gcd\bigl(g, x^{q^k} - x\bigr).$$

Thus, one may compute the distinct degree factorization of g by just calculating successive greatest common divisors. The product of linear factors of g is given by the formula $h_1 = \gcd(g, x^q - x)$. The product of irreducible quadratic factors could be computed as

$$h_2 = \frac{\gcd\bigl(g_2, x^{q^2} - x\bigr)}{h_1}.$$

Nevertheless, it is more convenient to get rid of the linear terms from g, setting $g_2 := {}^{g}/_{h_1}$ and then

$$h_2 = \gcd\bigl(g_2, x^{q^2} - x\bigr).$$

Continuing this process we have

$$h_k = \gcd\bigl(g_k, x^{q^k} - x\bigr),$$

where $g_k := {}^{g_{k-1}}/_{h_{k-1}}$. For the sake of completeness we set also $g_1 := g$. The computation of the greatest common divisors can be farther simplified taking the reminders (recall the first step of Euclidean algorithm) so that

$$h_k = \gcd\bigl(g_k, (x^{q^k} - x \bmod g_k)\bigr).$$

Now, if we define an auxiliary sequence $f_1, \ldots, f_s$ by a recursive formula $f_1 := (x^q \bmod g_1)$ and $f_k := (f_{k-1}^q \bmod g_{k-1})$ for $k > 1$, then $f_k - x \equiv x^{q^k} - x \pmod{g_k}$, hence the previous formula may be replaced by

$$h_k = \gcd\bigl(g_k, f_k - x\bigr).$$

Summarizing our discussion, we have just proved the correctness of the following algorithm.

Algorithm 5.5.3. *Given a monic square-free polynomial $g \in \mathbb{F}_q[x]$, this algorithm computes its distinct degree factors $h_1, \ldots, h_s$.*

1. *Initialize $f_0 := x$, $g_1 := g$ and $k := 1$;*
2. *as long as g_k is non-constant proceed as follows:*
   a) *compute $f_k := \left(f_{k-1}^q \bmod g_k\right)$;*
   b) *compute the next distinct-degree factor of g:*
   $$h_k := \gcd\left(g_k, f_k - x\right);$$
   c) *increment k and set $g_k := \frac{g_{k-1}}{h_{k-1}}$;*
3. *return $h_1, \ldots, h_k$.*

Example 67. Take a polynomial $g = x^8 + 6x^7 + 5x^6 + x^5 + 3x^4 + 4x^3 + 5x^2 + 6x + 4$ with coefficients in $\mathbb{F}_7$. Let us see how the above algorithm factors g. Initial values of the auxiliary polynomials are: $f_0 = 1$ and $g_1 = g$. In order to find the product of all the linear factors of g, we construct $f_1 := (f_0^7 \bmod g_1) = x^7$ and compute the greatest common divisor of $f_1 - x$ and g_1:

$$h_1 := \gcd(g_1, f_1 - x) = x^3 + 5x^2 + 2x + 6.$$

Consequently $g_2 := {}^{g}/_{h_1} = x^5 + x^4 + 5x^3 + 3x^2 + 3$ is not divisible by any linear polynomial. It is also clear now that g has 3 linear factors (hence also roots).

Now, we look for irreducible quadratic polynomials dividing g. To this end we take

$$f_2 := (f_1^7 \bmod g_2) = 3x^4 + 3x^3 + 2x^2 + x.$$

The product of irreducible quadratic divisors of g is then

$$h_2 := \gcd(g_2, f_2 - x) = x^2 + x + 3.$$

Since the degree of h_2 is 2, it must be an irreducible polynomial. Compute $g_3 := {}^{g_2}/_{h_2} = x^3 + 2x + 1$. It is a cubic polynomial, hence it is the last sought irreducible factor of g. It follows that the distinct-degree factorization of g is

$$g = (x^3 + 5x^2 + 2x + 6) \cdot (x^2 + x + 3) \cdot (x^3 + 2x + 1).$$

If we are interested just in finding the roots of a polynomial g, then it suffices to construct only h_1, the first distinct degree factor of g. The next observation follows immediately from Theorem 5.5.2 and the discussion preceding Algorithm 5.5.3. We note it down to simplify further references.

Observation 5.5.4. *Let $g \in \mathbb{F}_q[x]$ be a monic polynomial (not necessarily square-free). If $\xi_1, \ldots, \xi_k \in \mathbb{F}_q$ are all the roots of g in $\mathbb{F}_q$, then*

$$\gcd\left(g, x^q - x\right) = (x - \xi_1) \cdots (x - \xi_k).$$

Equal degree factorization

Once we performed the square-free decomposition and distinct degree factorization, we are left with monic square-free polynomials such that each one is a product of irreducible polynomials of a fixed degree. Our last task is to recover these irreducibles. This process is called the *equal degree factorization*

Fix a finite filed $\mathbb{F}_q$. We will restrict ourselves to an odd characteristic only, hence q is a power of some odd prime. Take a monic, square-free polynomial $f \in \mathbb{F}_q[x]$ of degree n and such that every irreducible factor of f has degree d. In other word $f = p_1 \cdots p_m$, where $\deg p_i = d$ for every $1 \leq i \leq m$ and $\deg f = n = d \cdot m$. Denote the quotient ring $\mathbb{F}_q[x]/\langle f \rangle$ by R and for every $1 \leq i \leq m$ let $K_i := \mathbb{F}_q[x]/\langle p_i \rangle$. Thus, K_i is a field extension of the base field $\mathbb{F}_q$ and $[K_i : \mathbb{F}_q] = d$. In particular $\sharp K_i = q^d$ and there is some $\xi_i \in K_i$, a root of p_i, such that $K_i = \mathbb{F}_q(\xi_i)$. Recall that a finite field of a given number of elements is unique up to an isomorphism. It follows that $K_1, \ldots, K_m$ are all isomorphic. Chinese remainder theorem (Theorem 2.2.5) asserts that there is a ring isomorphic

$$\Phi : R \xrightarrow{\sim} K_1 \times \cdots \times K_m.$$

Unfortunately, we cannot explicitly construct this isomorphism not knowing the factorization of f. These two tasks turn out to be equivalent and we are going to exploit this equivalence to factor f.

For brevity, given a polynomial $g \in \mathbb{F}_q[x]$ by $\overline{g}$ we will denote the coset of g modulo $\langle f \rangle$, that is the image of g in R. Each such coset contain a unique polynomial of degree strictly smaller than n, hence the ring R can be identified with the subset $\mathbb{F}_q[x]_{n-1} = \{g \in \mathbb{F}_q[x] \,|\, \deg g < n\}$.

Lemma 5.5.5. *Let $g \in \mathbb{F}_q[x]_{n-1}$. Then f, g are relatively prime (in the ring of polynomials) if and only if $\overline{g}$ is not a zero-divisor in R.*

Proof. Denote $a := \gcd(f, g)$ and $b := f/a$. By Bézout's identity, there polynomials $\alpha, \beta \in \mathbb{F}_q[x]$ such that

$$a = \alpha f + \beta g \qquad \text{and} \qquad \deg \alpha, \deg \beta < n.$$

Thus in the quotient ring R we have $\overline{a} = 0 + \overline{\beta} \cdot \overline{g}$.

If f, g are relatively prime, then $1 = \overline{\beta} \cdot \overline{g}$, hence $\overline{g}$ is invertible. In particular it cannot be a zero-divisor. Conversely if f, g are not relatively prime, then $b \neq f$ hence $\overline{b} \neq 0$ and we have

$$0 = \overline{a} \cdot \overline{b} = \left(\overline{\beta} \cdot \overline{b}\right) \cdot \overline{g}.$$

It follows that $\overline{g}$ is a zero divisor. □

Using the isomorphism Φ we may write

$$\Phi(\overline{g}) = \left(\overline{g}_1, \ldots, \overline{g}_m\right),$$

where each $\overline{g}_i$ is a coset of g modulo $\langle p_i \rangle$. A nonzero element $\overline{g} \in R$ is a zero-divisor if and only if at least one $\overline{g}_i$ is zero. Take a polynomial $g \in \mathbb{F}_q[x]_{n-1}$ relatively prime

to f. Hence every coordinate $\overline{g}_i$ of $\Phi(\overline{g})$ is a nonzero element of the field $K_i \cong \mathbb{F}_{q^d}$. Set $e := (q^d-1)/2$. It follows from that

missing

$$\Phi(\overline{g}^e) = (\overline{g}_1^e, \ldots, \overline{g}_m^e) = (\pm 1, \ldots, \pm 1),$$

with $\overline{g}_i^e = 1$ if and only if g_i is a square in K_i.

Lemma 5.5.6. *Let h be the remainder of $g^e - 1$ modulo f. If there are $i \neq j$ such that $\overline{g}_i$ is a square and $\overline{g}_j$ is a non-square, then $\gcd(h, f)$ is a proper, non-trivial divisor of f.*

Proof. We have $\overline{g}_i^e = 1$ and $\overline{g}_j^e = -1$. It follows that $\Phi(\overline{g}^e - 1)$ is a non-zero element with a zero at ith coordinate. Thus, $\overline{g}^e - 1$ is a zero-divisor in R and the assertion follows from the previous lemma. □

Now, the idea of Cantor–Zassenhaus algorithm of equal degree factorization is to pick at random a non-zero polynomial $g \in \mathbb{F}_q[x]$ of degree $\deg g < n$ and check if it gives raise to a non-trivial divisor of f as follows:

1. First check if $h := \gcd(g, f)$ is a non-trivial, proper divisor of f (i.e. if $h \neq 1$);
2. if not set $h := \gcd\big(g^e - 1 \bmod f, f\big)$ and check if this is a non-trivial, proper divisor.

Theorem 5.5.7. *Let g be a random polynomial of degree $\deg g < n$. Assuming the uniform distribution, probability that the above above procedure fails to produce a non-trivial divisor of f is less than $1/2$.*

Proof. There are $q^n - 1$ non-zero polynomials with coefficients in $\mathbb{F}_q$ of degree smaller than n, including the constant ones. The procedure does not return a divisor of f if and only if either all the coordinates of $\Phi(\overline{g})$ are squares or all are non-squares. Thus the number of polynomials for which the procedure fails is precisely

$$2 \cdot \Big(\frac{q^d - 1}{2}\Big)^m.$$

The sought probability is thus

$$\frac{(q^d-1)^m}{2^{m-1} \cdot (q^{dm} - 1)} < \frac{1}{2}$$

□

Exercise 11. Depending on the parameters: q (the cardinality of the base field) and m (the number of irreducible divisors) estimate which of the two steps in the above procedure is more likely to produce a proper divisor of f.

Theorem 5.5.7 implies that repeating the procedure described above with different random polynomials, the probability that we fail to factor f diminishes exponentially with each iteration. This observation gives raise to a probabilistic algorithm algorithm for equal degree factorization of f is now obvious.

Algorithm 5.5.8 (Cantor–Zassenhaus, 1981). *Let $f \in \mathbb{F}_q[x]$ be a monic square-free polynomial of degree $n = m \cdot d$ with $m \geq 2$ distinct irreducible factors, each one of degree d. This algorithm computes all the monic irreducible factors of f.*

1. *Denote $e := (q^d - 1)/2$ and initialize an empty list D of irreducible divisors of f;*
2. *push f onto a stack of remaining factors;*
3. *repeat the following steps as long as the stack is non-empty:*
   a) *pop a polynomial g from the top of the stack;*
   b) *if $\deg g = d$, then g is one of the irreducible factors, hence append it to the list D and reiterate the loop;*
   c) *pick at random a monic polynomial $h \in \mathbb{F}_q[x]_{n-1}$;*
   d) *if $\gcd(h, g) \neq 1$, then push h and g/h onto the stack and reiterate the loop;*
   e) *if $\gcd(h^e - 1, g) \neq 1$, then push h and g/h onto the stack and reiterate the loop;*
   f) *push g back onto the stack;*
4. *return the list D.*

Example

Shoup algorithm

Berlekamp algorithm

missing

5.6 Polynomial factorization over rationals

LLL-factorization, Henkel lifting

Bibliographic notes and further reading

Table 5.1 is based on [67, Sequence A002997], while 5.2 on [67, Sequence A014233].

The Miller–Rabin test appeared first in G. Miller's Ph.D. thesis [64] and the corresponding conference paper [63] in a conditional (under Riemann's hypothesis) version. It was subsequently converted into a randomized algorithm by M. Rabin [70].

Algorithm 5.3.5 was discovered independently by R. Tobey and E. Horowitz in their Ph.D. thesis (see [84] and [34]) respectively in 1967 and 1969. Algorithm 5.3.6 was also a part of a Ph.D. thesis [66] of its author, D. Musser. Subsection 5.3 is based on [31] with modification described in [47].

6

Univariate polynomial equations

6.1 Generalities

This chapter is fully devoted to a problem of finding roots of a given univariate polynomial $f \in K[x]$. We shall concentrate mostly on the problem of finding the real roots of f. Anyway, we begin with the following theorem, that gives the upper bound on the number of distinct complex roots.

Theorem 6.1.1 (Fundamental Theorem of Algebra)**.** *Let f be a univariate polynomial with coefficients in a subring of the complex numbers. Then f has precisely* $\deg f$ *roots in $\mathbb{C}$, when counted with multiplicities.*

This result is very well known and so we feel free to omit a proof. A thorough discussion of the theorem, together with six different proofs can be found in [26]. Recall that Observation 5.3.4 provides a method of finding a square-free polynomial with exactly the same roots as f. Therefore, when dealing with polynomials with exact coefficients, we may without loss of generality restrict our selves to the problem of finding roots of square-free polynomials. And a square-free polynomial of degree n has, by virtue of the above theorem, precisely n distinct roots in $\mathbb{C}$.

The next result is also well known. In fact, its particular case (see Corollary 6.1.3) is even taught in secondary schools (at least in Poland).

Theorem 6.1.2. *Let $f = f_0 + f_1 x + \cdots + f_n x^n$ be a polynomial with coefficients in a unique factorization domain R and let K be the field of fractions of R. If $\xi \in K$ is a root of f, then $\xi = {}^{p}/_{q}$, where p divides the constant term f_0 and q divides the leading coefficient f_n.*

Proof. Take a root $\xi \in K$ of f and express it as a reduce fraction ${}^{p}/_{q}$, with $p, q \in R$ relatively prime. The fact that ξ is a root of f means that

$$f_0 + \frac{f_1 p}{q} + \frac{f_2 p^2}{q^2} + \cdots + \frac{f_n p^n}{q^n} = 0.$$

After clearing the denominators this yields

$$f_0 q^n + f_1 p q^{n-1} + \cdots + f_{n-1} p^{n-1} q + f_n p^n = 0.$$

All the terms above, but the first one, are multiplicities of p. It follows that $f_0 q^n$ is divisible by p, as well. Now, p and q are relatively prime, hence p divides f_0. Analogously we show that q divides $f_n p^n$ and the fact that p, q are relatively prime implies that q divides f_n. □

A typical example of a UFD is the ring $\mathbb{Z}$ of integers and the above theorem is most frequently formulated just for polynomials over $\mathbb{Z}$:

Corollary 6.1.3. *Let $f = f_0 + f_1 x + \cdots + f_n x^n$ be a polynomial with integer coefficients. Then rational roots of f are of the form $\xi = p/q$, where p divides f_0 and q divides f_n.*

As we know, another example of a UFD is the ring of Gaussian integers. Hence in the above corollary we can substitute $\mathbb{Z}[i]$ for $\mathbb{Z}$ and $\mathbb{Q}(i)$ for $\mathbb{Q}$. Let us write the corollary in form of an algorithm.

Algorithm 6.1.4. *Given a polynomial $f = f_0 + f_1 x + \cdots + f_n x^n$ with rational coefficients, this algorithm returns all the rational roots of f.*

1. *Let $\mathcal{N} := \{m \in \mathbb{N} \mid m \text{ divides } f_0\}$ and $\mathcal{D} := \{m \in \mathbb{N} \mid m \text{ divides } \operatorname{lc}(f)\}$ be the sets of the divisors respectively of the constant and leading coefficients of f;*
2. *return the set $\{p/q \mid p \in \mathcal{N}, q \in \mathcal{D}, f(p/q) = 0\}$.*

We shall now mention some additional basic facts about roots of polynomials, that will be needed later in this chapter. All of them should be considered well known and are usually taught at basic algebra courses.

Proposition 6.1.5. *Let f be a polynomial with coefficients in $\mathbb{R}$. If a complex number $\xi \in \mathbb{C}$ is a root of f, then so is its conjugate $\overline{\xi}$.*

Proof. The assertion follows immediately from the fact that conjugation preserves sums and products. Since ξ is a root of f, we have

$$0 = f(\xi) = f_0 + f_1 \xi + \cdots + f_n \xi^n.$$

Conjugate both sides to get

$$0 = \overline{0} = \overline{f_0 + f_1 \xi + \cdots + f_n \xi^n} = \overline{f_0} + \overline{f_1}\,\overline{\xi} + \cdots + \overline{f_n}\,\overline{\xi}^n.$$

Now, all f_i's are real hence equal their conjugates and we have

$$0 = f_0 + f_1 \overline{\xi} + \cdots + f_n \overline{\xi}^n.$$

This means that $\overline{\xi}$ is a root of f, as claimed. □

One can express the coefficients of a polynomial in terms of sums of products of its roots. This technique dates back to works of F. Viète's[1] in 16th century.

Theorem 6.1.6 (Viète's formulæ)**.** *Let $f = f_0 + f_1 x + \ldots + f_{n-1}x^{n-1} + f_n x^n$ be a nonzero polynomial with coefficients in a subring of $\mathbb{C}$. Further, let $\xi_1, \ldots, \xi_n \in \mathbb{C}$ be all the roots of f, not necessarily distinct. Then the following identities hold:*

$$\begin{aligned}
(-1)\cdot\frac{f_{n-1}}{f_n} &= \xi_1 + \xi_2 + \cdots + \xi_n = \sum_{1\le i\le n} \xi_i \\
(-1)^2\cdot\frac{f_{n-2}}{f_n} &= \xi_1\xi_2 + \cdots + \xi_1\xi_n + \xi_2\xi_3 + \cdots + \xi_{n-1}\xi_n = \sum_{1\le i_1<i_2\le n} \xi_{i_1}\xi_{i_2} \\
&\vdots \\
(-1)^k\cdot\frac{f_{n-k}}{f_n} &= \sum_{1\le i_1<i_2<\cdots<i_k\le n} \xi_{i_1}\cdots\xi_{i_k} \\
&\vdots \\
(-1)^n\cdot\frac{f_0}{f_n} &= \xi_1\xi_2\cdots\xi_n.
\end{aligned}$$

Proof. In order to prove these formulæ, normilize the polynomial f and write it as a product of linear terms over $\mathbb{C}$:

$$\frac{1}{f_n}\cdot f = (x-\xi_1)\cdot(x-\xi_2)\cdots(x-\xi_n).$$

Expand the right-hand-side and compare the corresponding coefficients on both sides. □

Remark 48. Although we have formulated Viète's theorem for polynomials over a subring of $\mathbb{C}$, nevertheless it holds over any arbitrary integral domain. In the proof one needs just to replace $\mathbb{C}$ by the algebraic closure of the fraction field of the ring of coefficients.

Recall that *elementary symmetric polynomials* in n variables $x_1, \ldots, x_n$ are the polynomials:

$$\begin{aligned}
\sigma_1 &= x_1 + \cdots + x_n = \sum_{1\le i\le n} x_i \\
&\vdots \\
\sigma_k &= \sum_{1\le i_1<i_2<\cdots<i_k\le n} x_{i_1}\cdots x_{i_k} \\
&\vdots \\
\sigma_n &= x_1\cdot x_2\cdots x_n.
\end{aligned}$$

[1] François Viète (born around 1540 and died on 23 February 1603), known also as Franciscus Vieta, was a French lawyer and mathematician.

It follows that Viète's formulæ may be expressed in terms of elementary symmetric polynomials in a form $f_{n-i} = (-1)^i \cdot \sigma_i(a_1, \ldots, a_n)$. And immediate consequence of Theorem6.1.6 is that we may obtain the arithmetic mean of roots of a polynomial just from its coefficients:

Corollary 6.1.7. *If $f = f_0 + f_1 x + \cdots + f_{n-1}x^{n-1} + f_n x^n$ is a polynomial, then the (arithmetic) mean of its roots equals $\mu_f := -\frac{f_{n-1}}{n \cdot f_n}$.*

A monic polynomial f such that $\mu_f = 0$ is said to be *depressed*. We may think of a depressed polynomial as having its roots "balanced" at the origin. A depressed polynomial is immediately recognized by the fact that its coefficient at x^{n-1} is null (and the leading coefficient is of course 1). It is easy to transform any polynomial into a depressed one, shifting the mean of its roots to the origin.

Observation 6.1.8. *If $f = f_0 + f_1 x + \cdots + f_n x^n$ is a polynomial, then a polynomial g defined by a formula*

$$g := \frac{1}{f_n} \cdot f\Big(x - \frac{f_{n-1}}{n f_n}\Big)$$

is depressed.

Depressing a polynomial is a first step toward deriving explicit formulas for low degree polynomials. Moreover, as we shall see in Section 6.3, depressing a polynomial we generally improve estimation of a magnitude of its roots.

6.2 Explicit formulæ for low degree polynomials

The problem of finding the roots of univariate polynomials is probably as old as algebra itself. In this section we present classical close-form expressions for the roots of polynomials of degrees not exceeding four. The formulas presented below give exact solutions (in contrast to approximate solutions obtained by the tools developed in Section 6.6) and are not suitable for numerical evaluation. For a more elaborated history of these developments, we refer the reader to [38].

For a polynomial of degree one $f = ax+b$, the the unique root is of course $\xi = -b/a$. This is a kind of mathematics we encounter in a primary school. The solution of a quadratic equation $ax^2 + bx + c = 0$, that we all learned in the secondary school, was already known (in some form) to Babylonians around 1700 B.C. (c.f [41, Chapter 1]) and was explicitly given by Savasorda[2] in the middle of 12-th century. Actually, the Latin translation of Savasorda's treatise "*Ḥibbūr ha-meshīḥah we-ha-tishboret*" ("*Treatise on Measurement and Calculation*") appeared in the same year as the first Latin translation of "*Ilm al-jabr wa'l-muḳābala*" ("*Algebra*") by al-Khwārizmī[3], from which

[2] Abraham bar Ḥiyya ha-Nasi (born in 1070, died around 1140), also known as Savasorda, was a Jewish mathematician, astronomer and philosopher.

[3] Muḥammad ibn Mūsā al-Khwārizmī (born around 780, died around 850) was a Persian mathematician, astronomer and geographer.

the word "algebra" originates. As we know, the roots of $f = ax^2 + bx + c$ are given by the explicit formula:

$$\frac{1}{2a}\big(-b \pm \sqrt{\Delta}\big),$$

where $\Delta = b^2 - 4ac$ is called the *discriminant* of f (see Section 8.3 for a general notion of a discriminant).

It took nearly 400 years before a solution of a general cubic equation was discovered. The problem was first solved by S. del Ferro[4] for depressed cubics and shortly after by N. Tartaglia[5] in a general case. The solution was published for a first time in 1545 in "*Artis Magnæ, Sive de Regulis Algebraicis Liber Unus*" ("*Book number one about The Great Art, or The Rules of Algebra*") by G. Cardano[6]. Below we present a solution due to F. Viète, which differs in details from Cardano–Tartaglia solution. Suppose that we are given a cubic polynomial equation

$$ax^3 + bx^2 + cx + d = 0, \tag{6.1}$$

with $d \neq 0$. Depress it using Observation 6.1.8, that yields

$$x^3 + px + q = 0, \tag{6.2}$$

where

$$p = \frac{c}{a} - \frac{b^2}{3a^2} \qquad \text{and} \qquad q = \frac{2b^3}{27a^3} - \frac{bc}{3a^2} + \frac{d}{a}.$$

If $p = 0$, then the equation reduces to $x^3 + q = 0$ and clearly $\xi = \sqrt[3]{-q}$ is a solutions. From it we can easily recover a solution of the original (un-depressed) polynomial. If it is not a case, then we change a variable. Substitute $y - p/3y$ for x, where y is a new variable. This way, our equation reads as

$$y^3 + q - \frac{p^3}{27y^3} = 0.$$

Clearing the denominator, and substituting z for y^3, we finally arrive at:

$$z^2 + qz - \frac{p^3}{27} = 0. \tag{6.3}$$

This is a quadratic equation that we can solve using Savasorda's formula. Knowing a solution ζ for Eq. (6.3), we may revert all the substitutions to get a solution ξ for our original equation. The formula here is:

$$\xi = \sqrt[3]{\zeta} - \frac{p}{3\sqrt[3]{\zeta}} - \frac{b}{3a}.$$

[4]Scipione del Ferro (born in 1465, died in 1526) was an Italian mathematician.

[5]Niccolò Fontana Tartaglia (born around 1500, died in 1557) was a Venetian mathematician and engineer.

[6]Geronimo Cardano (born 24 September 1501, died 21 September 1576), known also as Hieronymus Cardanus, was an Italian mathematician, physician and philosopher.

Finally, knowing a single root ξ, one divides f by $x-\xi$, reducing the problem to finding the roots of a quadratic equation, which is easy to handle. Note, however, that Little Bézout Theorem was not yet know at the time of Tartaglia and Cardano. There exists also a close-form expression for all the three roots of f.

Proposition 6.2.1. *Let $f = ax^3 + bx^2 + cx + d$ be a cubic polynomial and let $\zeta = -1/2 + i\sqrt{3}/2$ denote the primitive root of unity of degree 3. Then the roots of f are given by the formula*

$$-\frac{1}{3a}\Big(b + \zeta^k \cdot C + \frac{\Delta_0}{\zeta^k \cdot C}\Big), \qquad \text{for} \quad k = 0, 1, 2.$$

Here we denote

$$C = \sqrt[3]{\frac{\Delta_1 + \sqrt{\Delta_1^2 - 4\Delta_0^3}}{2}}, \qquad \Delta_0 = c^2 - 3ac \quad \text{and} \quad \Delta_1 = 2b^3 - 9abc + 27a^2d.$$

Proof

It is time to turn our attention to quartic polynomials. Here the solution was found shortly after the cubic case by L. Ferrari[7], who was a student of G. Cardano. The general idea is similar to that of the solution of a cubic equation. First we reduce the equation to a depressed one and then the depressed form is reduced to a cubic equation. The solution presented below is due to J.-L. Lagrange.

We are given a quartic polynomial, without loss of generality we may assume that it is a already a depressed one:

$$f = r + qx + px^2 + x^4. \tag{6.4}$$

Denote the roots of f by $\xi_1, \xi_2, \xi_3, \xi_4$. Take a matrix

$$M = \frac{1}{2} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

and observe that its inverse equals again M. Make a linear transformation of roots, defining $\zeta_1, \ldots, \zeta_4$ by a formula

$$\begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \zeta_3 \\ \zeta_4 \end{bmatrix} := M \cdot \begin{bmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \\ \xi_4 \end{bmatrix}.$$

[7] Lodovico Ferrari (born on 2 February 1522, died on 5 October 1565) was an Italian mathematician.

In particular ζ_1, being the sum of roots of a depressed polynomial, is equal 0. Construct a polynomial h, whose roots are the squares of ζ_2, ζ_3 and ζ_4. Let

$$\begin{aligned} h &= \big(x-\zeta_2^2\big)\big(x-\zeta_3^2\big)\big(x-\zeta_4^2\big) \\ &= -\zeta_2^2\zeta_3^2\zeta_4^2 + \big(\zeta_2^2\zeta_3^2+\zeta_2^2\zeta_4^2+\zeta_3^2\zeta_4^2\big)x - \big(\zeta_2^2+\zeta_3^2+\zeta_4^2\big)x^2 + x^3. \end{aligned}$$

Now, substitute into ζ_i's the values expressed in terms of the ξ_i's. It results in a polynomial with enormously large coefficients. But all these coefficients are symmetric polynomials in variables $\xi_1, \ldots, \xi_4$. By the fundamental theorem of symmetric polynomials (see e.g. [51, Theorem IV.6.1]), every symmetric polynomial can be expressed in terms of elementary symmetric polynomials $\sigma_1, \ldots, \sigma_4$. Therefore the polynomial h takes a form $h = h_0 + h_1 x + h_2 x^2 + x^3$, where:

$$\begin{aligned} h_0 &= -\frac{1}{64}\sigma_1^6 + \frac{1}{8}\sigma_1^4\sigma_2 - \frac{1}{4}\sigma_1^2\sigma_2^2 - \frac{1}{4}\sigma_1^3\sigma_3 + \sigma_1\sigma_2\sigma_3 - \sigma_3^2, \\ h_1 &= \frac{3}{16}\sigma_1^4 - \sigma_1^2\sigma_2 + \sigma_2^2 + \sigma_1\sigma_3 - 4\sigma_4, \\ h_2 &= -\frac{3}{4}\sigma_1^2 + 2\sigma_2. \end{aligned}$$

The above formulas were computed using the computer algebra system Sage [25]. For a sake of completeness, we present a full code in an appendix. Recall that by Viète's formula $\sigma_1 = \xi_1 + \xi_2 + \xi_3 + \xi_4 = 0$ while $\sigma_2 = p$, $\sigma_3 = -q$ and $\sigma_4 = r$. It follows that the polynomial h has indeed a form

$$h = -q^2 + \big(p^2 - 4r\big)x + 2px^2 + x^3. \tag{6.5}$$

This is a polynomial of degree 3 and we can find its (exact) roots $\zeta_2^2, \zeta_3^2, \zeta_4^2$, using the methods described earlier in this section. Once we know all the four ζ_i's (recall that $\zeta_1 = 0$), we can recover the roots of f using the fact that $M^{-1} = M$ so that

$$\begin{bmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \\ \xi_4 \end{bmatrix} = M^{-1} \cdot \begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \zeta_3 \\ \zeta_4 \end{bmatrix} = M \cdot \begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \zeta_3 \\ \zeta_4 \end{bmatrix}.$$

Now, knowing how to solve polynomials of degrees: one, two, three and four, one might hope for a general solution for polynomials of any arbitrary degree. And so did mathematicians till nearly the end of 18-th century. Unfortunately, this hope is void in the light of the following theorem formulated originally by P. Ruffini[8] in 1799 in a paper "*Teoria Generale delle Equazioni, in cui si dimostra impossibile la soluzione algebraica delle equazioni generali di grado superiore al quarto*" ("*General Theory of equations, in which the algebraic solution of general equations of degree higher than four is proven*

[8] Paolo Ruffini (born on 22 September 1765, died 10 May 1822) was an Italian mathematician and philosopher.

impossible"), but completely proved only 24 years later by N. Abel[9]. In order to formulate the theorem, though, we need first to introduce a new term. Informally speaking, given a field K and its subring $R \subset K$, we say that an element $\alpha \in K$ is *expressible in radicals* over R, if it can be expressed by finitely many elements of R connected by the four basic arithmetic operations (i.e.: $+, -, \cdot, /$) together with the exponentiation $a \mapsto a^n$ and n-th roots $a \mapsto \sqrt[n]{a}$. A formal definition is the following one:

Definition 6.2.2. Let R be a subring of a field K and let $\alpha \in K$. We say that $\alpha \in K$ is *expressible in radicals* over R, if there is a finite chain $R = R_0 \subsetneq R_1 \subsetneq \cdots \subsetneq R_k$ of subrings of K such that $\alpha \in R_k$ and $R_i \cong {}^{R_{i-1}[x]}/_{(x^{n_i} - a_i)}$, where $a_i \in R_{i-1}$ and $n_i \in \mathbb{N}$ for $i \in \{1, \ldots, k\}$, .

Theorem 6.2.3 (Abel–Ruffini)**.** *For every $n \geq 5$, there is a polynomial f of degree $\deg f = n$, whose roots are not expressible in radicals over the ring of the coefficients of f.*

Proof

Examples of unsolvable polys

6.3 Bounds on roots

This section is devoted to a problem of bounding roots of a polynomial by some easy to compute expressions, depending on its coefficients. More precisely, given a polynomial (in a traditional form) $f = f_0 + f_1 x + \cdots + f_n x^n$, we want to find:

- a constant M (depending on $f_0, \ldots, f_n$) such that $M > |\xi|$ for every complex root ξ of f;
- a constant m such that $0 < m < |\xi|$ for every nonzero complex root ξ of f;
- two constants M_+ and M_- such that $M_- < \xi < M_+$ for every real root ξ of f.

It is easy to notice that answers to these three questions are interrelated. A trivial observation is that $-M \leq M_- \leq M_+ \leq M$ for every polynomial f. Similarly, if M_+ is an upper bound for real roots of $f(-x)$, then $-M_+$ is a lower bound for real roots of f. One may also obtain a lower bound m for nonzero roots of f, from the upper bound for roots of the reciprocal polynomial (see Definition 1.1.1).

Observation 6.3.1. *Let $f = f_0 + f_1 x + \cdots + f_n x^n$ be a polynomial and $g = \operatorname{rev}(f) = f_n + f_{n-1} x + \cdots + f_0 x^n$ be its reciprocal. If M is an upper bound for absolute values of roots of g, then $m := {}^1/_M$ is a lower bound for absolute values of nonzero roots of f.*

Indeed, Observation 1.1.2 asserts that $g = x^n \cdot f({}^1/_x)$. Thus, $\xi \neq 0$ is a roots of f if and only if ${}^1/_\xi$ is a root of g. Since $M > |{}^1/_\xi|$ it follows that $m = {}^1/_M < |\xi|$, as claimed. As mention in the first section, depressing a polynomial one may sometimes improve a bound on its roots.

[9] Niels Henrik Abel (born 5 August 1802, died 6 April 1829) was a Norwegian mathematician.

Observation 6.3.2. *Let $f = f_0 + f_1 x + \cdots + f_n x^n$ be a polynomial, $\mu_f = {}^{-f_{n-1}}/_{n \cdot f_n}$ the mean of its roots. Set $g := f(x + \mu_f)$. If M_+ is an upper bound for real roots of g, then $M_+ + \mu_f$ is an upper bound for real roots of f.*

A priori, it is neither obvious nor even true that the upper bound obtained by the means of the previous observation is any better than the one computed directly for f. Nevertheless, in practice, it is not uncommon that applying the same algorithm to a depressed form of a polynomial and adding back the mean renders better estimate than applying the same algorithm directly to f. Yet still, there are polynomials (and root estimation algorithms) for which the Observation 6.3.2 gives worse, or even much worse, bounds that the ones computed directly. Anyway, one thing that is clear (by the triangle inequality) is that, if $M(f)$ denotes the maximum absolute value of roots of f, then $-M(f) \leq \mu_f \pm M(g) \leq M(f)$, with a strict inequality whenever $\mu_f \neq 0$.

After this preliminaries, we may now present a few methods for estimating the magnitude of roots of a polynomial. We begin with a classical result by A.-L. Cauchy[10]:

Proposition 6.3.3 (Cauchy, 1829)**.** *Let $f = f_0 + f_1 x + \cdots + f_n x^n$ be a polynomial of degree n (i.e. $f_n \neq 0$) and let $\xi \in \mathbb{C}$ be a root of f. Then the absolute value of ξ can be bounded as follows*

$$|\xi| < 1 + \max_{0 \leq j \leq n-1} \left| \frac{f_j}{f_n} \right|. \tag{6.6}$$

Proof. If $|\xi| \leq 1$, then the assertion holds trivially. Suppose that $|\xi| > 1$. Normalize the polynomial, dividing it by $f_n = \mathrm{lc}(f)$. Since ξ is a root of f, we can write

$$\xi^n = -\frac{f_0}{f_n} - \frac{f_1}{f_n}\xi - \cdots - \frac{f_{n-1}}{f_n}\xi^{n-1}.$$

Denote $M := \max\limits_{0 \leq j \leq n} \left| \frac{f_j}{f_n} \right|$ and compute the absolute values of both sides of the above equation.

$$|\xi|^n = |\xi^n| = \left| \sum_{j=0}^{n-1} \frac{f_j}{f_n} \xi^j \right| \leq \sum_{j=0}^{n-1} \left| \frac{f_j}{f_n} \right| |\xi|^j \leq M \cdot \sum_{j=0}^{n-1} |\xi|^j.$$

Now, the last term is just a sum of a finite geometric sequence and can be expressed explicitly as $\frac{|\xi|^n - 1}{|\xi| - 1}$. This gives us an estimate

$$|\xi|^n \leq M \cdot \frac{|\xi|^n - 1}{|\xi| - 1}.$$

The denominator is positive, because we have assumed that $|\xi| > 1$, it follows that

$$|\xi| - 1 \leq M \cdot \frac{|\xi|^n - 1}{|\xi|^n} < M.$$

Consequently $|\xi| < M + 1$. □

[10] Augustin-Louis Cauchy (born 21 August 1789, died 23 May 1857) was a French aristocrat (he was a baron) and mathematician.

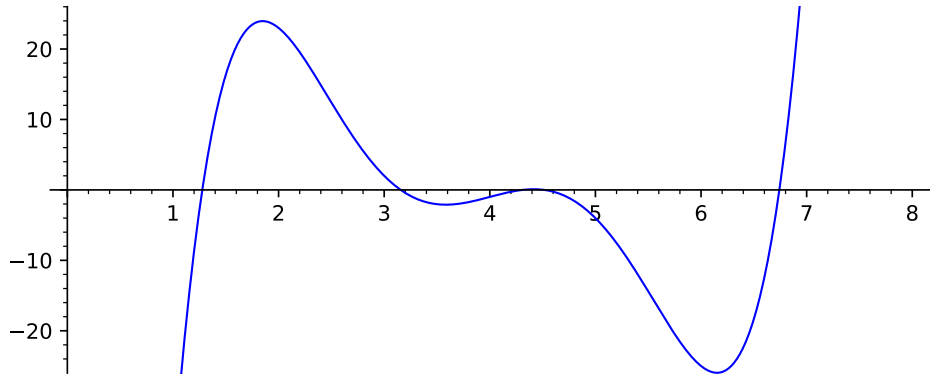


Figure 6.1: Graph of a polynomial $f = x^5 - 20x^4 + 152x^3 - 544x^2 + 900x - 529$.

Example 68. Take a polynomial $f = x^5 - 20x^4 + 152x^3 - 544x^2 + 900x - 529$. The graph of f is presented in Figure 6.1. This polynomial is unsolvable, hence we cannot compute its roots exactly. Using methods that will be discussed in Section 6.6 we may approximate the roots:

$$\xi_1 = 1.28, \quad \xi_2 = 3.16, \quad \xi_3 = 4.31, \quad \xi_4 = 4.52, \quad \xi_5 = 6.74.$$

The Cauchy bound for the roots of f is

$$M_f = 1 + \max\{|1|, |-20|, |152|, |-544|, |900|, |-529|\} = 901.$$

This estimate is definitely too excessive. We may improve the estimate using Observation 6.3.2. The mean of roots of f is

$$\mu_f = \frac{-f_4}{5 \cdot f_5} = 4.$$

Take thus the depressed form of f:

$$g := f(x+4) = x^5 - 8x^3 + 4x - 1.$$

Now, Proposition 6.3.3 gives a bound $M_g = 9$ for roots of g and so the resulting upper bound for roots of f is $M_g + \mu_f = 13$.

Remove next Cor?

We may now illustrate how to use Observation 6.3.1 to transform the above upper bound into a corresponding lower bound:

Corollary 6.3.4. *Let $f = f_m x^m + \cdots + f_n x^n$ be a polynomial, where $m < n$ and f_m, f_n are non-zero. If $\xi \neq 0$ is a root of f, then the absolute value of ξ can be bounded as follows*

$$|\xi| > \frac{1}{1 + \max\limits_{m<j\leq n} \left|\frac{f_j}{f_m}\right|}. \tag{6.7}$$

Another bound on the absolute value of a root of a polynomial, again named after A.-L. Cauchy, is the following one:

Proposition 6.3.5 (Cauchy, 1829)**.** *Let $f = f_0 + f_1 x + \cdots + f_n x^n$ be a polynomial and $\xi \in \mathbb{C}$ its root, then*

$$|\xi| \leq \max_{0\leq j<n} \left(n \cdot \left|\frac{f_j}{f_n}\right| \right)^{\frac{1}{n-j}}. \tag{6.8}$$

Proof. Denote the right-hand-side of the inequality (6.8) by M and fix a complex number $\xi \in \mathbb{C}$ having an absolute value $|\xi| > M$. Then

$$|\xi|^{n-j} > n \cdot \frac{|f_j|}{|f_n|} \qquad \text{for every} \quad 0 \leq j < n.$$

All the appearing terms are strictly positive. Rearranging the terms we obtain an estimate for a coefficient f_j of the polynomial f:

$$|f_j| < \frac{|\xi|^{n-j} \cdot |f_n|}{n}. \qquad \text{for every} \quad 0 \leq j < n.$$

Applying this estimate to every coefficient, but the leading one, we have:

$$\left|f_0 + f_1\xi + \cdots + f_{n-1}\xi^{n-1}\right| \leq |f_0| + |f_1||\xi| + \cdots + |f_{n-1}||\xi|^{n-1} < \frac{|\xi|^n \cdot |f_n|}{n} + \cdots + \frac{|\xi|^n \cdot |f_n|}{n} = |f_n\xi^n|.$$

Consequently $f(\xi) \neq 0$ and this proves the assertion. □

Example 69. Continue the precious example. We have a polynomial $f = x^5 - 20x^4 + 152x^3 - 544x^2 + 900x - 529$ and its depressed form $g = x^5 - 8x^3 + 4x - 1$. Proposition 6.3.5 yields an upper bound for roots of f:

$$M_f = \max\left\{ \sqrt[1]{5 \cdot |-20|}, \sqrt[2]{5 \cdot |152|}, \sqrt[3]{5 \cdot |-544|}, \sqrt[4]{5 \cdot |900|}, \sqrt[5]{5 \cdot |-529|} \right\} = 100.$$

We may again improve the estimate using Observation 6.3.2. The bound for the depressed polynomial is

$$M_g = \max\left\{ \sqrt[2]{5 \cdot |-8|}, \sqrt[4]{5 \cdot |4|}, \sqrt[5]{5 \cdot |-1|} \right\} = 2\sqrt{10} \approx 6.32.$$

Thus, $M_g + \mu_f \approx 10.32$ is an upper bound for roots of f.

Generating root bounds

In the previous section we presented two ad hoc bounds for roots of polynomials (Propositions 6.3.3 and 6.3.5). There is, however, a systematic approach—due to H. Wilf—for deriving such bound based on properties of... matrices. Fix a polynomial $f = f_0 + f_1 x + \cdots + f_n x^n$ and assume that the constant term f_0 is nonzero. If it is not the case, we may always divide f by a maximal power of x. The resulting polynomial f/x^m has a nontrivial constant term and the same roots as f, except the root at the origin.

Recall (see Definition 3.8.6 and Proposition 3.8.8) that a companion matrix of f has a form

$$C_f := \begin{pmatrix} 0 & 0 & \cdots & 0 & -f_0/f_n \\ 1 & 0 & \cdots & 0 & -f_1/f_n \\ 0 & 1 & \cdots & 0 & -f_2/f_n \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & 1 & -f_{n-1}/f_n \end{pmatrix}.$$

Consider a matrix $|C_f|$ obtained from C_f by replacing each entry by its absolute value

$$|C_f| := \begin{pmatrix} 0 & 0 & \cdots & 0 & |f_0/f_n| \\ 1 & 0 & \cdots & 0 & |f_1/f_n| \\ 0 & 1 & \cdots & 0 & |f_2/f_n| \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & 1 & |f_{n-1}/f_n| \end{pmatrix}.$$

It is clear that $|C_f|$ is non-negative. We wish to apply Frobenius theorem (Theorem 3.6.15) to it. To this end we must show its irreducibility (see Definition 3.6.14).

Lemma 6.3.6. *If $f_0 \neq 0$, then $|C_f|$ is irreducible.*

Proof. We must prove that for every pair of indices (i, j), there is an exponent $k \geq 1$ such that the corresponding entry of $|C_f|^k$ is strictly positive. Recall that in Section 3.6 we introduced a partial order relation $\leq$ by comparing matrices entry-wisely. It is easy to observe that for non-negative matrices the relation $\leq$ is preserved by exponentiation: if $A \leq B$ are non-negative, then $A^k \leq B^k$ for every exponent $k \geq 1$. This is a straightforward consequence of the fact that matrix multiplication is just an addition and multiplication of (non-negative) entries.

Denote $a := |f_0/f_n|$. By assumption $a > 0$. It follows from the previous paragraph, that it suffices to show that a matrix

$$M := \begin{pmatrix} 0 & 0 & \cdots & 0 & a \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix},$$

is irreducible, since clearly $|C_f| \geq M$. The consecutive powers of M are obtained by 'sliding' diagonals:

$$M = \left(\begin{array}{cc|c} & & a \\ & \iddots & \\ 1 & & \end{array}\right), \quad M^2 = \left(\begin{array}{cc|c} & & a \\ & \iddots & a \\ 1 & & \end{array}\right), \quad M^3 = \left(\begin{array}{cc|c} & & a \\ & \iddots & \iddots \\ 1 & & a \end{array}\right)$$

and so on, till we reach

$$M^{n-1} = \left(\begin{array}{c|cc} & a & \\ & \iddots & \\ & & a \\ \hline 1 & & \end{array}\right) \quad \text{and} \quad M^n = \begin{pmatrix} a & & \\ & \ddots & \\ & & a \end{pmatrix}$$

It is now obvious that a line of positive entries will eventually pass through any fixed spot of the matrix at most in the n-th iteration. □

Theorem 6.3.7 (Wilf, 1961)**.** *Let $f = f_0 + f_1 x + \cdots + f_n x^n$ be a polynomial and assume that $f_0 \neq 0$. Further, denote $a_0 := 0$ and let $a_1, \ldots, a_n$ be arbitrary positive numbers. If $\xi \in \mathbb{C}$ is a root of f, then*

$$|\xi| \leq \max_{0 \leq j \leq n-1} \left(\left| \frac{f_j}{f_n} \right| \cdot \frac{a_n}{a_{j+1}} + \frac{a_j}{a_{j+1}} \right).$$

Proof. The roots of f are precisely the eigenvalues of its companion matrix C_f, by Observation 3.8.7. Theorem 3.6.17 asserts that they are dominated by the maximal eigenvalue (spectral radius) Λ of $|C_f|$. The spectral radius, in turn, can be computed by the means of Theorem 3.6.16:

$$\Lambda = \min_{\nu > 0} \max_{1 \leq j \leq n} \frac{(|C_f| \cdot \nu)_j}{\nu_j}.$$

A vector $a := (a_1, \ldots, a_n)$ is definitely positive. Hence

$$\Lambda \leq \max_{1 \leq j \leq n} \frac{(|C_f| \cdot a)_j}{a_j} = \max_{1 \leq j \leq n} \left(\frac{a_{j-1} + |f_{j-1}/f_j| \cdot a_n}{a_j} \right),$$

with the convention that $a_0 = 0$. The assertion follows now by a simple change of indexation. □

A freedom of choice of parameters $a_1, \ldots, a_n$ is an evident advantage of Wilf theorem. Different choices lead to different formulas for root bounds. First let us see how

the classical Cauchy bound follows from the general theory. Take $a_1 = \cdots = a_n = 1$ and of course $a_0 = 0$. Theorem 6.3.7 asserts that for any root ξ we have

$$|\xi| \leq \max\left(\left|\frac{f_0}{f_n}\right|, \left|\frac{f_1}{f_n}\right| + 1, \ldots, \left|\frac{f_{n-1}}{f_n}\right| + 1\right) \leq 1 + \max_{0 \leq j \leq n-1} \left|\frac{f_j}{f_n}\right|,$$

which is just the Cauchy bound, except that in Proposition 6.3.3 one has a sharp inequality. Now consider two further examples:

Proposition 6.3.8 (Kojima, 1917). *Let $f = f_0 + f_1 x + \cdots + f_n x^n$ be a polynomial. Assume that all its coefficients are nonzero. If $\xi \in \mathbb{C}$ is a root of f, then*

$$|\xi| \leq \max\left(\left|\frac{f_0}{f_1}\right|, 2\left|\frac{f_1}{f_2}\right|, \ldots, 2\left|\frac{f_{n-1}}{f_n}\right|\right).$$

Proof. Set $a_j := |f_j|$ for $j \in \{1, \ldots, n\}$ and as always $a_0 := 0$. Use Wilf theorem to get

$$|\xi| \leq \max\left(\left|\frac{f_0}{f_n}\right| \cdot \left|\frac{f_n}{f_1}\right|, \left|\frac{f_1}{f_n}\right| \cdot \left|\frac{f_n}{f_2}\right| + \left|\frac{f_1}{f_2}\right|, \ldots, \left|\frac{f_{n-1}}{f_n}\right| \cdot \left|\frac{f_n}{f_n}\right| + \left|\frac{f_{n-1}}{f_n}\right|\right). \qquad \square$$

Proposition 6.3.9 (Fujiwara, 1916). *Let $f = f_0 + f_1 x + \cdots + f_n x^n$ be a polynomial with a nonzero constant coefficient. If $\xi \in \mathbb{C}$ is a root of f, then*

$$|\xi| \leq 2 \cdot \max_{0 \leq j \leq n-1} \left|\frac{f_j}{f_n}\right|^{\frac{1}{n-j}}.$$

Proof, example

Eneström-Kakeya bound ?

Ştefănescu–Vigklas theorem

In this subsection we introduce a method for generating upper bounds for **real** roots of polynomials. The method needs some preparations.

Better term?

Definition 6.3.10. We say that a polynomial $f \in \mathbb{R}[x]$ is in a *balanced form*, if it is written as

$$f = (g_1 x^{\varepsilon_1} - h_1 x^{e_1}) + \cdots + (g_m x^{\varepsilon_m} - h_m x^{e_m}) + r,$$

where $g_1, \ldots, g_m, h_1, \ldots, h_m$ are strictly positive real numbers, r is a polynomial with non-negative coefficients and $\varepsilon_j > e_j$ for every $j \in \{1, \ldots, m\}$.

In the above definition we **do not** assume that exponents $\varepsilon_1, \ldots, \varepsilon_m$ (neither $e_1, \ldots, e_n$) are distinct!

Observation 6.3.11. *A nonzero polynomial can be expressed in a balanced form if and only if its leading coefficient is positive.*

Indeed, take $f = f_0 + f_1 x + \cdots + f_n x^n$ with $f_n > 0$ and denote by s the number of negative coefficients of f. Write f in a balanced form as

$$f = \sum_{\substack{k=0,\ldots,n-1 \\ f_k<0}} \left(\frac{f_n}{s} x^n - f_k x^k \right) + \sum_{\substack{k=0,\ldots,n-1 \\ f_k>0}} f_k x^k.$$

On contrary, if $f_n < 0$ then there is no $\varepsilon_k > n$ and so $f_n x^n$ cannot be matched with any $g_k x^{\varepsilon_k}$. In practice, the restriction $\operatorname{lc}(f) > 0$ does not limit us in any way, since we can always replace f by $-f$, as they both have the same set of roots. A balanced form of a polynomial is definitely not unique, nevertheless any balanced form of a polynomial give rise to some estimate on the magnitude of its roots. How tight is the bound depend on a choice of the form.

Theorem 6.3.12 (Ştefănescu–Vigklas). *Let $f = (g_1 x^{\varepsilon_1} - h_1 x^{e_1}) + \cdots + (g_m x^{\varepsilon_m} - h_m x^{e_m}) + r$ be a polynomial in a balanced form. Then*

$$M_+ := \max_{1 \le j \le m} \left(\frac{h_j}{g_j} \right)^{\frac{1}{\varepsilon_j - e_j}}$$

is an upper bound for real roots of f.

Proof. It is clear that M_+ is non-negative, hence if f has no positive roots, then the assertion holds trivially. Take now any $\xi > 0$, then

$$f(\xi) = \left(g_1 \xi^{\varepsilon_1} - h_1 \xi^{e_1} \right) + \cdots + \left(g_m \xi^{\varepsilon_m} - h_m \xi^{e_m} \right) + \text{non-negative terms}.$$

If $\xi > M_+$, then for every $j \in \{1, \ldots, m\}$ we have

$$g_j \xi^{\varepsilon_j} - h_j \xi^{e_j} = \xi^{e_j} \cdot \left(g_j \xi^{\varepsilon_j - e_j} - h_j \right) > M_+^{e_j} \cdot \left(g_j \cdot \frac{h_j}{g_j} - h_j \right) = 0.$$

This shows that $f(\xi) > 0$ for $\xi > M_+$. □

Example 70. Take a polynomial $f = x^5 - 20x^4 + 152x^3 - 544x^2 + 900x - 529$. We may write it in a balanced form by splitting the leading coefficient as suggested in the comment following Observation 6.3.11:

$$f = \left(\frac{1}{3} x^5 - 20x^4 \right) + \left(\frac{1}{3} x^5 - 544x^2 \right) + \left(\frac{1}{3} x^5 - 529 \right) + \left(152x^3 + 900x \right).$$

Ştefănescu–Vigklas theorem provides an upper bound for roots of f:

$$M_+ = \max \left\{ \left(\frac{20}{\frac{1}{3}} \right)^{\frac{1}{5-4}}, \left(\frac{544}{\frac{1}{3}} \right)^{\frac{1}{5-2}}, \left(\frac{529}{\frac{1}{3}} \right)^{\frac{1}{5-0}} \right\} = 60.$$

But we may also group the term of f differently:

$$f = \left(x^5 - 20x^4 \right) + \left(152x^3 - 544x^2 \right) + \left(900x - 529 \right).$$

Now, Theorem 6.3.12 yields a better bound:

$$M_+ = \max\left\{\left(\frac{20}{1}\right)^{\frac{1}{5-4}}, \left(\frac{544}{152}\right)^{\frac{1}{3-2}}, \left(\frac{529}{900}\right)^{\frac{1}{1-0}}\right\} = 20.$$

As it is evident from the above example, a choice of a balanced form of a polynomial determines how good the resulting bound is. Below we describe two strategies for computing an upper bound for roots of polynomials that use Theorem 6.3.12.

The first strategy is called a *local-max* method. Here, h_j's are just the negative coefficients of f and every such $h_j < 0$ is paired with $g_j = {}^{f_i}/_{2^t}$, where f_i is the largest positive coefficient of f of higher degree and the exponent t is the number of times this particular coefficient f_i has been used so far. In form of a pseudo-code the local-max method can be expressed like this:

Algorithm 6.3.13 (Local-max bound)**.** *Given a polynomial $f = f_0 + f_1 x + \cdots + f_n x^n$ with a real coefficients, this algorithm returns an upper bound for real roots of f.*

1. *If the leading coefficient of f is negative, then replace f by $-f$;*
2. *initialize the index of a local maximum $i := n$, its exponent $t := 1$ and the upper bound $M := 0$;*
3. *for every $j = n-1, n-2, \ldots, 0$ do the following:*
   a) *if the coefficient f_j is positive and greater than f_i, then update the index of the local maximum setting $i := j$ and reset the exponent $t := 1$;*
   b) *if the coefficient f_j is negative, then update the upper bound setting*

$$M := \max\left\{M, \left(2^t \cdot \frac{-f_j}{f_i}\right)^{\frac{1}{i-j}}\right\}$$

   and increment the exponent $t := t + 1$;
4. *return the upper bound M.*

Example 71. Consider again the polynomial $f = x^5 - 20x^4 + 152x^3 - 544x^2 + 900x - 529$. Algorithm 6.3.13 matches each negative term of f with a quotient of a maximal positive coefficient of a higher exponent. Hence, the polynomial is conceptually converted into a balanced form

$$f = \left(\frac{1}{2^1}x^5 - 20x^4\right) + \left(\frac{152}{2^1}x^3 - 544x^2\right) + \left(\frac{900}{2^1}x - 529\right) + \left(\frac{1}{2}x^5 + 76x^3 + 450x\right).$$

Theorem 6.3.12 yields the following upper bound for real roots of f

$$M_f = \max\left\{\left(\frac{529}{\frac{900}{2}}\right)^{\frac{1}{1-0}}, \left(\frac{544}{\frac{152}{2}}\right)^{\frac{1}{3-2}}, \left(\frac{20}{\frac{1}{2}}\right)^{\frac{1}{5-4}}\right\} = 40.$$

The bound can be again improved using Observation 6.3.2, which yields an upper bound 8.

Lagrange-MacLaurin bound (?)

Hong's bound

Another bound for positive roots of f, that can be proved using Ştefănescu–Vigklas theorem, is the following one:

Proposition 6.3.14 (Hong, 1998)**.** *Let $f = f_0 + f_1 x + \cdots + f_n x^n$ be a polynomial with a strictly positive leading coefficient. If $\xi \in \mathbb{R}$ is a root of f, then*

$$\xi < 2 \max_{\substack{j\in\{0,\ldots,n-1\}\\ f_j<0}} \min_{\substack{i\in\{j+1,\ldots,n\}\\ f_i>0}} \left(\frac{-f_j}{f_i}\right)^{\frac{1}{i-j}}.$$

Proof. Denote the set of indices of negative coefficients of f by $\mathscr{N} := \big\{j \in \{0,\ldots,n-1\} \,|\, f_j < 0\big\}$. For every $j \in \mathscr{N}$, let $m(j)$ be the index $k > j$ such that

$$\left(\frac{-f_j}{f_k}\right)^{\frac{1}{k-j}} = \min\left\{\left(\frac{-f_j}{f_i}\right)^{\frac{1}{i-j}} \;\middle|\; i > j \text{ and } f_i > 0\right\}.$$

Write a polynomial f in a balanced form as

$$f = \sum_{j\in\mathscr{N}}\left(\frac{f_{m(j)}}{2^{m(j)-j}}x^{m(j)} - (-f_j)x^j\right) + \left(\sum_{j\in\mathscr{N}} \frac{f_{m(j)}}{1-\sum\limits_{\substack{i\in\mathscr{N}\\ m(i)=m(j)}} 2^{m(j)-i}} \cdot x^{m(j)} + \sum_{\substack{0\le k\le n\\ f_k>0\\ k\notin m(\mathscr{N})}} f_k x^k\right),$$

here $k \notin m(\mathscr{N})$ means that k is not one of $m(j)$ for $j \in \mathscr{N}$. If $\xi \in \mathbb{R}$ is a root of f, then Theorem 6.3.12 asserts that

$$\xi < \max_{j\in\mathscr{N}}\left(\frac{-f_j}{\frac{f_{m(j)}}{2^{m(j)-j}}}\right)^{\frac{1}{m(j)-j}} = 2\cdot\max_{j\in\mathscr{N}}\min_{\substack{i>j\\ f_i>0}}\left(\frac{-f_j}{f_i}\right)^{\frac{1}{i-j}}. \qquad \square$$

Example 72. Take again the same polynomial we have used for the last few examples:

$$f = x^5 - 20x^4 + 152x^3 - 544x^2 + 900x - 529.$$

This polynomial has three negative coefficients. For each one we must compute a corresponding minimum:

$$\begin{aligned}
-529: &\quad \min\left\{\left(\frac{529}{900}\right)^{\frac{1}{1-0}}, \left(\frac{529}{152}\right)^{\frac{1}{3-0}}, \left(\frac{529}{1}\right)^{\frac{1}{5-0}}\right\} = \frac{529}{900},\\
-544: &\quad \min\left\{\left(\frac{544}{152}\right)^{\frac{1}{3-2}}, \left(\frac{544}{1}\right)^{\frac{1}{5-2}}\right\} = \frac{68}{19},\\
-20: &\quad \min\left\{\left(\frac{20}{1}\right)^{\frac{1}{5-4}}\right\} = 20.
\end{aligned}$$

The Hong's bound for f is twice the maximum of these numbers:

$$M_f = 2 \cdot \max\left\{\frac{529}{900}, \frac{68}{19}, 20\right\} = 40.$$

Once again, Observation 6.3.2 may be used to improve the result. Recall that $g = x^5 - 8x^3 + 4x - 1$ is the depressed polynomial obtained from f. Now, the Hong's bound for roots of g is $M_g = 4\sqrt{2} \approx 5.66$. Consequently the corresponding bound for roots of f equals $M_g + \mu_f \approx 9.66$.

The time complexity of computing Hong's bound directly by definition is $O(n^2)$, where n denotes the degree of a polynomial in question. There is however another method for computing this bound that has time complexity only $O(n)$. Denote half of the Hong's bound by $H = H(f)$:

$$H := \max_{\substack{0 \le j < n \\ f_j < 0}} \min_{\substack{j < i \le n \\ f_i > 0}} \left(\frac{|f_j|}{|f_i|}\right)^{\frac{1}{i-j}}.$$

Since a function $x \mapsto \lg x := \log_2 x$ that maps x to its base-two logarithm is monotonic, we have

$$\lg H = \max_{\substack{0 \le j < n \\ f_j < 0}} \min_{\substack{j < i \le n \\ f_i > 0}} \frac{\lg|f_j| - \lg|f_i|}{i-j}.$$

If we interpret a pair $p_i := \big(i, -\lg|f_i|\big)$ as a point in the real plane $\mathbb{R}^2$, then the expression

$$s_{ij} := \frac{\big(-\lg|f_i|\big) - \big(-\lg|f_j|\big)}{i-j}$$

is the slope of the line through p_i and p_j. We say that a point $p_i = \big(i, -\lg|f_i|\big)$ is *positive* (respectively *negative*) if $f_i > 0$ (resp. $f_i < 0$). The set of positive points with abscissæ greater or equal to some fixed integer j will be denoted

$$\mathscr{P}_j := \big\{(i, -\lg|f_i|) \,|\, i \ge j \text{ and } f_i > 0\big\}.$$

It corresponds to the positive coefficients in a "tail" $f_j x^j + \cdots + f_n x^n$ of the polynomial f. Analogously we define the set of negative points:

$$\mathscr{N}_j := \big\{(i, -\ln|f_i|) \,|\, i \ge j \text{ and } f_i < 0\big\}.$$

Denote by $\operatorname{conv}(\mathscr{P}_j)$ the convex hull of $\mathscr{P}_j$. If p_j is a positive point, then p_j and p_n are respectively the left-most and right-most points of $\operatorname{conv}(\mathscr{P}_j)$. Hence they lie on the boundary of the convex hull and split this boundary into two chains of points (see Figure 6.2). One chain lies under (or on) the line through p_j and p_n, the other one lies above (or on) this line. The former chain is called the *lower hull* of $\mathscr{P}_j$ and denoted $\underline{\operatorname{conv}}(\mathscr{P}_j)$. The later one is called the *upper hull* and denoted $\overline{\operatorname{conv}}(\mathscr{P}_j)$, but we do not

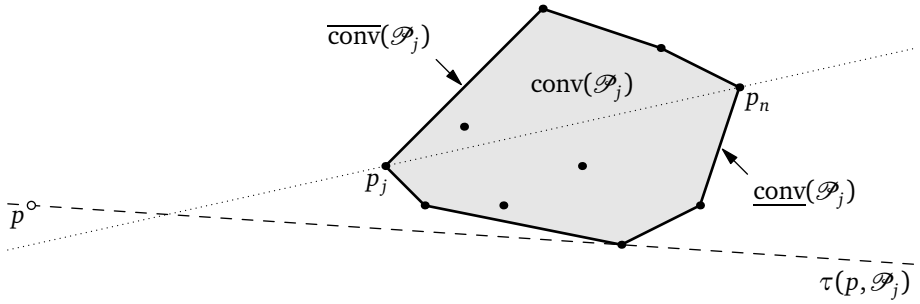


Figure 6.2: Convex hull of $\mathscr{P}_j$.

have any use for it here. In a special case, when the convex hull degenerates into a line segment (or even a point), the lower and upper hulls coincide.

Look again at Hong's bound. We have

$$\lg H = \max_{p_j \in \mathscr{N}_0} \min_{p_i \in \mathscr{P}_j} s_{ij}. \tag{6.9}$$

Thus, for any negative point p_j we are looking for a line of a minimal slope that passes through p_j and some positive point p_i to the right of p_j. It is easy to observe that such a line must "touch" the lower hull of $\mathscr{P}_j$. More precisely:

Definition 6.3.15. A line $L \subset \mathbb{R}^2$ is called a *lower tangent* of $\text{conv}(\mathscr{P}_j)$, if it passes through at least one point of $\mathscr{P}_j$ and every point $p_i \in \mathscr{P}_j$ lies either on or above L.

Given a point p lying to the left of $\mathscr{P}_j$, the unique lower tangent to $\mathscr{P}_j$ passing through p will be denoted $\tau(p, \mathscr{P}_j)$.

Observation 6.3.16. *If p is a negative point left of $\mathscr{P}_j$, then the lower tangent $\tau(p, \mathscr{P}_j)$ has the minimal slope among all the lines that pass through p and some positive point $p_i \in \mathscr{P}_j$.*

It follows from the above observation that the quantity $\lg H$, we are after, is just the maximal slope of the lower tangents $\tau(p_j, \mathscr{P}_j)$, where the maximum is taken over the set of all negative points. Given a lower tangent $\tau(p, \mathscr{P}_j)$ to the convex hull of $\mathscr{P}_j$, the left-most (i.e. the one with a minimal abscissa) point of $\tau(p, \mathscr{P}_j) \cap \mathscr{P}_j$ is called the *point of tangency* of $\tau(p, \mathscr{P}_j)$. It is obvious that the point of tangency belongs to the lower hull of $\mathscr{P}_j$.

Suppose that $\underline{\text{conv}}(\mathscr{P}_j)$ is a chain of points $(p_{i_1}, \ldots, p_{i_m})$ and the points are sorted with respect to their abscissæ. Let $p \in \mathbb{R}^2$ be a point positioned to the left of $\mathscr{P}_j$. The point p_{i_k} of tangency of $\tau(p, \mathscr{P}_j)$ is uniquely determined by the following two conditions:

- its left neighbor $p_{i_{k-1}}$ lies strictly above $\tau(p, \mathscr{P}_j)$, providing that $k > 1$;

- its right neighbor $p_{i_{k+1}}$ lies on or above $\tau(p, \mathscr{P}_j)$, providing that $k < m$.

Thus, the tangent point can be found by a direct linear search.

Algorithm 6.3.17. *Given a lower hull* $\operatorname{conv}(\mathscr{P}_j) = (p_{i_1}, \ldots, p_{i_m})$ *of* $\mathscr{P}_j$ *and a point* p *to the left of it, this algorithm finds the point of tangency of* $\tau(p, \mathscr{P}_j)$.

1. *For every index* $k \in \{1, \ldots, m\}$ *proceed as follows:*
   a) *if* $k > 1$*, then compute the determinant of a* 2×2 *matrix*
   $$a := \det \begin{pmatrix} p - p_{i_k} \\ p_{i_{k-1}} - p_{i_k} \end{pmatrix},$$
   otherwise set $a := -1$*;*
   b) *if* $k < m$*, then compute the determinant of a* 2×2 *matrix*
   $$b := \det \begin{pmatrix} p - p_{i_k} \\ p_{i_{k+1}} - p_{i_k} \end{pmatrix},$$
   otherwise set $b := -1$*;*
   c) *if* $a < 0$ *and* $b \leq 0$*, then* p_{i_k} *is the point of tangency of* $\tau(p, \mathscr{P}_j)$*, return it and quit.*

In the above algorithm, the differences $p - p_{i_k}$, $p_{i_{k-1}} - p_{i_k}$ and $p_{i_{k+1}} - p_{i_k}$ are treated as row vectors. The correctness of the algorithm follows from a well known geometric interpretation of the determinant. We let the reader provide the missing details. With the aim of Algorithm 6.3.17, it is quite easy to construct the lower hull of $\mathscr{P}_{j-1}$ from the lower hull of $\mathscr{P}_j$ and consequently to build all the lower hulls incrementally, starting from $\underline{\operatorname{conv}}(\mathscr{P}_n) = \{p_n\}$. Indeed, if p_{j-1} is a positive point, $\underline{\operatorname{conv}}(\mathscr{P}_j) = (p_{i_1}, \ldots, p_{i_m})$ and $p_{i_k} \in \underline{\operatorname{conv}}(\mathscr{P}_j)$ is a point of tangency of $\tau(p_{j-1}, \mathscr{P}_j)$, then

$$\underline{\operatorname{conv}}(\mathscr{P}_{j-1}) = (p_{j-1}, p_{i_k}, p_{i_{k+1}}, \ldots, p_{i_m})$$

In other words, one builds a lower hull of $\mathscr{P}_{j-1}$ by replacing the points of $\underline{\operatorname{conv}}(\mathscr{P}_j)$, that sit to the left of the found point of tangency, by p_{j-1}. It is said that one picture is worth a thousand words, hence let a picture speak. We refer the reader to Figure 6.3, to see why the above claim is true.

Observe that the "tail" $(p_{i_k}, p_{i_{k+1}}, \ldots, p_{i_m})$ of the lower hull $\underline{\operatorname{conv}}(\mathscr{P}_{j-1})$ is itself a lower hull $\underline{\operatorname{conv}}(\mathscr{P}_{i_k})$ of $\mathscr{P}_{i_k}$. This fact let us store **all** the lower hulls in a single array of indices, say V. For a positive point p_i, we set $V[i]$ to be the index of the point of tangency of $\tau(p_i, \mathscr{P}_{i+1})$. In other words, $V[i]$ is the index of the **second** point of the lower hull $\underline{\operatorname{conv}}(\mathscr{P}_i)$. Consequently, the lower hull of $\mathscr{P}_i$ equals

$$\underline{\operatorname{conv}}(\mathscr{P}_i) = \big(p_i, p_{V[i]}, p_{V[V[i]]}, \ldots, p_n\big).$$

This leaves the entries of V corresponding to negative points uninitialized. Assume that p_i is a negative point and let j be the index of the first positive point to the right of p_i, that is

$$j = \min\{k > i \mid f_k > 0\}.$$

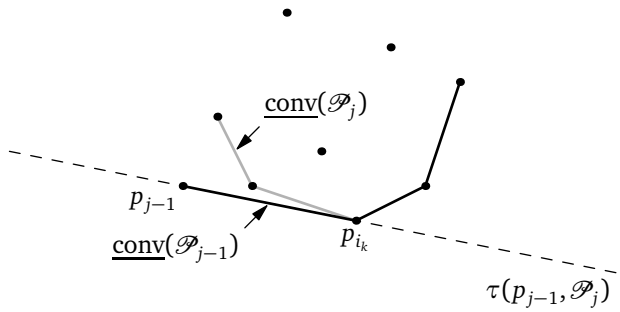


Figure 6.3: Construction of conv($\mathscr{P}_{j-1}$) from conv($\mathscr{P}_j$).

Then $\mathscr{P}_i = \mathscr{P}_j$ and so also conv($\mathscr{P}_i$) = conv($\mathscr{P}_j$). It will be convenient to set $V[i] := j$, in this case. In particular, for a negative point p_i, we have

$$\text{conv}(\mathscr{P}_i) = \big(p_{V[i]}, p_{V[V[i]]}, \ldots, p_n\big).$$

Algorithm 6.3.18. *Given a polynomial $f = f_0 + f_1 x + \cdots + f_n x^n$ with a positive leading coefficient, this algorithm construct an array V such that for every nonzero term f_i of f the lower hull of $\mathscr{P}_i$ has a form*

$$\text{conv}(\mathscr{P}_i) = \begin{cases} (p_i, p_{V[i]}, p_{V[V[i]]}, \ldots, p_n) & \text{if } f_i > 0 \\ (p_{V[i]}, p_{V[V[i]]}, \ldots, p_n) & \text{if } f_i < 0, \end{cases}$$

where $p_i = \big(i, -\lg|f_i|\big)$ for every $i \in \{0, \ldots, n\}$.

1. *Set $V[n] := -1$ and initialize the index of the last visited positive point to $k := n$;*
2. *iterate over nonzero coefficients of f in decreasing order of indices $i \in \{n-1, n-2, \ldots, 0\}$.*
   a) *if $f_i < 0$, then set $V[j] := k$;*
   b) *if $f_i > 0$, then*
      i. *execute Algorithm 6.3.17 with with input p_i and conv($\mathscr{P}_{i+1}$) to find the tangent point $p_j = \big(j, -\lg|f_j|\big)$ of $\tau(p_i, \mathscr{P}_{i+1})$;*
      ii. *set $V[i] := j$ and update k, setting $k := i$;*
3. *return the array V.*

The correctness of Algorithm 6.3.18 follows from the discussion preceding its presentation. We shall analyze the time complexity of this algorithm. At first sight it may seem that it works in $O(n^2)$ time since in step 2 we have a loop and inside this loop we execute Algorithm 6.3.17, which contains another loop. This looks like a double loop over points. But the quadratic complexity is only apparent. If we look carefully, we will notice that every point we stepped over in the inner loop is removed from the

lower hull. Hence, it will not be visited again. The only point that is accessed by two successive calls to Algorithm 6.3.17 is the point of tangency, returned by the first call. It is the point, where we terminated iteration first time and resumed it second time. This means that each time the inner loop is executed it iterates over a different chain of points. Thus we have shown:

Observation 6.3.19. *Algorithm 6.3.18 runs in $O(n)$ time.*

We are now ready to show how to compute the Hong's bound, knowing all the lower hulls. Recall (see Observation 6.3.16 and Eq. (6.9)) that we are looking for the maximal slope of lower tangents $\tau(p_j, \mathscr{P}_j)$ for all negative points p_j. Scan the negative points left-to-right (i.e. scan the negative coefficients of f in increasing order of indices). Let $p_j = \big(j, -\lg|f_j|\big)$ be a negative point. Denote the slope of $\tau(p_j, \mathscr{P}_j)$ by s_j, hence

$$s_j = \min_{p_i \in \mathscr{P}_j} s_{ij} = \min_{\substack{i>j \\ f_i>0}} \frac{\lg|f_j| - \lg|f_i|}{i-j}.$$

Assume that we have already found a point p_k, $k < j$ such that the slope of the lower tangent $\tau(p_k, \mathscr{P}_k)$ is maximal among all the points to the left of the current point p_j. Let $t_k := p_m = \big(m, -\lg|f_m|\big)$ be the point of tangency of $\tau(p_k, \mathscr{P}_k)$. We need to consider two cases.

First suppose that $m < j$, then t_k cannot belong to the lower tangent of $\mathscr{P}_j$. In this case we just construct the lower tangent $\tau(p_j, \mathscr{P}_j)$ and compare its slope s_j with s_k. If $s_j > s_k$, then we got a new candidate for the maximum. Otherwise, we keep s_k. Observe that, in order to find the lower tangent, we had to scan only the positive points lying to the right of p_j and consequently also to the right of t_k.

Secondly, suppose that t_k lies to the right of p_j (i.e. $m > j$, it cannot be equal since p_j is negative and $t_k = p_m$ is positive). Then t_k must belong to the lower hull $\underline{\mathrm{conv}}(\mathscr{P}_m) \subseteq \underline{\mathrm{conv}}(\mathscr{P}_j)$. Observe that, if the current point p_j is on or above the lower tangent $\tau(p_k, \mathscr{P}_k) = \tau(p_k, \mathscr{P}_j)$, then the slope of the lower tangent $\tau(p_j, \mathscr{P}_j)$ is necessarily less or equal s_k. Therefore, we may simply step over the point p_j, ignoring it. Otherwise, if p_j lies below $\tau(p_k, \mathscr{P}_j)$, then the point of tangency t_j of the lower tangent $\tau(p_j, \mathscr{P}_j)$ cannot lie to the left of t_k. Thus, one finds t_j scanning the lower hull $\underline{\mathrm{conv}}(\mathscr{P}_m)$. This means that, we again examine only the positive points to the right of t_k. The possible configurations of the lower tangents are illustrated in Figure 6.4. We may now summarize our discussion in form of an algorithm.

Algorithm 6.3.20. *Given a polynomial $f = f_0 + f_1 x + \cdots + f_n x^n$, this algorithm computes the Hong's upper bound for real roots of f. In what follows, we denote $p_i := \big(i, -\lg|f_i|\big)$ for $i \in \{0, \ldots, n\}$.*

1. *If all the coefficients of f has the same sign, then return 0 and quit;*
2. *if the leading coefficient of f is negative, then replace f by $-f$;*
3. *using Algorithm 6.3.18 construct all the lower hulls $\underline{\mathrm{conv}}(\mathscr{P}_j)$ for $j \in \{0, \ldots, n\}$, store them in and array V;*

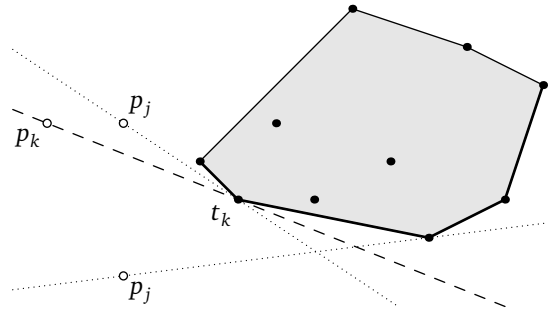


Figure 6.4: Possible configurations of the lower tangents $\tau(p_k, \mathscr{P}_j)$ and $\tau(p_j, \mathscr{P}_j)$.

4. *let k be the lowest index of a negative coefficients of f;*
5. *using Algorithm 6.3.17 find the point of tangency $t_k := p_m = \big(m, -\lg|f_m|\big)$ of $\tau(p_k, \mathscr{P}_k)$ and set s_k to be the slope of the lower tangent $\tau(p_k, \mathscr{P}_k)$;*
6. *for every $j \in \{k+1, \dots, n-1\}$ if f_j is negative, do the following:*
   a) *if the point of tangency t_k lies to the left of p_j (i.e. $m < j$), then execute Algorithm 6.3.17 with input data p_j and $\underline{\mathrm{conv}}(\mathscr{P}_j) = (p_{V[j]}, p_{V[V[j]]}, \dots, p_n)$ to find the point of tangency t_j of the lower tangent $\tau(p_j, \mathscr{P}_j)$, compare the slopes s_j, s_k if $s_j > s_k$, then update:*

$$k := j, \qquad p_k := p_j, \qquad s_k := s_j$$

   and reiterate the loop;
   b) *if the point of tangency $t_k = \big(m, -\lg|f_m|\big)$ lies to the right of p_j, then compute the determinant*

$$d := \det \begin{pmatrix} p_k - t_k \\ p_j - t_k \end{pmatrix}$$

   to determine if p_j lies under $\tau(p_k, \mathscr{P}_j)$;
   c) *if $d < 0$ (i.e. p_j lies under $\tau(p_k, \mathscr{P}_j)$), then execute Algorithm 6.3.17 with input data p_j and $\underline{\mathrm{conv}}(\mathscr{P}_m) = (p_m, p_{V[m]}, p_{V[V[m]]}, \dots, p_n)$ to find the point of tangency t_j of the lower tangent $\tau(p_j, \mathscr{P}_j) = \tau(p_j, \mathscr{P}_m)$;*
   d) *update $k := j$, $p_k := p_j$ and $s_k := s_j$;*
7. *return the Hong's bound $2H = 2^{1+s_k}$.*

Laguerre–Samuelson inequality

So far we have been working with general polynomials. It should not come with a big surprise that enforcing some additional conditions, we may obtain better bounds on the roots. In particular, if the polynomial f has only real roots, we can localize

roots using only three highest coefficients of f. The next result was first proved by E. Laguerre[11] in 1880 and independently rediscovered by A. Samuelson[12] in 1968.

Theorem 6.3.21 (Laguerre–Samuelson inequality). *Let $f = f_0 + f_1 x + \cdots + f_n x^n$ be a polynomial of degree $\deg f \geq 2$. Assume that all the roots of f are real. Then the roots of f are contained in the interval $[M_-, M_+]$, where*

$$M_\pm := -\frac{f_{n-1}}{n \cdot f_n} \pm \frac{n-1}{n} \cdot \sqrt{\frac{f_{n-1}^2}{f_n^2} - \frac{2n \cdot f_{n-2}}{(n-1) \cdot f_n}}$$

Proof. Let $\xi_1, \ldots, \xi_n \in \mathbb{R}$ be all the roots of f. Denote $A := \sum_{i=1}^n \xi_i$ and $B := \sum_{i=1}^n \xi_i^2$. Express A and B in terms of the coefficients of f using Viète's formulæ.

$$A = -\frac{f_{n-1}}{f_n} \qquad \text{and} \qquad B = \left(\sum_{i=1}^n \xi_i\right)^2 - 2 \cdot \sum_{1 \leq i < j \leq n} \xi_i \xi_j = \left(\frac{f_{n-1}}{f_n}\right)^2 - 2 \cdot \frac{f_{n-2}}{f_n}. \tag{6.10}$$

Fix an index $j \in \{1, \ldots, n\}$. For every real number x, one has

$$(x - \xi_j)^2 \leq \sum_{i=1}^n (x - \xi_i)^2.$$

Expand both sides to obtain

$$x^2 - 2x\xi_j + \xi_j^2 \leq \sum_{i=1}^n \bigl(x^2 - 2x\xi_i + \xi_i^2\bigr) = nx^2 - 2xA + B.$$

Rearranging the terms we write

$$(n-1) \cdot x^2 + 2 \cdot (\xi_j - A) \cdot x + (B - \xi_j^2) \geq 0.$$

Now, the quadratic polynomial can be non-negative for every $x \in \mathbb{R}$ only when its discriminant is non-positive, hence

$$4 \cdot \bigl((\xi_j - A)^2 - (n-1)(B - \xi_j^2)\bigr) \leq 0.$$

This yields an inequality for ξ_j:

$$n \cdot \xi_j^2 - 2A \cdot \xi_j + A^2 - (n-1) \cdot B \leq 0.$$

This is again a quadratic inequality. A root ξ_j of f satisfies it if and only if it lies in the interval $[M_-, M_+]$, where $M_\pm$ are the roots of the quadratic polynomial $nx^2 - 2Ax + (A^2 - (n-1) \cdot B)$. Finding the roots we get:

$$M_\pm = \frac{2A \pm \sqrt{4 \cdot (n-1) \cdot (nB - A^2)}}{2n} = \frac{A}{n} \pm \frac{\sqrt{n-1}}{n} \cdot \sqrt{nB - A^2}. \tag{6.11}$$

[11] Edmond Nicolas Laguerre (born 9 April 1834, died 14 August 1886) was a French mathematician and a member of the Académie Française.

[12] Paul Anthony Samuelson (born 15 May 1915, died 13 December 2009) was an American economist and Nobel laureate.

Using Eq. (6.10) we may express the last formula in terms of the coefficients of f:

$$M_{\pm} = -\frac{f_{n-1}}{n \cdot f_n} \pm \frac{n-1}{n} \cdot \sqrt{\frac{f_{n-1}^2}{f_n^2} - \frac{2n \cdot f_{n-2}}{(n-1) \cdot f_n}} \qquad \square$$

Although Laguerre–Samuelson inequality may seem a bit intimidating at first sight, it is easy to interpret it. Corollary 6.1.7 asserts that the first term is just the mean μ_f of the roots. And we claim that the second terms equals $\sqrt{n-1} \cdot \sigma_f$, where σ_f denotes the standard deviation of the roots of f. Denote $A := \sum_{i=1}^{n} \xi_i$ and $B := \sum_{i=1}^{n} \xi_i^2$, as we did in the above proof. We have

$$\sigma_f = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\xi_i - \mu_f)^2} = \sqrt{\frac{1}{n}\Big(\sum_{i=1}^{n}\xi_i^2 - n \cdot \mu_f^2\Big)} = \frac{1}{n} \cdot \sqrt{nB - A^2}.$$

Our claim follows now immediately from Eq. (6.11). All in all, Laguerre–Samuelson inequality can be rephrased as

$$\mu_f - \sqrt{n-1} \cdot \sigma_f \leq \xi_i \leq \mu_f + \sqrt{n-1} \cdot \sigma_f \qquad \text{for every } i \in \{1, \ldots, n\}.$$

For depressed polynomials Laguerre–Samuelson inequality simplifies considerably:

Corollary 6.3.22. *Let $f = f_0 + f_1 x + \cdots + f_{n-2}x^{n-2} + x^n$ be a depressed polynomial of degree $\deg(f) \geq 2$. If all the roots of f are real, then they lie in an interval $[-M, M]$, where*

$$M = \sqrt{\frac{2-2n}{n} \cdot f_{n-2}}.$$

The assumption that all the roots of f must be real is indispensable. Indeed, take for instance $f = x^3 - 1$. It has three roots: $1, \cos\frac{2\pi}{3} \pm i \cdot \sin\frac{2\pi}{3}$, all lying on the unit circle. Yet, the formula in the above corollary evaluates to zero.

Remark 49. It should be noted that the Laguerre–Samuelson uses only three leading coefficients of a polynomial, hence its time complexity is constant, independent of the degree of a polynomial. It is yet another feature that distinguishes it from other bounds discussed in this section, that are all linear in number of terms of a polynomial.

For an extensive discussion on Laguerre–Samuelson result and its various applications, plus a number of different proofs, we refer the reader to [35].

Example 73. Take again $f = x^5 - 20x^4 + 152x^3 - 544x^2 + 900x - 529$. All the roots of f are real, hence we may apply Theorem 6.3.21, which provides an upper bound for roots of f of the form:

$$M_+(f) = -\frac{(-20)}{5 \cdot 1} + \frac{4}{5} \cdot \sqrt{\frac{(-20)^2}{1^2} - \frac{10 \cdot 152}{4 \cdot 1}} = \frac{8}{5}\sqrt{5} + 4 \approx 7.58.$$

The corresponding lower bound is $M_f(f) = -\frac{8}{5}\sqrt{5}+4 \approx 0.42$. Notice that Laguerre–Samuelson inequality intrinsically takes care of the means of the roots. Consequently, for this estimation Observation 6.3.2 yields exactly the same result. Indeed, the depressed form of f is $g = x^5 - 8x^3 + 4x - 1$. Since all the roots of f are real, so are the roots of g. The roots of g are bounded from above (see Corollary 6.3.22) by

$$M_g = \sqrt{\frac{2-10}{5} \cdot 152} = \frac{8}{5}\sqrt{5} \approx 3.58$$

and $M_g \pm \mu_f = M_\pm(f)$.

Fujiwara bound, Cohen, Alan M., "Bounds for the roots of polynomial equations", Mathematical Gazette 93, March 2009

6.4 Distances between roots

This subsection will be completely rewritten
Dandelin–Graeffe root squaring, Hong 2018 (?)

In the previous section we investigated the problem how big (or small) a root of a polynomial can be. Another related question is how close to each other two roots of a given polynomial can lie. Let again $f = f_0 + f_1 x + \cdots + f_n x^n$ be a polynomial with coefficients in $\mathbb{C}$ or its subring (a particularly important case is $f \in \mathbb{Z}[x]$). We want to estimate the minimal distance between roots of f. In order to be able to talk about distance between roots, f must have at least two roots. Therefore in this section we will always assume that $\deg f = n \geq 2$. Let $\xi_1, \ldots, \xi_n \in \mathbb{C}$ be all the roots of f. Denote the minimal distance between two distinct complex roots of f by $\operatorname{sep}(f)$, i.e.:

$$\operatorname{sep}(f) := \min\bigl\{|\xi_i - \xi_j| \mid \xi_i \neq \xi_j j\bigr\},$$

with the usual convention that the minimum of an empty set equals infinity. Similarly, by $\operatorname{rsep}(f)$ we will denote the minimal distance between two distinct **real** roots of f:

$$\operatorname{rsep}(f) := \min\bigl\{|\xi_i - \xi_j| \mid \xi_i, \xi_j \in \mathbb{R},\ \xi_i \neq \xi_j\bigr\}.$$

It is obvious that $\operatorname{rsep}(f) \geq \operatorname{sep}(f)$ for every f. Recall that the 1-norm of a vector is the sum of absolute values of its coefficients. Applying this notion to the vector of coefficients of f we write

$$\|f\| = \sum_{i=1}^{n} |f_i|.$$

We need to introduce also one more quantity:

Definition 6.4.1. Let $\xi_1, \ldots, \xi_n \in \mathbb{C}$ be all the roots of the polynomial f. Then

$$m_f := |\operatorname{lc}(f)| \cdot \prod_{|\xi_i|>1} |\xi_i|$$

is called the *Mahler measure* of f

The following lemma establishes a link between the 1-norm and Mahler measure.

Lemma 6.4.2. *If $f \in \mathbb{C}[x]$ is a non-constant polynomial, then $m_f \leq \|f\|$.*

This lemma is based on Jensen's formula and so its proof would take us too far afield. Hence we take the liberty to omit the proof, instead referring an interested reader to the original paper [55].

Consider now the Vandermonde matrix (for a definition see Section 3.8) associated with the roots of f:

$$V_{\xi_1,\ldots,\xi_n} := \begin{pmatrix} 1 & \xi_1 & \xi_1^2 & \cdots & \xi_1^{n-1} \\ 1 & \xi_2 & \xi_2^2 & \cdots & \xi_2^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \xi_n & \xi_n^2 & \cdots & \xi_n^{n-1} \end{pmatrix}$$

Theorem 3.8.3 asserts that the determinant of the Vandermonde matrix can be computed as a product

$$\det V_{\xi_1,\ldots,\xi_n} = \prod_{i<j} (\xi_j - \xi_i). \tag{6.12}$$

Denote

$$\Delta_f := \left(\mathrm{lc}(f)^{(n-1)} \cdot \det(V_{\xi_1,\ldots,\xi_n})\right)^2. \tag{6.13}$$

This is called the *discriminant* of f. It follows from Eq. (6.12) that Δ_f is a symmetric polynomial in $\xi_1, \ldots, \xi_n$, hence by Viète's formulas it can be expressed in terms of coefficients of f. In Section 8.3 we discuss this notion in depth. In particular we present a couple of methods for computing the discriminant of a polynomial, without knowing its roots. We are now ready to show the first estimate for the distance of the roots of a polynomial.

Theorem 6.4.3 (Mahler, 1964)**.** *Let f be a square-free polynomial of degree $n \geq 2$, then*

$$\mathrm{sep}(f) > \frac{\sqrt{3|\Delta_f|}}{n^{\frac{n+2}{2}} \cdot \|f\|^{n-1}}.$$

Proof. Let $\xi, \zeta \in \mathbb{C}$ be the two roots of f satisfying $\mathrm{sep}(f) = |\xi - \zeta|$. Without loss of generality we may assume that $|\xi| \geq |\zeta|$. We should consider two cases. First assume that $|\xi| > 1$, i.e. ξ lies outside the (closed) unit disc $D := \{z \in \mathbb{C} \mid |z| \leq 1\}$. Rearrange the order of roots of f in the following manner:

$$\xi_1 := \xi, \quad |\xi_2| > 1, \ldots, |\xi_m| > 1 \qquad \text{and} \qquad |\xi_{m+1}| \leq 1, \ldots, |\xi_n| \leq 1.$$

As above, let $V_{\xi_1,\ldots,\xi_n}$ be the Vandermonde matrix associated with the roots of f. From the first row of $V_{\xi_1,\ldots,\xi_n}$ subtract the one corresponding to ζ. Subtraction of rows does

not alter the determinant, hence we have

$$\det V_{\xi_1,\dots,\xi_n} = \det \begin{pmatrix} 0 & \xi-\zeta & \xi^2-\zeta^2 & \cdots & \xi^{n-1}-\zeta^{n-1} \\ 1 & \xi_2 & \xi_2^2 & \cdots & \xi_2^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \xi_n & \xi_n^2 & \cdots & \xi_n^{n-1} \end{pmatrix}.$$

Divide the rows of $V_{\xi_1,\dots,\xi_n}$ corresponding to the roots of f not in D (i.e. the first m rows) by $\xi_1^{n-1},\dots,\xi_m^{n-1}$, respectively. Denote the resulting determinant by Q. We have

$$Q = \frac{\det V_{\xi_1,\dots,\xi_n}}{(\xi_1\cdots\xi_m)^{n-1}} = \det \begin{pmatrix} 0 & \frac{\xi-\zeta}{\xi^{n-1}} & \cdots & \frac{\xi^{n-2}-\zeta^{n-2}}{\xi^{n-1}} & \frac{\xi^{n-1}-\zeta^{n-1}}{\xi^{n-1}} \\ \xi_2^{-(n-1)} & \xi_2^{-(n-2)} & \cdots & \xi_2^{-1} & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ \xi_m^{-(n-1)} & \xi_m^{-(n-2)} & \cdots & \xi_m^{-1} & 1 \\ 1 & \xi_{m+1} & \cdots & \xi_{m+1}^{n-2} & \xi_{m+1}^{n-1} \\ \vdots & \vdots & & \vdots & \vdots \\ 1 & \xi_n & \cdots & \xi_n^{n-2} & \xi_n^{n-1} \end{pmatrix}.$$

Finally, divide the first row of the above matrix by $\xi-\zeta$ and denote the emerging entries by $q_0,\dots,q_{n-1}$. This way we have

$$\frac{Q}{\xi-\zeta} = \det \begin{pmatrix} q_0 & q_1 & \cdots & q_{n-2} & q_{n-1} \\ \xi_2^{-(n-1)} & \xi_2^{-(n-2)} & \cdots & \xi_2^{-1} & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ \xi_m^{-(n-1)} & \xi_m^{-(n-2)} & \cdots & \xi_m^{-1} & 1 \\ 1 & \xi_{m+1} & \cdots & \xi_{m+1}^{n-2} & \xi_{m+1}^{n-1} \\ \vdots & \vdots & & \vdots & \vdots \\ 1 & \xi_n & \cdots & \xi_n^{n-2} & \xi_n^{n-1} \end{pmatrix}, \tag{6.14}$$

where $q_0 = 0$ and

$$q_k = \frac{\xi^k-\zeta^k}{(\xi-\zeta)\cdot\xi^{n-1}} = \frac{\xi^{k-1}}{\xi^{n-1}} + \frac{\xi^{k-2}\cdot\zeta}{\xi^{n-1}} + \cdots + \frac{\zeta^{k-1}}{\xi^{n-1}}$$

for $k \geq 1$. All the entries of the above matrix, except possibly the ones in the first row, lie in D. Therefore the Hadamard's inequality (Theorem 3.4.21), applied to the transposed matrix, yields

$$\left|\frac{Q}{\xi-\zeta}\right|^2 \leq \left(\sum_{i=0}^{n-1}|q_i|^2\right)\cdot\left(\prod_{j=2}^{n}\left(\sum_{i=1}^{n}1^2\right)\right) = n^{n-1}\cdot\left(\sum_{i=0}^{n-1}|q_i|^2\right).$$

From the assumption $|\zeta| \leq |\xi|$ we infer that

$$\left|\frac{\xi^{k-i}\zeta^i}{\xi^{n-1}}\right| \leq \left|\frac{\xi^k}{\xi^{n-1}}\right| \leq 1 \qquad \text{for every } k \leq n-1.$$

Thus, directly from the definition of q_k we obtain

$$|q_k|^2 \le (1+\cdots+1)^2 \le k^2.$$

Consequently we have

$$\left|\frac{Q}{\xi-\zeta}\right|^2 \le n^{n-1}\cdot\sum_{i=0}^{n-1} i^2 = n^{n-1}\cdot\frac{n\cdot(n-1)\cdot(2n-1)}{6} < n^{n-1}\cdot\frac{n^3}{3} = \frac{n^{n+2}}{3}. \tag{6.15}$$

Recall that $\det V_{\xi_1,\dots,\xi_n} = (\xi_1\cdots\xi_m)^{n-1}\cdot Q$. Together with the definition of the Mahler measure and Eq. (6.13), this yields

$$|\Delta_f| = |f_n|^{2n-2}\cdot\left|(\xi_1\cdots\xi_m)^{n-1}\cdot Q\right|^2 = m_f^{2n-2}\cdot|Q|^2. \tag{6.16}$$

Combine (6.15) and (6.16) to get

$$|\xi-\zeta| > \frac{\sqrt{3|\Delta_f|}}{n^{\frac{n+2}{2}}\cdot m_f^{n-1}}.$$

Lemma 6.4.2 asserts that $m_f \le \|f\|$ and this concludes the proof in the case when $|\xi|>1$.

In the case $|\xi|\le 1$ the proof runs along the same lines except that—when computing Q—we do not divide the first row of the matrix by ξ^{n-1}. It follows that the elements $q_0,\dots,q_{n-1}$ in Eq. (6.14) are given by the formula

$$q_0 = 0 \qquad \text{and} \qquad q_k = \frac{\xi^k-\zeta^k}{\xi-\zeta} = \xi^{k-1}+\xi^{k-2}\zeta+\cdots+\zeta^{k-1}.$$

Again we have $|q_k|\le k$ for every $k\ge 0$. Like in the previous case we obtain

$$|\xi-\zeta| > \frac{\sqrt{3|\Delta_f|}}{n^{\frac{n+2}{2}}\cdot m_f^{n-1}} \ge \frac{\sqrt{3|\Delta_f|}}{n^{\frac{n+2}{2}}\cdot\|f\|^{n-1}}. \qquad \square$$

Later we will learn (see Corollary 8.3.7) that the absolute value of discriminant of a square-free polynomial with integer coefficients is at least 1. This leads to the following estimate:

Corollary 6.4.4. *If $f\in\mathbb{Z}[x]$ is a square-free polynomial of degree $n\ge 2$, then*

$$\operatorname{sep}(f) > \frac{\sqrt{3}}{n^{\frac{n+2}{2}}\cdot\|f\|^{n-1}}.$$

Example 74. Take a polynomial

$$f = x^5 - x^3 - 1000x^2 + 10.$$

The roots of f—obtained by methods discussed in later sections—are approximately equal:

$$-0.10, 0.10, 10., -5.0 - 8.5i, -5.0 + 8.5i.$$

It follows that $\operatorname{sep}(f) \approx 0.20$. On the other hand, we have

$$\|f\| = |10| + |-1000| + |-1| + |1| = 1012.$$

Thus Mahler's inequality (Corollary 6.4.4) yields the following estimate for $\operatorname{sep}(f)$:

$$\operatorname{sep}(f) > \frac{\sqrt{3}}{5^{\frac{7}{2}} \cdot 1012^4} \approx 5.8 \times 10^{-15}.$$

This estimate is definitely true, nevertheless quite pessimistic.

Theorem 6.4.5 (Rump). *Let $f = f_0 + f_1 x + \cdots + f_n x^n$ be a polynomial with integer coefficients, then*

$$\operatorname{rsep}(f) > \frac{2\sqrt{2}}{n^{\frac{n+2}{n}} \cdot \left(1 + \|f\|_1\right)^n}.$$

The proof of this theorem can be found in [75].

Bibliographic notes Like the bounds for the roots, the problem of their minimal distance can also be traced back essentially to Cauchy (see e.g. [16, Ineq. (48), p. 398]). Lemma 6.4.2 comes from [55], but is sometimes accredited to E. Landau. The proof of Theorem 6.4.3 is based on [56], while Theorem 6.4.5 comes from [75] Recently Herman, Hong and Tsigaridas in [32] improved Mahler's result finding a formula, which is well-behaved with respect to a linear scaling of roots. Some other formulas were developed in [14, 61, 62]. Beside distances between isolated roots, another problem being investigated is a bound for a distance between *clusters* of roots. We refer the reader to [15] for a survey of recent results.

6.5 Root counting and isolating

In this section we concentrate on a problem of determining how many roots a polynomial has in a given interval (possibly unbounded). Of course, by the Fundamental Theorem of Algebra (Theorem 6.1.1) we know, how many zeros a polynomial has in $\mathbb{C}$. But it is often desirable to know how many roots it has in some subset of $\mathbb{C}$. In particular how many roots are real.

Sturm theorem

We begin with a classical result due to J. Sturm[13]. For a finite sequence $A = (a_0, \ldots, a_n)$ of real numbers, we denote the number of sign changes **ignoring zeros** within this

[13] Jacques Charles François Sturm (born 29 September 1803, died 15 December 1855) was a French mathematician.

sequence by var(A). In particular, if $\mathscr{F} := (f_0, \dots, f_n)$ is a sequence of real polynomials and $a \in \mathbb{R}$, then var($\mathscr{F}(a)$) is the number of sign changes in $\big(f_0(a), f_1(a), \dots, f_n(a)\big)$.

Definition 6.5.1. Let f be a square-free polynomial with coefficients in a subfield of $\mathbb{R}$. The *Sturm sequence* of f is a finite sequence $(s_0, s_1, \dots, s_k)$ of polynomials of decreasing degree defined by the following recursive formula. Let $s_0 := f$ and let s_1 be the derivative of f. For $i > 1$, take $s_i := -r$, where r is the remainder of the division of s_{i-2} by s_{i-1}, as long as s_i is not zero. That is:

$$s_0 := f,\ s_1 := f', \dots,\ s_i = -(s_{i-2} \bmod s_{i-1}), \dots,\ s_k = \text{const}.$$

It is obvious that the Sturm sequence of a polynomial is computed using just a slight modification of Euclid's algorithm.

Algorithm 6.5.2. *Given a square-free polynomial f, this algorithm returns the Sturm sequence of f.*

1. *Set $s_0 := f$ and $s_1 := f'$;*
2. *initialize $k := 1$;*
3. *as long as s_k is not a constant proceed as follows:*
   a) *use Algorithm 1.1.12 to compute the remainder r of the division s_{k-1} by s_k;*
   b) *increment k and set $s_k := -r$;*
4. *return $(s_0, \dots, s_k)$.*

Sturm with pseudo-division and subresultants

Example 75. Take a polynomial $f = x^4 - 4x^3 + 2$ and compute the Sturm sequence of f. By definition

$$s_0 = f = x^4 - 4x^3 + 2 \qquad \text{and} \qquad s_1 = f' = 4x^3 - 12x^2,$$

while the remaining elements are:

$$s_2 = 3x^2 - 2, \quad s_3 = -\frac{8}{3}x + 8,$$
$$s_4 = -25.$$

The Sturm sequence provides us with a mean of computing a number of real solutions to a univariate polynomial equation (and also to some multivariate systems, as we shall see in Chapter 8)

Shape Lemma

Theorem 6.5.3 (Sturm, 1829)**.** *Let f be a square-free polynomial, $a, b \in \mathbb{R} \cup \{\pm\infty\}$ with $a < b$ and let $\mathscr{S} = (s_0, \dots, s_k)$ denote the Sturm sequence of f. Assume that $f(a) \neq 0 \neq f(b)$, then f has precisely*

$$\text{var}(\mathscr{S}(a)) - \text{var}(\mathscr{S}(b))$$

roots in the interval (a, b).

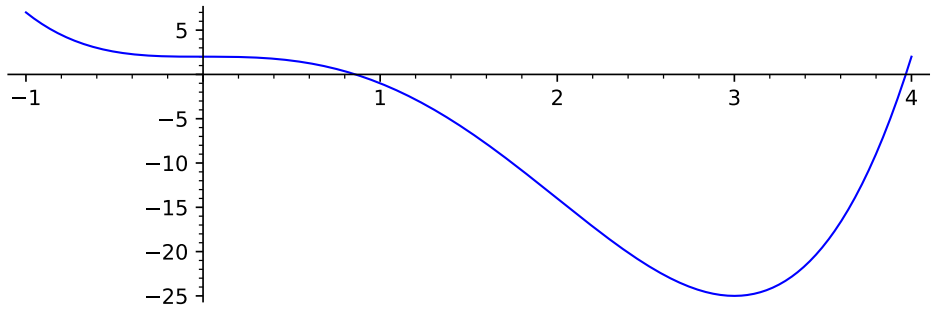


Figure 6.5: Plot of $f = x^4 - 4x^3 + 2$.

Remark 50. The above one is a somehow weakened version of Sturm's theorem. In fact one can omit the assumption that f is square-free and $f(a), f(b) \neq 0$ can be omitted, assuming only that a and b are not **multiple** roots of f. Then $\operatorname{var}(\mathscr{S}(a)) - \operatorname{var}(\mathscr{S}(b))$ is the number of distinct roots in a half-open interval $(a, b]$.

Proof. Let $\mathscr{S} = (s_0, \ldots, s_k)$ be the Sturm sequence of f. We claim that no two successive elements of this sequence share a common root. Suppose a contrario, that there are: $\alpha \in \mathbb{C}$ and $i \in \{0, \ldots, k-1\}$ such that

$$s_i(\alpha) = s_{i+1}(\alpha) = 0.$$

First suppose that $i > 0$, hence $s_{i+1} = -(s_{i-1} \bmod s_i)$. In particular, there exists q_i such that $s_{i-1} = q_i \cdot s_i - s_{i+1}$. Evaluating both sides at α, we obtain

$$s_{i-1}(\alpha) = q_i(\alpha) \cdot s_i(\alpha) - s_{i+1}(\alpha) = 0.$$

Thus, any α that is a common root of s_i and s_{i+1} must also be a root of s_{i-1}. By a simple induction, it follows that α is a common root of $s_0 = f$ and $s_1 = f'$. But this is impossible since f is square-free by the assumption. This contradiction proves our claim.

Polynomial functions are continuous and so they can change sign only at their roots. Therefore the function $\operatorname{var}_{\mathscr{S}} : x \mapsto \operatorname{var}(\mathscr{S}(x))$ is piecewise constant with discontinuities at most at roots of s_i's. We shall show that such discontinuities (i.e. value changes of $\operatorname{var}_{\mathscr{S}}$) occur only at the roots of s_0, with the roots of the remaining polynomials having no impact on $\operatorname{var}_{\mathscr{S}}$. This is trivially true for s_k, since it is constant. Now, let $\alpha \in (a, b)$ be a root of s_j for some $j \in \{1, \ldots, k-1\}$. We already know that neither s_{j-1} nor s_{j+1} may vanish at α. By the very definition, there is a polynomial q_j such that $s_{j-1} = q_j \cdot s_j - s_{j+1}$. Therefore, $s_{j-1}(\alpha) = -s_{j+1}(\alpha) \neq 0$. In particular, evaluating $\mathscr{S}$ at α we see that a sing change occurs at $j-1/j+1$:

$$\mathscr{S}(\alpha) = (\ldots, \pm, 0, \mp, \ldots).$$

Polynomials are continuous, hence there is a neighborhood $(\alpha-\varepsilon,\alpha+\varepsilon)$ of α on which both s_{j-1} and s_{j+1} remain nonzero and s_j is monotonous. There are four possible cases: s_j can be either increasing or decreasing at a vicinity of α and s_{j-1} can be either positive or negative. Suppose, for example, that s_j is increasing and s_{j-1} is negative. Then for any $\beta\in(\alpha-\varepsilon,\alpha)$, we have

$$\mathscr{S}(\beta)=(\dots,-,-,+,\dots),$$

for $\beta=\alpha$ we have

$$\mathscr{S}(\alpha)=(\dots,-,0,+,\dots),$$

while for $\beta\in(\alpha,\alpha+\varepsilon)$ we have

$$\mathscr{S}(\beta)=(\dots,-,+,+,\dots).$$

Thus, in all three sequences the sign change between $j-1$ and $j+1$ prevails. The only thing that changes is the position, that shifts from $j-1/j$, through $j-1/j+1$, to $j/j+1$, while passing through α. It follows that $\operatorname{var}_{\mathscr{S}}$ remains constant and α has no impact on it. The three remaining cases are fully analogous. All in all, $\operatorname{var}_{\mathscr{S}}$ depends entirely on the roots of $s_0=f$.

Take now a root α of f and let a_1,b_1 be two numbers such that: $a\le a_1<\alpha<b_1\le b$ and α is the only root of f in the interval (a_1,b_1). Moreover assume that f' has no root in this interval. (In fact, this last condition already implies that f has at most one root in (a_1,b_1).) Suppose that $s_1(\alpha)=f'(\alpha)>0$, hence f is increasing on (a_1,b_1). Thus, evaluating the Sturm sequence at both endpoints we have

$$\mathscr{S}(a_1)=(+,-,\dots)\qquad\text{and}\qquad\mathscr{S}(b_1)=(+,+,\dots).$$

Now, $\operatorname{var}\big(s_2(a_1),\dots,s_k(a_1)\big)=\operatorname{var}\big(s_2(b_1),\dots,s_k(b_1)\big)$ by the previous part of the proof. It follows that $\operatorname{var}(\mathscr{S}(a_1))-\operatorname{var}(\mathscr{S}(b_1))=1$ is precisely the number of roots of f on (a_1,b_1). The same holds true, when $f'(\alpha)<0$.

To conclude the proof, assume that f has precisely m roots $\alpha_1,\dots,\alpha_m$ in (a,b). Let $a\le a_1<b_1\le a_2<b_2\le\dots\le a_m<b_m\le b$ be such that there is precisely one root of f and no roots of f' between every a_i and b_i. We have already proved that $\operatorname{var}(\mathscr{S}(a_i))=1+\operatorname{var}(\mathscr{S}(b_i))$ for all i. Moreover, since only the roots of f have any impact on $\operatorname{var}_{\mathscr{S}}$, hence $\operatorname{var}(\mathscr{S}(b_i))=\operatorname{var}(\mathscr{S}(a_{i+1}))$ for $0\le i<m$, also $\operatorname{var}(\mathscr{S}(a))=\operatorname{var}(\mathscr{S}(a_0))$ and $\operatorname{var}(\mathscr{S}(b_m))=\operatorname{var}(\mathscr{S}(b))$. All in all

$$\begin{aligned}&\operatorname{var}(\mathscr{S}(a))-\operatorname{var}(\mathscr{S}(b))\\&=\big(\operatorname{var}(\mathscr{S}(a_1))-\operatorname{var}(\mathscr{S}(b_1))\big)+\big(\operatorname{var}(\mathscr{S}(a_2))-\operatorname{var}(\mathscr{S}(b_2))\big)+\dots+\big(\operatorname{var}(\mathscr{S}(a_m))-\operatorname{var}(\mathscr{S}(b_m))\big)=m.\end{aligned}$$ □

Example 76 (continued). Take the same polynomial $f=x^4-4x^3+2$ as in the previous example (for a plot of f see Figure 6.5). We have already computed the Sturm sequence $\mathscr{S}$ of f. Substitute $a=-\infty$ and $b=\infty$ into it and count the numbers of sign changes. At the minus infinity, there are precisely

$$\operatorname{var}\big(\mathscr{S}(-\infty)\big)=\operatorname{var}(+\infty,-\infty,+\infty,+\infty,-25)=3,$$

sign changes, while at the plus infinity there occur none:

$$\operatorname{var}\big(\mathscr{S}(\infty)\big) = \operatorname{var}(+\infty, +\infty, +\infty, -\infty, -25) = 1.$$

It follows that f has precisely $3-1=2$ real roots.

Next we check how many roots are positive. Take $a=0$ and leave $b=\infty$, then

$$\operatorname{var}(\mathscr{S}(0))-\operatorname{var}(\mathscr{S}(\infty)) = \operatorname{var}(2,0,-2,8,-25)-\operatorname{var}(+\infty,+\infty,+\infty,-\infty,-25) = 3-1 = 2.$$

As an immediate consequence of Sturm's theorem, we may devise an algorithm that checks whether all the roots of a given polynomial are real. Recall that this was a necessary condition for example in Laguerre–Samuelson theorem.

Algorithm 6.5.4. *Given a polynomial $f \in R[x]$, $R \subseteq \mathbb{R}$, this algorithm returns true if all the roots of f are real and false otherwise.*

1. *Using Observation 5.3.4, compute the radical of f*

$$\operatorname{rad} f = \frac{f}{\gcd(f, f')};$$

2. *using Algorithm 6.5.2, compute the Sturm sequence $\mathscr{S}$ of* $\operatorname{rad} f$*;*
3. *evaluate $\mathscr{S}$ at $\pm\infty$ and count the number of sign changes;*
4. *if* $\operatorname{var}\mathscr{S}(-\infty) - \operatorname{var}\mathscr{S}(\infty) = \deg \operatorname{rad} f$*, then return true, otherwise return false.*

The proof of correctness of this algorithm is straightforward. Sturm's theorem gives raise also to a method of isolating real roots of a polynomial. First let us introduce the notion of root isolating sequences.

Definition 6.5.5. Let $f \in R[x]$ be a polynomial with coefficients in a subring of $\mathbb{R}$. A sequence $(a_0, b_0), \ldots, (a_m, b_m)$ of pairs of real numbers is said to be *separating roots* of f, if for every i:

- $b_{i-1} \le a_i \le b_i \le a_{i+1}$;
- $a_i = b_i$ if and only if a_i is a root of f;
- if $a_i < b_i$, then the interval (a_i, b_i) contains exactly one root of f;
- every root of f between a_0 and b_m is either equal to some a_i or is contained in an interval (a_i, b_i) for a unique i.

We say that the sequence is *tightly separating roots* of f if it is separating the roots of f and additionally

- for every $a_i < b_i$ neither the first nor the second derivative of f vanishes inside (a_i, b_i).

Algorithm 6.5.6. *Given a polynomial f and two constants $a, b \in \mathbb{R} \cup \{\pm\infty\}$, $a < b$, this algorithm returns a sequence tightly separating the roots of f.*

1. *If either of the endpoints is infinite (i.e. $a = -\infty$ or $b = \infty$), then replace it by respectively a lower or upper bound on the roots of f, using results from Section 6.3;*
2. *if a is a root of f, then:*
   a) *find such $\varepsilon > 0$ that f has no roots in $(a, a+\varepsilon)$;*
   b) *evaluating this algorithm recursively, compute a tightly root-separating sequence $\mathscr{U}$ of f on $(a+\varepsilon, b)$;*
   c) *return $\big((a,a)\big) \oplus \mathscr{U}$ and quit;*
3. *if b is a root of f, then:*
   a) *find such $\varepsilon > 0$ that f has no roots in $(b-\varepsilon, b)$;*
   b) *evaluating this algorithm recursively, compute a tightly root-separating sequence $\mathscr{L}$ of f on $(a, b-\varepsilon)$;*
   c) *return $\mathscr{L} \oplus \big((b,b)\big)$ and quit;*
4. *compute the number N of roots of f in (a,b);*
5. *if $N = 0$ then return an empty sequence and quit;*
6. *if $N = 1$, then compute the numbers N_1, N_2 of roots of respectively f' and f'' in (a,b);*
7. *if both N_1 and N_2 are zero, then return $\big((a,b)\big)$ and quit;*
8. *bisect the interval (a,b) taking $c := \frac{a+b}{2}$;*
9. *if c is a root of f, then*
   a) *find such $\varepsilon > 0$ that c is the only root of f in $(c-\varepsilon, c+\varepsilon)$;*
   b) *evaluating this algorithm recursively, compute a tightly root-separating sequence $\mathscr{L}$ of f on $(a, c-\varepsilon)$;*
   c) *evaluating this algorithm recursively, compute a tightly root-separating sequence $\mathscr{U}$ of f on $(c+\varepsilon, b)$;*
   d) *return $\mathscr{L} \oplus \big((c,c)\big) \oplus \mathscr{U}$ and quit;*
10. *otherwise, when c is not a root of f:*
    a) *evaluating this algorithm recursively, compute a tightly root-separating sequence $\mathscr{L}$ of f on (a,c);*
    b) *evaluating this algorithm recursively, compute a tightly root-separating sequence $\mathscr{U}$ of f on (c,b);*
    c) *return $\mathscr{L} \oplus \mathscr{U}$.*

$\oplus$ = concatenation, explain

Remark 51. In order to find in step 2a (respectively in 3a and 9a) $\varepsilon > 0$ such that f has no roots in $(a, a+\varepsilon)$ (respectively in $(b-\varepsilon, b)$ and $(c-\varepsilon, c) \cup (c, c+\varepsilon)$), one can either use Mahler's or Rump's theorems (c.f. Theorem 6.4.3 and 6.4.5) or apply Corollary 6.3.4 to $f(t+a)$. When using above-mentioned theorems we may compute ε only once and store it along with f. This is particularly convenient when using lazy initialization and object-oriented programming paradigm.

Remark 52. Construction of a Sturm sequence is quite costly due to a number of gcd being evaluated. Thus, the Sturm sequence, once computed, should be stored and reused, in order to avoid recomputing it repeatedly during the recursion. Again, this can be most easily achieved using lazy initialization in an object-oriented programming language.

Hankel matrix and Hermite form,
complex roots counting/isolating: Schur-Cohn algorithm; Saux Picart 1998 - Schur-Cohn subtransform (?); Routh–Hurwitz theorem
David C. Kurtz, A Sufficient Condition for All the Roots of a Polynomial To Be Real

Hermite's method

We now discuss another method for counting real roots of a polynomial. It is based on a theorem due to Ch. Hermite[14] (see Theorem 6.5.10 below). Again let $f = f_0 + f_1 x + \cdots + f_n x^n$ be a polynomial with real coefficients and $\xi_1, \ldots, \xi_n \in \mathbb{C}$ be all its roots, with multiple roots being repeated. Recall (see Section 3.8) that there are two matrices that we may naturally associate to f. These are: the companion matrix C_f and the Vandermonde matrix $V_{\xi_1,\ldots,\xi_n}$:

$$C_f = \begin{pmatrix} 0 & 0 & \cdots & 0 & -f_0/f_n \\ 1 & 0 & \cdots & 0 & -f_1/f_n \\ 0 & 1 & \cdots & 0 & -f_2/f_n \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & 1 & -f_{n-1}/f_n \end{pmatrix} \quad \text{and} \quad V_{a_1,\ldots,a_n} = \begin{pmatrix} 1 & \xi_1 & \xi_1^2 & \cdots & \xi_1^{n-1} \\ 1 & \xi_2 & \xi_2^2 & \cdots & \xi_2^{n-1} \\ 1 & \xi_3 & \xi_3^2 & \cdots & \xi_3^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \xi_n & \xi_n^2 & \cdots & \xi_n^{n-1} \end{pmatrix}$$

In addition we shall denote $H_f := V^T_{\xi_1,\ldots,\xi_n} \cdot V_{\xi_1,\ldots,\xi_n}$. Thus the entry h_{ij} at the i-th row and j-th column of H_f equals

$$h_{ij} = \sum_{k=1}^{n} \xi_k^{i-1} \cdot \xi_k^{j-1} = s_{i+j-2}.$$

This way we obtain:

Observation 6.5.7. *The matrix H_f defined above has the form*

$$H_f = \begin{pmatrix} s_0 & \cdots & s_{n-2} & s_{n-1} \\ \vdots & \iddots & \iddots & s_n \\ s_{n-2} & \iddots & \iddots & \vdots \\ s_{n-1} & s_n & \cdots & s_{2n-2} \end{pmatrix}, \qquad \text{where} \quad s_k = \sum_{j-1}^{n} \xi_j^k.$$

In particular, H_f is a Henkel matrix.

[14] Charles Hermite (born 24 December 1822, died 14 January 1901) was a French mathematician.

Later we will present an explicit way for computing the entries of H_f. For now it suffices to show that they are all real.

Lemma 6.5.8. *We have $s_0, s_1, \dots, s_{2n-2} \in \mathbb{R}$.*

Proof. The coefficients of f are real, by assumption. Hence, the non-real complex roots of f come in conjugate pairs by Proposition 6.1.5. Let $\xi_1, \dots, \xi_r \in \mathbb{R}$ be all the real roots of f and $\zeta_1, \overline{\zeta}_1, \dots, \zeta_c, \overline{\zeta}_c \in \mathbb{C} \setminus \mathbb{R}$ be the non-real ones. Then we have

$$s_k = \sum_{i=1}^{r} \xi_i^k + \sum_{j=1}^{c} \left(\zeta_j^k + \overline{\zeta}_j^k\right) = \sum_{i=1}^{r} \xi_i^k + 2 \cdot \sum_{j=1}^{c} \operatorname{re}(\zeta_j^k) \in \mathbb{R}. \qquad \square$$

We know (see Proposition 3.8.5) that the rank of $V_{\xi_1,\dots,\xi_n}$ is the number of distict elements among $\xi_1, \dots, \xi_n$, hence the number of distinct roots of f. The same holds for H_f.

Proposition 6.5.9. *The rank of H_f equals the number of distinct roots of f.*

Proof. Let $\xi_{i_1}, \dots, \xi_{i_k}$ be all the distinct roots of f. Combining Corollary 3.8.4 with Theorem 3.4.1 we have

$$\det\left(V^T_{\xi_{i_1},\dots,\xi_{i_k}} \cdot V_{\xi_{i_1},\dots,\xi_{i_k}}\right) = \det\left(V_{\xi_{i_1},\dots,\xi_{i_k}}\right)^2 \neq 0.$$

Consequently $\operatorname{rank} H_f \geq k$. On the other hand, if we append any of the omitted ξ_j's, the resulting determinant will vanish. Thus $\operatorname{rank} H_f < k+1$. $\square$

We will show that the matrix H_f encodes the number of distinct *real* roots of f, too. Recall that the eigenvalues of a real symmetric matrix are all real (see Proposition 3.6.8). The number of positive eigenvalues minus the number of the negative ones is called the *signature* of the matrix.

Theorem 6.5.10 (Hermite)**.** *The signature of H_f equals the number of distinct real roots of f.*

Proof

Although the previous theorem provides a definite answer for the number of the real roots of f, to turn this result into a practical tool we need a method for computing the entries of H_f. Keep the notation of Observation 6.5.7, so that $s_k = \xi_1^k + \cdots + \xi_n^k$, where $k \in \{0, \dots, 2n-2\}$ and $\xi_1, \dots, \xi_n$ are all the roots of f. Of course, $s_0 = n$ and $s_1 = -f_{n-1}/f_n$ by Viète's formulas. To find a general formula we use the second matrix recalled in the introduction to this section, namely the companion matrix of f. Recall that the trace of a matrix is the sum of its diagonal entries.

Proposition 6.5.11. *If C_f is the companion matrix of f, then s_k is the trace of C_f^k for every $k \in \{0, \dots, 2n-2\}$.*

Proof. Observation 3.8.7 asserts that the roots $\xi_1, \dots, \xi_n$ of f are the eigenvalues of C_f. Next, Proposition 3.6.9 implies that $\xi_1^k, \dots, \xi_n^k$ are the eigenvalues of C_f^k. Thus the assertion follows directly from Proposition 3.6.7. $\square$

Descartes' rule of signs

Below we present another method for isolating roots of polynomials. Take a nonzero polynomial $f = f_0 + f_1 x + \cdots + f_n x^n$ with real coefficients. Observe that if all the coefficients of f are non-negative (and at least one is strictly positive, since $f \neq 0$), then f cannot have any positive roots. Indeed, $f(\xi)$ is strictly positive for any $\xi \in (0, \infty)$ as a nonzero sum of non-negative terms. The same holds true when all the coefficients of f are non-positive, then again f cannot have positive roots. This suggests that signs—more precisely: sign varaitions—of coefficients may determine the number of positive roots of a polynomial. In fact it does.

Theorem 6.5.12 (Descartes' rule of signs)**.** *Let $f = f_0 + f_1 x + \cdots + f_n x^n$ be a polynomial with real coefficients. Assume that f has exactly p positive roots, when counted with multiplicities. Denote by $v := \operatorname{var}(f_0, \ldots, f_n)$ the number of sign changes in the list of coefficients of f. Then $v \geq p$ and the difference $v - p$ is even.*

The proof of the theorem will be preceded by few preparatory facts. Below we keep the notation of Theorem 6.5.12.

Lemma 6.5.13. *Denote the numbers of sign changes in the sequences of coefficients of f and its derivative f' by $v := \operatorname{var}(f)$ and $v' := \operatorname{var}(f')$, respectively. Then*

$$v - 1 \leq v' \leq v.$$

Proof. If $f = f_0 + f_1 x + \cdots + f_n x^n$, then $f' = f_1 + 2f_2 x + \cdots + n f_n x^{n-1}$. In particular, the coefficients of f (except the constant term) are multiplied by positive integers. Therefore, there are only two possibilities. Either $\operatorname{sgn} f_0 = \operatorname{sgn} f_1$ and then $v = v'$, or $\operatorname{sgn} f_0 \neq \operatorname{sgn} f_1$, then passing from f to f' we loose one sign change and consequently $v' = v - 1$. □

Notice that we have the following consequence of Rolle's mean-value theorem:

Observation 6.5.14. *Let $\xi_i < \xi_j$ be two consecutive real roots of f. Then there is a root ζ of f' such that $\xi_i < \zeta < \xi_j$.*

Now, let $\xi \in \mathbb{R}$ be a root of f of multiplicity $m_f(\xi) = k > 1$. Recall (see Lemma 5.3.1) that ξ is then a root of f' of multiplicity $m_{f'}(\xi) = k - 1$. Combining this fact with the preceding observation we obtain:

Corollary 6.5.15. *If $\xi_1 \leq \xi_2 \leq \cdots \leq \xi_n$ are all the real roots of f, then there are roots $\zeta_1 \leq \zeta_2 \leq \cdots \leq \zeta_{n-1}$ of f' such that*

$$\xi_1 \leq \zeta_1 \leq \xi_2 \leq \cdots \leq \xi_i \leq \zeta_i \leq \xi_{+1} \leq \cdots \leq \xi_n.$$

We shall briefly say that $n - 1$ roots of the derivative lie between n roots of f, whereas for a root ξ_i of multiplicity k this is to be understood as

$$\zeta_{i-1} < \xi_i = \zeta_i = \xi_{i+1} = \cdots = \zeta_{i+k-2} = \xi_{i+k-1} < \zeta_{i+k}.$$

Proof. When we traverse the list of coefficients of f, starting at the constant term f_0 and ending at the leading term f_n, we will encounter an even number of sign changes if and only is the sings of f_0 and f_n agree. Otherwise, the number of sign changes must be odd. Now, $f_0 = f(0)$ and the sign of f_n is the same as that of $\lim_{x\to\infty} f(x)$. It follows that v is odd if and only if these two signs differ. On the other hand, a root ξ of f has an odd multiplicity if and only if f changes sign at ξ. Now, p is a sum of the multiplicities of all positive roots. Thus, the parity of p depends only on the number of roots with odd multiplicities. If this number itself is odd, then p is also odd and the signs of $f(0)$ and $\lim_{x\to\infty} f(x)$ differ. Otherwise, p is even and the signs agree. This shows that c and p have the same parity. In other words, $v - p \in 2\mathbb{Z}$.

We still need to show that $v - p$ is non-negative. Use an induction on the degree of f. The case $\deg f = 0$ is trivial, since f neither has any roots nor its list of coefficients admits any sign changes. Take now a polynomial f of degree $\deg f = n$ and assume that the thesis holds for all polynomials of degree $n-1$. Take the derivative f' of f. Denote the number of sign changes in the list of coefficients of f' by v' and the number of positive roots of f' by p'. The derivative of f has degree $n-1$, hence by the inductive hypothesis $v' \geq p'$. Corollary 6.5.15 asserts that f' has at least $p-1$ real roots that lie between the p positive roots o f. Thus $p-1 \leq p' \leq v'$ and $v' \leq v$ by Lemma 6.5.13. It follows that $v \geq p-1$, but we have already proved that $v-p$ is an even number, hence v cannot be equal to $p-1$. It follows that $v \geq p$, as desired. □

Proposition 6.5.16. *Keep the notation of the previous theorem. If all the roots of f are real, then $v = p$.*

Proof. By Descartes' rule of signs $v \geq p$. We need to show that $v \leq p$ if f has only real rots. We again argue by induction on the degree of f. The assertion is vacuously true for $\deg f = 0$. Take thus a polynomial f of degree $\deg f = n$ and assume that the thesis holds for all polynomials of lower degree. Let $\xi_1 \leq \xi_2 \leq \cdots \leq \xi_n$ be all the roots of f. Now, Corollary 6.5.15 says that between very two consecutive roots of f there is a root of its derivative. Thus, f' has at least $n-1$ real roots. The degree of f' is $n-1$, hence these are all its roots. Therefore, the inductive hypothesis applied to f' states that $v' = p'$, where, as before, $v' = \mathrm{var}(f')$ and p' is the number of positive roots of f'.

Observe that the roots of f' that lie between the positive roots of f mus be positive, too. Likewise, the roots of f' lying between the negative roots of f are negative. Consequently

$$p - 1 \leq p' = v' \leq p.$$

Now, Lemma 6.5.13 asserts that v' equals either v or $v-1$. It follows that $p \in \{v-1, v\}$ and Descartes' rule of sign rules out the case $p = v-1$. □

Before we discuss root isolation through Descartes' rule of signs, we first present its another application that at first may be quite surprising to the reader. Recall (see Definition 3.4.10) of a matrix M is $\chi_M = \det(xI_n - M)$, where I_n is the unit matrix of the size of M. If all eigenvalues of M are real, then the difference between the numbers of positive and negative ones is called the signature. We have the following effective method for computing the signature of a matrix.

Corollary 6.5.17. *Let M be a square matrix and $\chi_M = f_0 + f_1 x + \cdots + f_n x^n$, with $f_n = 1$, be its characteristic polynomial. Assume that all eigenvalues of M are real. Then the signature of M equals*

$$\operatorname{sgn} M = \operatorname{var}\big(f_0, f_1, \ldots, f_n\big) - \operatorname{var}\big((-1)^0 \cdot f_0, (-1)^1 \cdot f_1, \ldots, (-1)^n \cdot f_n\big).$$

Proof. Roots of χ_M are the eigenvalues of M. By assumption they are all real, hence Proposition 6.5.16 asserts that the number of positive eigenvalues of M (positive roots of χ_M) equals $\operatorname{var}(f_0, \ldots, f_n)$. On the other hand, the number of negative eigenvalues of M is the number of positive roots of $\chi_M(-x)$ and by the same proposition equals $\operatorname{var}\big((-1)^0 \cdot f_0, (-1)^1 \cdot f_1, \ldots, (-1)^n \cdot f_n\big)$. □

Example 77. Let us compute the signature of a matrix

$$M = \begin{pmatrix} 2 & 6 & 4 & 1 \\ 6 & 8 & 2 & -4 \\ 4 & 2 & 6 & -7 \\ 1 & -4 & -7 & 4 \end{pmatrix}.$$

The matrix is symmetric, hence all its eigenvalues are real by Proposition 3.6.8. The characteristic polynomial of M is

$$\chi_M = \det(x \cdot I_4 - M) = x^4 - 20x^3 + 18x^2 + 704x - 1264.$$

We have:

$$\operatorname{var}(-1264, 704, 18, -20, 1) = 3,$$
$$\operatorname{var}(-1264, -704, 18, 20, 1) = 1.$$

Consequently $\operatorname{sgn} M = 2$.

Let us now proceed to the root isolation problem. An immediate consequence of Descartes' rule of signs is the following fact. The first thesis is just the restatement of the observation we made on the beginning of this subsection.

Corollary 6.5.18. *Let $f = f_0 + f_1 x + \cdots + f_n x^n$ be a polynomial with real coefficients,*

- *if $\operatorname{var}(f_0, \ldots, f_n) = 0$, then f has no positive roots;*
- *if $\operatorname{var}(f_0, \ldots, f_n) = 1$, then f has precisely one positive root.*

Nowadays, there exist a number of root-isolating techniques that employ the above corollary from Descartes' rule of signs. The basic idea of all of them is to perform a substitution, that maps an interval suspected of containing a root onto a half-line $(0, \infty)$. If after the substitution, the previous corollary delivers a definitive answer, then we know that the original interval contains either none or just one root. Otherwise, we subdivide the interval and repeat the process. The negative roots are handled by taking $f(-x)$ in place of f. And verification if zero is a root is trivial as it boils down to looking at the constant term of f.

This idea goes back to 19th century papers by A. Vincent[15]. The various existing algorithms differ in subdivision strategies.

The above-mentioned transformation is the following *homographic map* (also known as *Möbius*[16] *map*):

$$x \mapsto \frac{\alpha+\beta x}{\gamma+\delta x}, \qquad \text{where} \quad \det\begin{pmatrix}\alpha & \beta\\ \gamma & \delta\end{pmatrix} \neq 0.$$

It is clear that this function maps 0 to α/γ and ∞ to β/δ. Given a polynomial $f \in \mathbb{R}[x]$ the transformed polynomial (observe that a direct substitution leads to a rational function not a polynomial) is obtained as

$$g := \big(\gamma+\delta x\big)^{\deg f} \cdot f\Big(\frac{\alpha+\beta x}{\gamma+\delta x}\Big).$$

Probably the simplest of root-separation algorithms based on Descartes' rule of signs is the one due to A. Alesina and M. Galuzzi. In this method the denominator in the above map takes a form $1+x$ (i.e. $\gamma=\delta=1$) and analyzed intervals are just bisected.

Algorithm 6.5.19 (Alesina–Galuzzi, 2000)**.** *Given a nonzero square-free polynomial $f \in \mathbb{R}[x]$ and an open interval $(a,b) \subseteq (0,\infty)$, this algorithm constructs a sequence of intervals separating roots of f contained in (a,b).*

1. *If $b=\infty$, then replace it by an upper bound on real roots of f;*
2. *if $a \geq b$, then return an empty sequence and quit;*
3. *set*
$$g := \big(1+x\big)^{\deg f} \cdot f\Big(\frac{a+bx}{1+x}\Big)$$
and let $g_0,\ldots,g_n$ be the coefficients of g;
4. *compute the number $v := \operatorname{var}(g_0,\ldots,g_n)$ of sign changes in the list of coefficients of g;*
5. *if $v=0$, then g has no positive roots, hence f has no roots in (a,b), in such a case return and empty sequence and quit;*
6. *if $v=1$, then g has precisely one positive root and so f has a unique root in (a,b), thus return (a,b) and quit;*
7. *if $v \geq 2$, then set $c := {}^1/_2\cdot(a+b)$ and execute this algorithm recursively for $\big(f,(a,c)\big)$ and $\big(f,(c,b)\big)$, denote the returned sequences respectively by $\mathscr{L}$ and $\mathscr{U}$;*
8. *if $f(c)=0$, then return $\mathscr{L} \oplus \big((c,c)\big) \oplus \mathscr{U}$, otherwise return $\mathscr{L} \oplus \mathscr{U}$.*

[15] Alexandre Joseph Hidulphe Vincent (born 20 November 1797, died 26 November 1868) was a French mathematician.

[16] August Ferdinand Möbius (born 17 November 1790, died 26 September 1868) was a German mathematician and astronomer.

We need to show the correctness of this algorithm. Since a nonzero polynomial has only finite number of roots, it is obvious that recursively halving intervals, we will eventually arrive at intervals so small that they contain no more than one root. Thus, the only aspect that needs to be proved is that this algorithm halts. In other words, that for short enough intervals the condition $v \leq 1$ (see steps 5 and 6) is guaranteed. This is ensured by the following theorem.

Theorem 6.5.20 (Vincent–Alesina–Galuzzi)**.** *Let $f \in \mathbb{R}[x]$ be a real polynomial. Then there is $\delta > 0$ such that for every $a, b \in \mathbb{R}$, if $|a-b| < \delta$, then the polynomial*

$$g := (1+x)^{\deg f} \cdot f\Big(\frac{a+bx}{1+x}\Big)$$

admits at most one sign change in its list of coefficients.

Proof

Remark 53. Theorem 6.5.20 was presented and proved by A. Alesina and M. Galuzzi in [4] but accredited by the authors to A. Vincent. To do justice to all parties involved, here we decided to associate all three names to the theorem.

Example 78. Take a polynomial $f = x^5 - 8x^3 + 4x - 1$. This is the depressed form of the polynomial we considered in Examples 68 through 73. Algorithm 6.3.13 returns an upper bound for positive roots of f equal $M = 4$. The intervals examined by Algorithm 6.5.19 can be visualized in form of a tree presented in Figure 6.6. In order to separate **all** real roots of f, we need to take care also of the negative ones. To this end, we apply Algorithm 6.5.19 to $f(-x) = -x^5 + 8x^3 - 4x - 1$ (see Figure 6.7) and subsequently change signs of returned intervals. Combining these two lists we obtain a roots separating sequence:

$$\Big((-4,-2),(-2,0),\Big(0,\frac{1}{2}\Big),\Big(\frac{1}{2},1\Big),(2,4)\Big)$$

Figure 6.6: Tree of intervals examined by Algorithm 6.5.19 for $f = x^5 - 8x^3 + 4x - 1$.

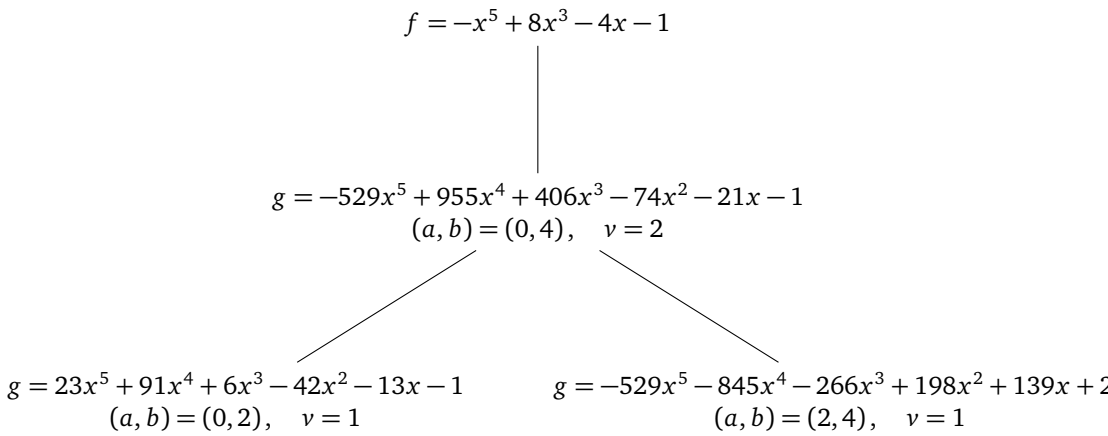


Figure 6.7: Tree of intervals examined by Algorithm 6.5.19 for $f = -x^5 + 8x^3 - 4x - 1$.

Exercise 12. Improve Algorithm 6.5.19 in such a way that it will return a tightly separating sequence.

Another algorithm for separating real roots is due to A. Akritas and A. Strzeboński. It is based on a 19th century method of A. Vincent and is often referred to as *Vincent–Akritas-Strzeboński continued-fraction method*. This algorithm utilizes a series of homographic maps τ_i of a general form:

$$\tau_i(x) := a_i + \frac{1}{x} = \frac{1 + a_i x}{x},$$

with carefully selected coefficients $a_i \geq 0$. Denote by $T_k := \tau_k \circ \tau_{k-1} \circ \cdots \circ \tau_1$.

Lemma 6.5.21. *For every $k \geq 1$, the function T_k is again a homographic map.*

Proof. This lemma is a special case of a far more general phenomenon and is best viewed on grounds of projective geometry. However, not to drive a reader too far afield, we give here an elementary, down-to-earth proof. We argue by induction on k. The assertion is clear for $k = 1$. Take, thus, $k \geq 1$ and assume that there are such $\alpha, \beta, \gamma, \delta \in \mathbb{R}$ that

$$T_{k-1} = \frac{\alpha + \beta x}{\gamma + \delta x}.$$

It follows that

$$T_k = \tau_k \circ T_{k-1} = a_k + \frac{\gamma + \delta x}{\alpha + \beta x} = \frac{(a_k\alpha + \gamma) + (a_k\beta + \delta) \cdot x}{\alpha + \beta x}$$ □

A key to Vincent–Akritas–Strzeboński continued-fractions method is the following theorem, which also explains where the "continued-fractions" come from:

Theorem 6.5.22 (A. Vincent). *Let $f \in \mathbb{R}[x]$ be a square-free polynomial and $(a_i)_{i\in\mathbb{N}}$ a sequence of integers such that $a_1 \geq 0$ and $a_i \geq 1$ for $i > 1$. Then for a sufficiently large n, the sequence of coefficients of a polynomial*

$$g_n = (\gamma_n + \delta_n x)^{\deg f} \cdot f\Big(\frac{\alpha_n + \beta_n x}{\delta_n + \gamma_n x}\Big)$$

admits at most one change of signs, where $\alpha_n, \beta_n, \delta_n, \gamma_n$ are coefficients of a map T_n defined above (i.e. $T_n = \frac{\alpha_n+\beta_n x}{\gamma_n+\delta_n x}$).

Moreover, if for every n large enough, g_n admits precisely one sign change, then a continued fraction of a form

$$\xi = a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4 + \cdots}}}$$

converges to a positive roots of f.

Proof

We are now ready to present a continued fraction version of Vincent's method.

Algorithm 6.5.23 (Vincent–Akritas–Strzeboński). *Given a nonzero square-free polynomial $f = f_0 + f_1 x + \cdots + f_n x^n$ with real coefficients and a homographic map $\Phi(x) = \frac{\alpha+\beta x}{\gamma+\delta x}$, this algorithm constructs a sequence of intervals separating roots of f.*

missing

1. *Let $v := \operatorname{var}(f_0, \ldots, f_n)$ be the number of sign changes in the list of coefficients of f;*
2. *if $v = 0$, then return an empty sequence and quit;*
3. *if $v = 1$, then:*
   a) *set $a := \min\{{}^{\alpha}/_{\gamma}, {}^{\beta}/_{\delta}\} (= \min\{\Phi(0), \Phi(\infty)\})$ and $b := \max\{{}^{\alpha}/_{\gamma}, {}^{\beta}/_{\delta}\} (= \max\{\Phi(0), \Phi(\infty)\})$;*
   b) *if $b = \infty$, then compute an upper bound M for positive roots of f, using methods described in Section 6.3, and replace b by $\Phi(M)$;*
   c) *return a singleton $\big((a, b)\big)$ and quit;*
4. *compute a lower bound m for positive roots of f and set $\mu := \lfloor m \rfloor$;*
5. *if μ is bigger then a fixed threshold μ_0 (see Remark 54), then substitute $f(\mu \cdot x)$ for f and $\frac{\alpha+\beta\mu x}{\gamma+\delta\mu x}$ for Φ and skip the next step;*
6. *if $\mu \geq 1$ but less or equal μ_0, then:*
   a) *replace f by $f(x + \mu)$ and compose Φ with a translation by μ, setting*

$$\Phi(x) := \frac{(\alpha + \beta\mu) + \beta x}{(\gamma + \delta\mu) + \delta x};$$

   b) *if after the substitution $f(0) = 0$, then execute this algorithm recursively for ${}^{f}/_{x}$ and Φ, denote the result by $\mathscr{S}$, return $\big((\Phi(0), \Phi(0))\big) \oplus \mathscr{S}$ and quit;*

7. *separate roots of f contained in the unit interval $(0,1)$ as follows:*

   a) *map $(0,1)$ onto the positive half line, setting:*

$$g := (1+x)^{\deg f} f\Big(\frac{1}{1+x}\Big) \qquad \text{and} \qquad \Psi(x) := \Phi\Big(\frac{1}{1+x}\Big) = \frac{(\alpha+\beta)+\alpha x}{(\gamma+\delta)+\gamma x};$$

   b) *execute this algorithm recursively for g, Ψ and denote the result by $\mathscr{L}$;*

8. *separate roots of f contained in the half line $(1,\infty)$ as follows:*

   a) *map $(1,\infty)$ onto the whole positive half line $(0,\infty)$, setting:*

$$h := f(1+x) \qquad \text{and} \qquad \Psi(x) := \Phi(1+x) = \frac{(\alpha+\beta)+\beta x}{(\gamma+\delta)+\delta x};$$

   b) *execute this algorithm recursively for h, Ψ and denote the result by $\mathscr{U}$;*

9. *if 1 is a root of f (i.e. $f(1)=0$), then return $\mathscr{L} \oplus \big((\Phi(1),\Phi(1))\big) \oplus \mathscr{U}$, otherwise return $\mathscr{L} \oplus \mathscr{U}$.*

Remark 54. The threshold μ_0 at step 5 needs to be experimentally computed. In [3] it is suggested that $\mu_0 = 16$. However, to the best of the knowledge of the author, there is no rationale behind this particular value.

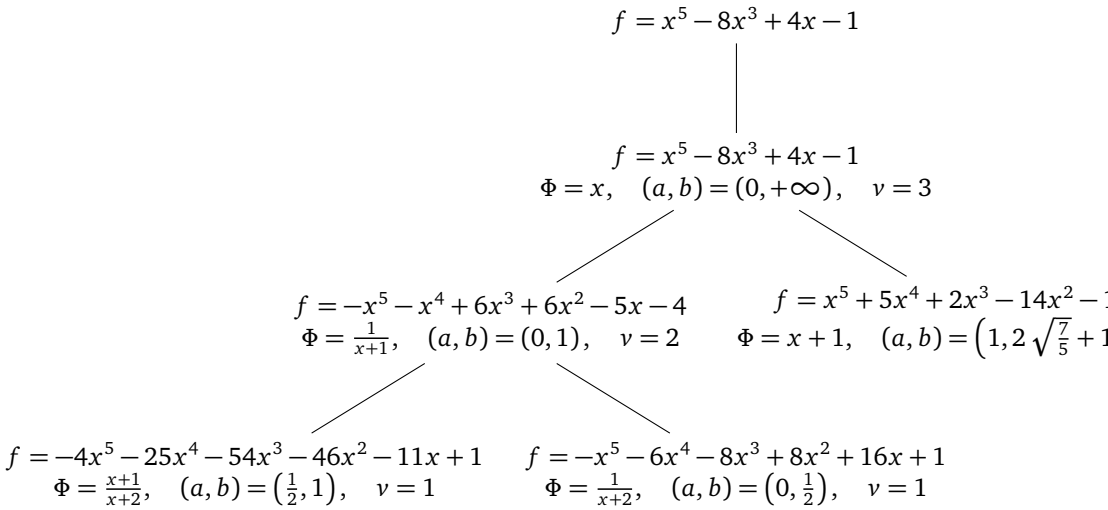
Remark 55. If the polynomial f has integer coefficients, then all the intermediate polynomials constructed by Algorithm 6.5.23 have integer coefficients, as well. This is one of big assets of this algorithm.

Remark 56. The substitutions of a form $f \mapsto f(\mu+x)$ can be implemented using Algorithm 1.1.34, while the substitution $f \mapsto (1+x)^{\deg f} \cdot f(1/(1+x))$ in step 7a is obtained as a composition of a substitution $f \mapsto f(1+x)$ followed by taking a reciprocal of the resulting polynomial (see Observation 1.1.2). The reader should be however aware of Knuth's warning that "premature optimization is the root of all evil". It is quite possible that the library for polynomial computations, that one uses, already takes care of such low-level optimizations.

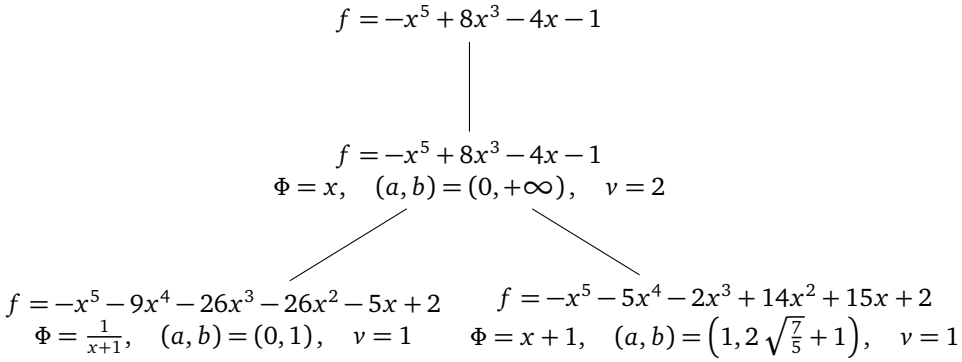
Example 79. Consider the same polynomial that we used to illustrate Algorithm 6.5.19:

$$f = x^5 - 8x^3 + 4x - 1.$$

The initial homographic map is the identity and the following tree gathers intervals visited by Algorithm 6.5.23:



As with Algorithm 6.5.19, the negative roots are handled replacing f by $f(-x)$. For $f(-x)$ Algorithm 6.5.23 analyzes the following intervals:



All in all we have obtained the following list of intervals each containing a single roots of f:

$$\left(\left(-2\sqrt{\frac{7}{5}} - 1, -1\right), (-1, 0), \left(0, \frac{1}{2}\right), \left(\frac{1}{2}, 1\right), \left(1, 2\sqrt{\frac{7}{5}} + 1\right)\right).$$

Exercise 13. Improve Algorithm 6.5.23 to make it tightly separate roots of f.

Exercise 14. Algorithm 6.5.23 is presented in a recursive form. Write an iterative version.

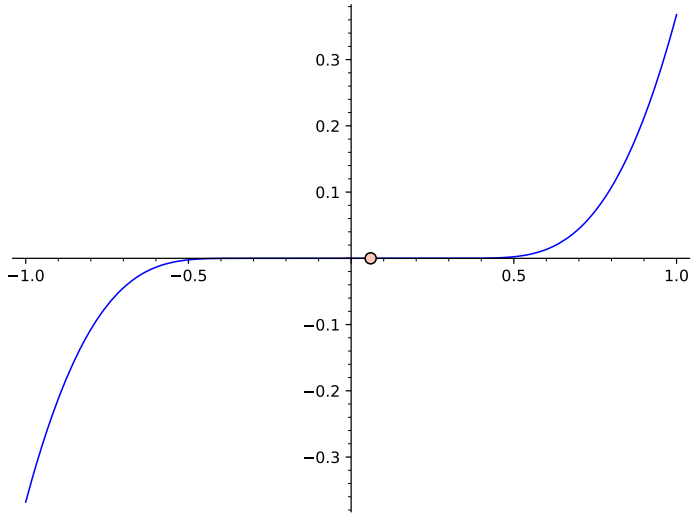


Figure 6.8: Graph of $f = x^3 \cdot \exp(-x^{-2})$.

Speed comparison of various root-separating algs
multivariate isolation: Root isolation of zero-dimensional polynomial systems with linear univariate representation, Jin-San Cheng, Xiao-Shan Gao,Leilei Guo

6.6 Root approximation

Abel–Ruffini theorem (see Theorem 6.2.3) asserts that roots of a general polynomial of high degree cannot be represented exactly. At least not when we are using only n-th roots and the four basic arithmetic operations. Hence in this section, we concentrate on a problem of approximating a single (isolated) root of a polynomial (or more generally a smooth function) up to a desired accuracy.

What is an accuracy?

First of all, we need to specify, what we mean, when we say that a root is approximated with some accuracy. The point is, that there is no single condition that suits all applications. Let $f : D \to \mathbb{C}$ be a continuous function defined on some domain $D \subseteq \mathbb{C}$ and $\delta > 0$ a fixed positive number. A straightforward requirement for $x \in \mathbb{C}$ to approximate a zero of f is

$$|f(x)| < \delta. \tag{6.17}$$

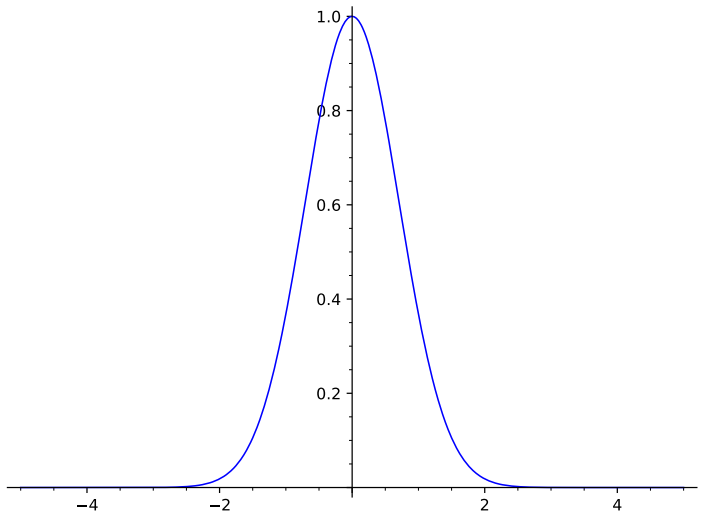


Figure 6.9: Graph of $f = \exp(-x^2)$.

Unfortunately, this test used alone can easily lead to false positives, when graph of f 'looks flat'. Take for example

$$f = x^3 \cdot \exp\left(-x^{-2}\right) \qquad \text{and} \qquad \delta = 10^{-100}.$$

The function f has a single zero at the origin (the graph of f is presented in Figure 6.8), but already $x = 0.06$ passes the test (6.17). This means that, f evaluates to zero at x up to one hundred decimal digits, while x has only one digit in common with the true root, which is a rather poor approximation. An even more extreme example is the following one. Take

$$f = \exp(-x^2),$$

which has no real zeros at all (see Figure 6.9). Nevertheless, for any $0 < \delta < 1$, if $|x| > \sqrt{-\ln \delta}$, then $|f(x)| < \delta$.

Instead of taking the absolute value of $f(x)$, we may opt to measure a distance from x to a root of f. Let $\xi \in D$ be a zero of f and $\varepsilon > 0$ be a requested accuracy. We can formulate a condition for x to approximate ξ as

$$|x - \xi| < \varepsilon. \tag{6.18}$$

However, this approach still has two pitfalls. First, the formula involves ξ and if we already know the root, there is absolutely no point in approximating it! We may overcome this obstacle assuming that $f : D \to \mathbb{R}$ with $D \subseteq \mathbb{R}$ and f changes sign at ξ (e.g. f is a real polynomial and ξ is a simple root of f). Let $\alpha < \beta$ be two numbers

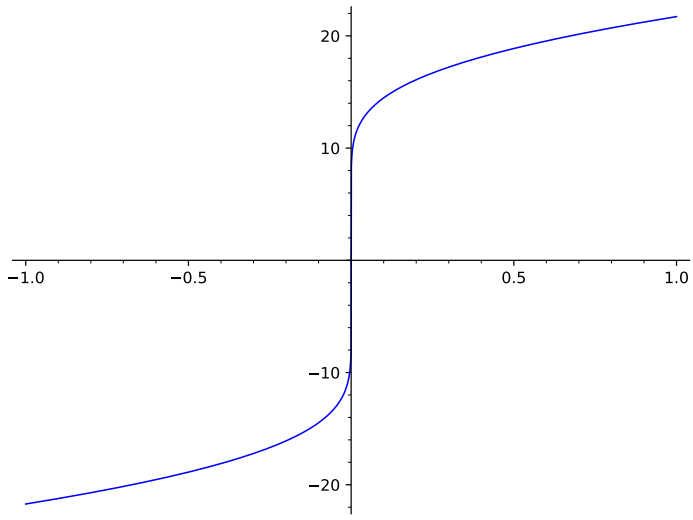


Figure 6.10: Graph of $f = -\frac{\operatorname{sgn}(x)}{\ln\left(\left|\frac{x}{100}\right|^{1/100}\right)}$.

such that $|\alpha - \beta| < \varepsilon$. If $f(\alpha) \cdot f(\beta) < 0$, then f has a root ξ in between (remember that f is continuous by the assumption). In such a case, we say that numbers α and β *bracket* the root ξ. Since clearly $|\alpha - \xi| < \varepsilon$ as well as $|\beta - \xi| < \varepsilon$. Hence, both α and β approximate this root with the desired accuracy. We will encounter this situation in bisection and QIR methods described below. The second problem is dual to the one we have encountered considering condition (6.17). The function f may approach zero very rapidly and even if x is very close to a root, the value of f at x can still be unacceptably high. Take for example

$$f = -\frac{\operatorname{sgn}(x)}{\ln\left(\left|\frac{x}{100}\right|^{1/100}\right)} \qquad \text{and} \qquad x = 10^{-100}.$$

Then f has a single zero at the origin (the graph of f is presented in Figure 6.10) and x may be viewed as rather close to it. Nevertheless, evaluating f at x we have $f(x) \approx 0.4258$.

One way to avoid this problem is to enforce stronger assumptions on f, like for example the Lipschitz condition. Another option is to combine the conditions (6.17) and (6.18) and require x to satisfy both:

$$|f(x)| < \delta \qquad \text{and} \qquad |x - \xi| < \varepsilon$$

or in another form

$$\left\|\left(\omega_1 \cdot f(x), \omega_2 \cdot (x - \xi)\right)\right\| < \varepsilon, \tag{6.19}$$

where $\|\cdot\|$ is a $\mathbb{R}^2$-norm (e.g. $\|(x,y)\| := \max\{|x|,|y|\}$ or $\|(x,y)\| := |x|+|y|$), the parameters ω_1, ω_2 are some fixed 'weights' and $\varepsilon > 0$ is the desired accuracy.

Bisection

Suppose we are given a continuous real function $f : [a,b] \to \mathbb{R}$ such that the signs of $f(a)$ and $f(b)$ differ. Then f must have at least one zero between a and b. Definitely the most straightforward approach to approximate a root is to split the interval at the midpoint and repeat the procedure replacing $[a,b]$ by either its left or right half, namely this one, where f changes sign. Continue this process as long as the length of the interval exceeds the requested accuracy.

Algorithm 6.6.1. *Let $f : [a,b] \to \mathbb{R}$ be a continuous (and computable) function. Assume that $f(a) \cdot f(b) < 0$. Given an accuracy $\varepsilon > 0$, this algorithm return $\zeta \in [a,b]$ such that there is a root ξ of f satisfying $|\zeta - \xi| < \varepsilon$.*

1. *As long as $|a-b| \geq \varepsilon$ repeat the following steps:*
   a) *bisect the interval taking $c := \frac{1}{2} \cdot (a+b)$;*
   b) *if $f(a) \cdot f(c) < 0$, then substitute c for b, otherwise substitute c for a;*
2. *return a.*

Since at each iteration, a and b bracket the root, thus she proof of correctness of this algorithm is trivial. We leave it to the reader. Observe that if f has more roots in $[a,b]$, this algorithm will just pick and approximate one of them.

Exercise 15. The above algorithm uses Ineq. (6.18) to measure accuracy of the root approximation. Alter it to use Ineq. (6.19) instead.

An important factor, when assessing a root-approximation algorithm, is to measure how quickly it converges. Suppose the algorithm in question constructs a sequence $(\zeta_i)_{i\in\mathbb{N}}$, convergent to a root ξ of f. We say that the *rate of convergence* of the sequence (resp. of the algorithm) equals p, if there is a constant $\kappa > 0$ such that

$$|\zeta_i - \xi| \approx \kappa \cdot |\zeta_{i-1} - \xi|^p,$$

for all sufficiently large indices i's. In case of the above algorithm, the sequence $(\zeta_i)_{i\in\mathbb{N}}$ of the approximations of the roots is just the sequence of the midpoints of the successively shrunken intervals. It is clear that $|\zeta_i - \xi| \approx 1/2 \cdot |\zeta_{i-1} - \xi|$, hence:

Observation 6.6.2. *The rate of convergence of Algorithm 6.6.1 equals* 1.

One should not confuse the rate of convergence with the actual time needed to approximate a root with desired accuracy. The latter depends additionally on the cost of a single step of the iteration. Although it is generally true, that algorithms with higher convergence rates tend to be also faster, it is by no means guaranteed.

Regula falsi

The next three algorithms, that we discuss, are very similar to each other. The general idea, in all three cases, is the following one: we assume that locally, on a small enough interval, f behaves more or less like a linear function. We then find the root of this hypothetical linear function and check if it is close enough to a root of f itself. If not, then this means that the interval was not small enough, hence we narrow it down and reiterate.

To be more precise, let $a < b$ and $f : [a, b] \to \mathbb{R}$ be a C^2-smooth function, having a root in (a, b). Assume that neither the first nor the second derivative of f changes sign in $[a, b]$. Thus, f is monotonic and consequently has precisely one root in (a, b). There are four possible cases illustrated in Figure 6.11:

1. $f'(x) \geq 0$, $f''(x) \geq 0$ for $x \in [a, b]$, so f is increasing and convex,
2. $f'(x) \geq 0$, $f''(x) \leq 0$ for $x \in [a, b]$, so f is increasing and concave,
3. $f'(x) \leq 0$, $f''(t) \geq 0$ for $x \in [a, b]$, so f is decreasing and convex,
4. $f'(x) \leq 0$, $f''(t) \leq 0$ for $x \in [a, b]$, so f is decreasing and concave.

Denote

$$z := \begin{cases} b & \text{in cases 1 and 4} \\ a & \text{in cases 2 and 3} \end{cases} \qquad \text{and} \qquad x_0 := \begin{cases} a & \text{in cases 1 and 4} \\ b & \text{in cases 2 and 3.} \end{cases}$$

Define a sequence $(x_i)_{i\geq 0}$ recursively, starting from x_0 above. Let x_i be the root of a linear function interpolating f (i.e. the chord of the graph of f) at z and x_{i-1}. Thus, x_i is the root of $\alpha + \beta x$, where

$$\begin{cases} \alpha + \beta \cdot z = f(z) \\ \alpha + \beta \cdot x_{i-1} = f(x_{i-1}). \end{cases}$$

It follows that

$$x_i = x_{i-1} - \frac{z - x_{i-1}}{f(z) - f(x_{i-1})} \cdot f(x_{i-1}). \tag{6.20}$$

Proposition 6.6.3. *The sequence $(x_i)_{i\geq 0}$ converges to the root of f.*

Proof. We will prove the assertion under the assumptions of case 1. The other cases are analogous. We claim that $(x_i)_{i\geq 0}$ is increasing and bounded from above by the root $\xi \in (a, b)$ of f. This is clear for the initial term x_0. Proceed by induction and assume that $x_{i-1} \leq \xi$. Lagrange mean value theorem asserts that there is $\zeta \in (x_{i-1}, z)$ such that

$$f'(\zeta) = \frac{f(z) - f(x_{i-1})}{z - x_{i-1}}.$$

Substituting the above formula into Eq. (6.20), we get

$$x_i - x_{i-1} = -\frac{1}{f'(\zeta)} \cdot f(x_{i-1}).$$

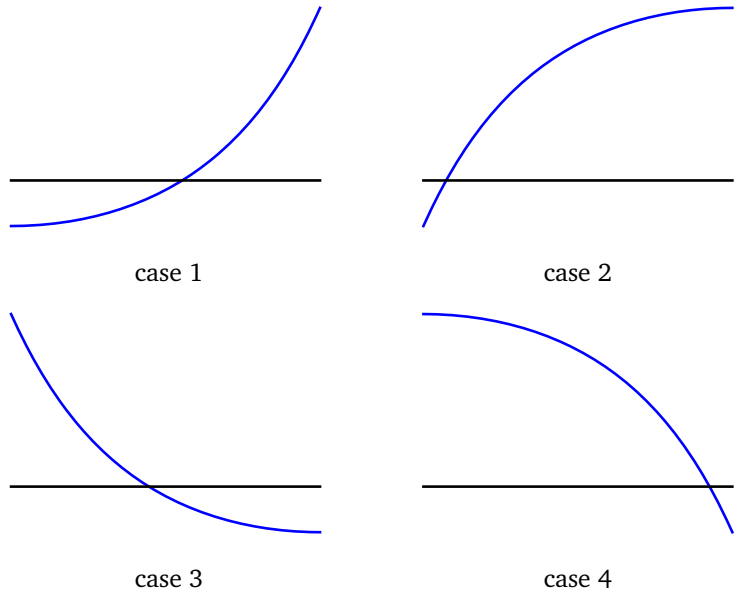


Figure 6.11: Four cases considered in regula falsi method

By the assumptions of case 1, the derivative of f is positive on (a,b), hence $f'(\zeta) > 0$. On the other hand $f(x_{i-1}) \leq 0$ because f is increasing and $x_{i-1} \leq \xi$ by the inductive hypothesis. It follows that $x_i \geq x_{i-1}$. Moreover $f(x_i) \leq 0$ since f is convex. Thus, $x_{i-1} \leq x_i \leq \xi$ as claimed.

The sequence $(x_i)_{i\geq 0}$, being bounded from above and increasing, is convergent. Denote its limit by σ. Computing the limits of both sides of Eq. (6.20) we have

$$\sigma = \sigma - \frac{z-\sigma}{f(z)-f(\sigma)} \cdot f(\sigma).$$

Therefore $(z-\sigma)\cdot f(\sigma) = 0$ and since $z \neq \sigma$, hence $f(\sigma) = 0$. Now, ξ is the only root of f in (a,b). Therefore $\xi = \sigma = \lim_{i\to\infty} x_i$ as desired. □

Figure 6.12 illustrates how the successive terms of the sequence approach the root of f. The above proposition justifies the following algorithm of root approximation.

Algorithm 6.6.4. *Let $a < b$ and $f : [a,b] \to \mathbb{R}$ be a function of class C^2. Assume that $f(a)\cdot f(b) < 0$ and neither f' nor f'' has a root in (a,b). This algorithm approximates the (unique) root of f in (a,b) up to a given accuracy $\varepsilon > 0$.*

1. *If $\operatorname{sgn} f'\big|_{(a,b)} = \operatorname{sgn} f''\big|_{(a,b)}$, then set $z := b$ and $x_0 := a$, otherwise set $z := a$ and $x_0 := b$;*

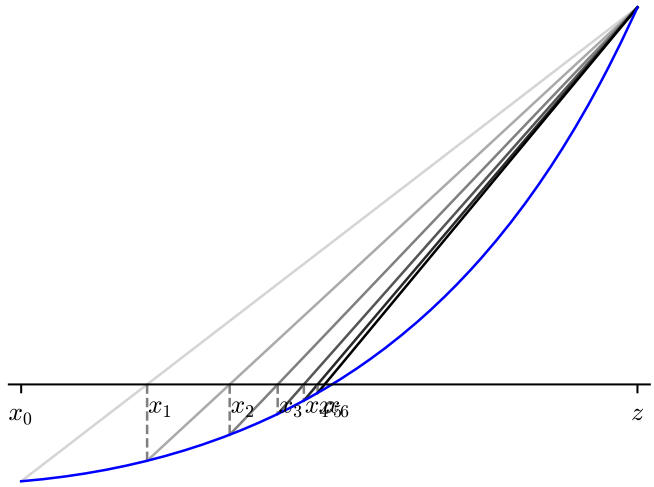


Figure 6.12: Successive approximations in regula falsi

2. *initialize the iteration counter* $i := 0$*;*
3. *repeat the following steps until* x_i *approximates the root of* f *with the requested accuracy:*
   a) *increment* i*;*
   b) *improve the approximation setting:*

$$x_i = x_{i-1} - \frac{z - x_{i-1}}{f(z) - f(x_{i-1})} \cdot f(x_{i-1}).$$

4. *return* x_i*.*

In step 3 of the above algorithm, we are testing whether the current term x_i approximates the root ξ of f with the desired accuracy. While the use of condition Ineq. (6.17) here is clear, the condition Ineq. (6.18) requires some explanation. A straightforward approach is to compute either $f(x_i + \varepsilon)$, when $x_0 = a$ or $f(x_i - \varepsilon)$, when $x_0 = b$. If the sign of it differs from the sign of $f(x_i)$, which is the same as the sign of $f(x_0)$, since the sequence is monotonic, then the root lies at the distance less than ε from x_i. As simple as this approach is, it comes with a cost of evaluating f twice in each iteration: once at x_i in step 3b and the second time at $x_i \pm \varepsilon$. We may avoid it by using the following result (providing that we are able to compute f', e.g. when f is a polynomial).

Proposition 6.6.5. *Let* $m := \min\{|f'(a)|, |f'(b)|\}$ *and* $M := \max\{|f'(a)|, |f'(b)|\}$*, then for every* $n > 0$

$$|x_i - \xi| \leq \frac{M - m}{m} \cdot |x_i - x_{i-1}|.$$

Proof

It follows that in step 3 we may test whether $|x_i - x_{i-}| \cdot (M - m) < \varepsilon \cdot m$, where $M - m$ and $\varepsilon \cdot m$ are constants precomputed outside the main loop. A word of warning: in some cases the estimate of Proposition 6.6.5 can be too large, resulting in far too many iterations being computed. This could easily kill the gain of reducing the number of function evaluations. In [68] it is suggested that one should just check whether the distance $|x_i - x_{i-1}|$ between two successive approximations is smaller than ε. However, in general, this does not guarantee that $|x_i - \xi| < \varepsilon$.

Proposition 6.6.6. *The rate of convergence of Algorithm 6.6.4 equals* 1.

Proof

One may weaken the assumptions on f in Algorithm 6.6.4, generalizing it to a much wider class of functions. The idea is to replace the fixed point z by the last term of the sequence $(x_i)_{i\in\mathbb{N}}$, where the function f took the opposite sign to $f(x_{i-1})$. This way, the algorithm to some extend resembles the bisection procedure. We present the following algorithm without a proof.

Algorithm 6.6.7. *Let $a < b$ and $f : [a, b] \to \mathbb{R}$ be a continuous function. Assume that $f(a) \cdot f(b) < 0$. This algorithm approximates a root of f in (a, b) up to a given accuracy $\varepsilon > 0$.*

1. *Set x_l and x_u to be the endpoints a, b, with the convention that $f(x_l) < 0$;*
2. *initialize the iteration counter $i := 1$ and $x_0 := x_u$;*
3. *repeat the following steps until x_i approximates the root of f with the requested accuracy:*
   a) *increment i;*
   b) *compute the next term of the sequence, setting:*

$$x_i = x_u - \frac{x_l - x_u}{f(x_l) - f(x_u)} \cdot f(x_u).$$

   c) *if $f(x_i) \cdot f(x_l) < 0$, then update the right endpoint, setting $x_u := x_i$, otherwise update the left endpoint, setting $x_l := x_i$;*
4. *return x_i.*

The secant method

In Algorithm 6.6.4 we approximated a root of f by constructing a sequence of secants, all of them passing through a fixed point $\big(z, f(z)\big)$. Formally, such a family of lines is called a *pencil*. We may improve the rate of convergence by relieving the condition that one point remains fixed. Instead, we shall always use the two previous terms of the sequence to construct the next secant. That is, in each iteration of the algorithm we shall construct a line through $\big(x_{i-1}, f(x_{i-1})\big)$ and $\big(x_{i-2}, f(x_{i-2})\big)$, where x_{i-1} and

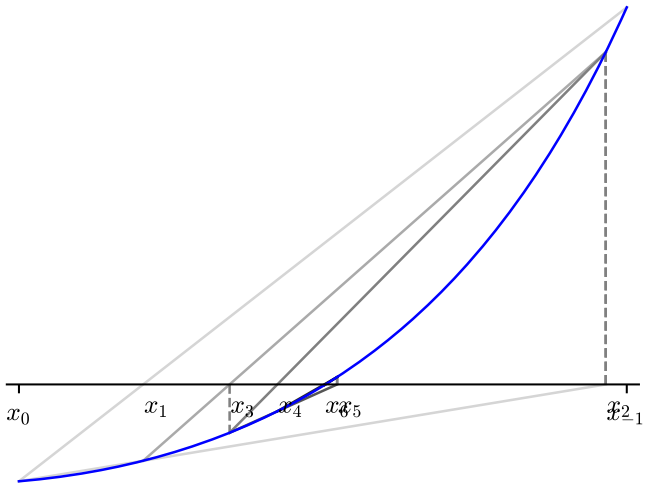


Figure 6.13: Successive approximations steps in the secant method

x_{i-2} are the last two approximations of the root obtained so far. Figure 6.13 illustrate, how the successive elements of the sequence are constructed. The same calculations as above yields:

$$x_i = x_{i-1} - \frac{x_{i-2} - x_{i-1}}{f(x_{i-2}) - f(x_{i-1})} \cdot f(x_{i-1}). \tag{6.21}$$

Thus, the new algorithm is very similar to the previous one, the only change is to use Eq. (6.21) in place of Eq. (6.20) in step 3b of Algorithm 6.6.4.

Algorithm 6.6.8. *Let $a < b$ and $f : [a, b] \to \mathbb{R}$ be a function of class C^2. Assume that $f(a) \cdot f(b) < 0$ and neither the first not the second derivative of f has a root in $[a, b]$. This algorithm approximates the (unique) root of f in (a, b) up to a given accuracy $\varepsilon > 0$.*

1. *Set x_{-1} and x_0 to be the two endpoint a, b of the considered intervals, with the convention that $|f(x_0)| \leq |f(x_{-1})|$;*
2. *initialize $i := 0$;*
3. *repeat the following steps until x_i approximates the root of f with the requested accuracy:*
   a) *increment i;*
   b) *compute the next term of the sequence $(x_i)_{i \geq -1}$ setting:*

$$x_i = x_{i-1} - \frac{x_{i-2} - x_{i-1}}{f(x_{i-2}) - f(x_{i-1})} \cdot f(x_{i-1});$$

4. *return x_i.*

The proof of correctness of this algorithm is nearly identical to that of Proposition 6.6.3. We leave it to the reader to fill in the details.

Remark 57. Implementing the secant method using floating-point numbers one needs to be careful, because Eq. (6.21) may potentially lead to catastrophic cancellation, when the successive iterations x_{i-1}, x_{i-2} are very close to each other. Therefore, an implementation ought to detect and evade this trap.

This algorithm is noticeably faster than regula falsi. With the same cost of a single step, it achieves a better convergence rate.

Proposition 6.6.9. *The rate of convergence of Algorithm 6.6.8 equals* $\frac{1}{2}(1+\sqrt{5}) \approx 1.618$.

Proof

Another advantage of the secant method, comparing to regula falsi, is that it is now easier to test whether x_i approximates the root ξ with a given accuracy. Since the rate of convergence is greater than one (we say that the method is *super-linear*), therefore near the root the distances between the approximations and the roots shrink faster than the actual distances between consecutive approximations. Hence, in order to ensure $|x_i - \xi| < \varepsilon$ (see Ineq. (6.18)), it suffices to check that $|x_i - x_{i-1}| < \varepsilon$.

improve this part

Newton–Raphson method

The algorithm described in this subsection is probably the beast known numerical root-finder. It is called *Newton–Raphson method* or sometimes just *Newton method*. Despite its name, it is not quite clear, who was the inventor of this technique. On one hand, [68] claims that the method was discovered first by I. Newton and independently later by J. Raphson[17]. On the other hand, according to [78] this algorithm was introduced for the first time by T. Simpson[18], while the methods described by I. Newton and J. Raphson were substantially different.

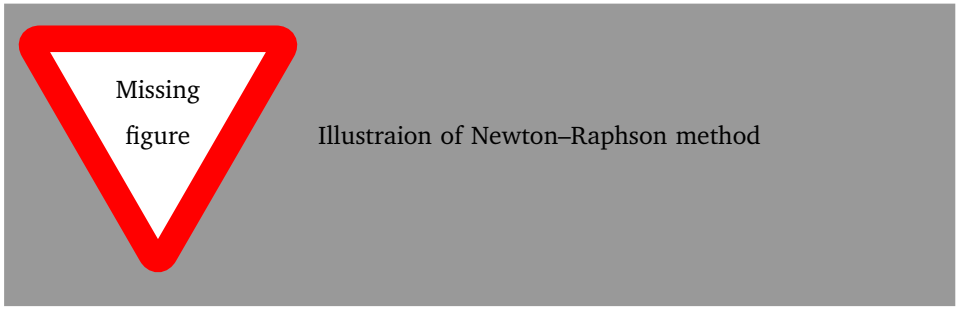
Recall the formula Eq. (6.21) from the secant method:

$$x_{i+1} = x_i - \frac{x_{i-1} - x_i}{f(x_{i-1}) - f(x_i)} \cdot f(x_i).$$

Observe that the fraction above is just the inverse of the difference quotient of f. If the sequence $(x_i)_{i\in\mathbb{N}}$ is convergent, then the differences between successive terms tend to zero. Thus, near a root of f this different quotient approximates the derivative of f. The idea of Newton–Raphson method is to replace this quotient by the derivative. Therefore, in this method we define a sequence of approximations of a root by the following recursive formula:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}. \tag{6.22}$$

[17] Joseph Raphson (born ca. 1648, died ca. 1715) was an English mathematician.
[18] Thomas Simpson (born 20 August 1710, died 14 May 1761) was a British mathematician.



Illustraion of Newton–Raphson method

Geometrically speaking, in place of a secant, determined by two previous terms, we take a tangent to the graph of f at the previous approximation x_i (see Figure 6.6). The starting point (i.e. the initial term of the sequence) is some $x_0 \in (a,b)$, called the *initial guess*. As we will see below (c.f. Proposition 6.6.13), Newton–Raphson method is faster than the two previous methods. This comes however with a price. We need to strengthen the assumptions to make sure that the sequence is convergent.

Theorem 6.6.10. *Let $a < b$ and $f : [a,b] \to \mathbb{R}$ be a function of class C^2. Assume that $f(a) \cdot f(b) < 0$ and neither f' nor f'' has a root in (a,b). If both tangent lines to the graph of f at the endpoints a, b have zeros inside (a,b), then for every initial guess $x_0 \in [a,b]$, the sequence $(x_i)_{i\in\mathbb{N}}$ converges to the (unique) root of f lying in $[a,b]$.*

Proof. As in the case of regula falsi and secant methods there are fours possible cases depending on the signs of the first and second derivatives of f. We will prove the assertion in case $f', f'' > 0$ on (a,b). The other cases are fully analogous and thus left as an exrecise for the reader.

Fix an element x_i of the sequence $(x_i)_{i\in\mathbb{N}}$ defined by Eq. (6.22). If x_i is actually equal to the root ξ of f, then

$$x_{i+1} = \xi - \frac{f(\xi)}{f'(\xi)} = \xi.$$

Hence, starting from the index i, the sequence is stationary. It is then clear that its limit is ξ. Thus, without loss of generality we may assume that x_i is not the root of f. It is either strictly less or strictly greater than ξ.

First assume that $x_i < \xi$. We claim that in this case $x_{i+1} \in (\xi, b)$. To see this let us apply Lagrange mean value theorem to f restricted to $[x_i, \xi]$. There is $\zeta \in (xi_i, \xi)$ such that the tangent to the graph of f ate ζ is parallel to the secant through x_i and ξ. Formally speaking

$$f'(\zeta) = \frac{f(x_i) - f(\xi)}{x_i - \xi}.$$

We have assumed that $f'' > 0$ and so f' is increasing. Consequently

$$f'(x_i) < f'(\zeta) = \frac{f(x_i) - f(\xi)}{x_i - \xi}.$$

This means that the slope of the tangent at x_i is lower than the slope of the secant at x_i and ξ (see Figure). Thus the tangent intersects the x-axis to the right of ξ. This shows that $x_{i+1} > \xi$. The same argument applied to f restricted to $[a, x_i]$ shows that $x_{i+1} < b$. missing

Finally we consider the case when x_i is greater than ξ. We claim that $x_{i+1} \in (\xi, x_i)$. Apply Lagrange mean theorem to f restricted to $[\xi, x_i]$. The same argument as above shows that the slope of the tangent at x_i is greater than the slope of the secant through ξ and x_i. Consequently the tangent intersects the x-axis in between ξ and x_i.

All in all, we have proved that the sequence $(x_i)_{i\geq 1}$ is decreasing and bounded from below by ξ. It follows that it must be convergent. It remains to show that its limit indeed equals ξ. Denote this limit by ζ. It follows from Eq. (6.22) that

$$\zeta = \zeta - \frac{f(\zeta)}{f'(\zeta)}.$$

Therefore $f(\zeta) = 0$ and so ζ is a root of f. But ξ is the only root of f in (a, b). Consequently $\xi = \zeta = \lim_{i\to\infty} x_i$. □

Since f is C^2-smooth and the sequence $(x_i)_{i\in\mathbb{N}}$ is stationary if $x_i = \xi$, there is a neighborhood of ξ shich satisfies the assumption about the zeros of the tangent lines. This leads to the following (more widely used) version of the above theorem.

Corollary 6.6.11. *Let $a < b$ and $f : [a, b] \to \mathbb{R}$ be a C^2-smooth function. There is a neighborhood U of the (unique) root ξ of f such that for every initial element $x_0 \in U$ the sequence is convergent to ξ.*

Let us write the algorithm in form of a pseudo-code.

Algorithm 6.6.12. *Let $a < b$ and $f : [a, b] \to \mathbb{R}$ be a C^2-smooth function, $x_0 \in [a, b]$ be an initial guess and $N \in \mathbb{N}$. Assume that f has a root in (a, b). This algorithm approximates the root of f in (a, b) up to a given accuracy $\varepsilon > 0$ or throws an exception, when the sequence $(x_i)_{i\in\mathbb{N}}$ is detected to be divergent.*

1. *initialize the counter setting $i := 0$;*
2. *repeat the following steps until x_i approximates the root of f with desired accuracy:*
   a) *increment the counter i;*
   b) *compute the next term of the sequence $(x_i)_{i\geq 0}$ setting:*

$$x_i = x_{i-1} - \frac{f(x_{i-1})}{f'(x_{i-1})}.$$

   c) *if either $x_i < a$ or $x_i > b$, then throw an exception and quit;*
   d) *if $i > N$, then check whether*

$$|f(x_i)| < |f(x_{i-1})| \qquad \text{and} \qquad |x_i - x_{i-1}| < |x_{i-1} - x_{i-2}|,$$

   if any of these this inequalities fails to hold, then throw an exception and quit.

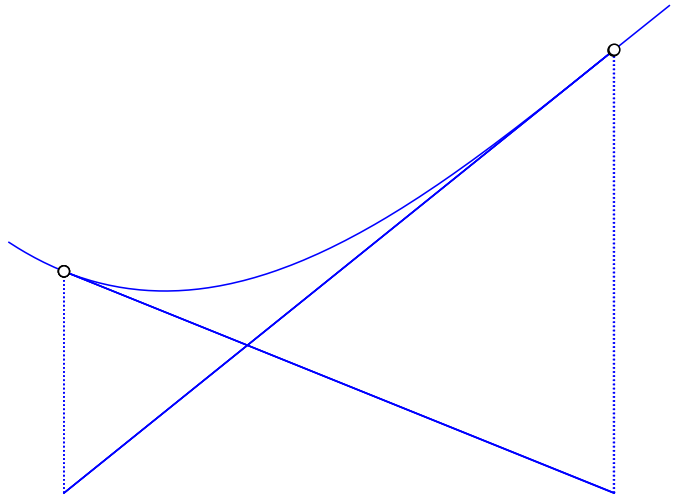


Figure 6.14: Example of an oscillation in Newton method.

3. return x_i;

The assumption of Theorem 6.6.10, that the tangents have zeros inside the interval (a, b) is indispensable. To see this, take f to be a polynomial $f = -x^3 + 2x + 2$. Then of course f is C^2 smooth. Its derivative $f' = -3x^2 + 2$ has two roots equal $\pm -\frac{1}{3}\sqrt{6} \approx -0.81650$. The second derivative has a unique roots at zero. But if we start with x_0 close enough to a root of a derivative of f, then the sequence $(x_i)_{i\in\mathbb{N}}$ will eventually oscillate between a neighborhoods of -1 and 0. For example take $x_0 = a = -0.815$ and $b = -0.1$. Then $x_1 = -125.230477815695$ lies outside our interval. If we continue computing the consecutive elements of the sequence we will see that for $i > 25$ we have $x_i \approx -1$ for odd i and $x_i \approx 0$ for even i (see Figure 6.14).

As observed above, the sequence $(x_i)_{i\in\mathbb{N}}$ may diverge. But if it converges it does so more rapidly than any of the previously described methods.

Proposition 6.6.13. *The rate of convergence of Algorithm 6.6.12 equals* 2.

Proof

Remark 58. Although the convergence rate of the secant method is evidently worse than of Newton–Raphson method, we must remember that the former one in each step needs only to evaluates f at x_i, while the latter one needs to evaluate both f and f'. Hence the cost of a single iteration in Newton–Raphson method is higher. Consequently, the speedup of this method compare to the secant method is not as big as the ratio of convergence rates could suggest.

As an example application, Newton–Raphson algorithm can be used to approximate a square root of a given positive number $a \in \mathbb{R}$ (this can be also easily generalize to complex numbers). This boils down to finding a root of a polynomial $f = x^2 - a$. The derivative of f is $f' = 2x$. Take the initial guess $x_0 := a/2$ and according to Eq. (6.22) define recursively

$$x_{i+1} := x_i - \frac{x_i^2 - a}{2 \cdot x_i} = \frac{1}{2} \cdot \left(x_i + \frac{a}{x_i} \right).$$

It follows from the above theorem, that this sequence converges to $\sqrt{a}$. As a side remark, let us note that although we derived this technique of approximating $\sqrt{a}$ based on Newton–Raphson algorithm, the method itself predates Newton–Raphson algorithm by at least 16 centuries and was described by Heron of Alexandria[19].

Improve convergence argument

Laguerre's method

Recall that in regula falsi, secant and Newton–Raphson methods, we operated under a 'dubious assumption', that the f reassembles a linear function near the root being approximated. Even when the assumption was not met, its recursive applications let us approximate the root accurately. In Laguerre's method, described here, another 'dubious assumption' is adopted, still leading to a method of approximating a root.

Laguerre's method is designed specifically for polynomials. Suppose that $f = f_0 + f_1 x + \cdots + f_n x^n$ is a polynomial with coefficients in some (computable) subring R of $\mathbb{C}$. By the fundamental theorem of algebra (Theorem 6.1.1), f has exactly n complex roots: $\xi_1, \xi_2, \ldots, \xi_n$, not necessarily distinct. As in the other methods described in this section, we construct a sequence $(x_i)_{i \in \mathbb{N}}$ intended to approximate a root of f. Suppose that it is ξ_1 that is being approximated. The 'dubious assumption' mentioned above is that ξ_1 is a unique simple roots of f, while all other roots are equal, i.e. $\xi_2 = \xi_3 = \cdots = \xi_n$. Fix a term x_i of the constructed sequence and denote

$$d := x_i - \xi_1, \qquad D := x_i - \xi_2 = \cdots = x_i - \xi_n. \tag{6.23}$$

Write the polynomial f as

$$f = f_n \cdot (x - \xi_1) \cdot (x - \xi_2)^{n-1}.$$

Compute the natural logarithm of both sides to get

$$\ln|f| = \ln|f_n| + \ln|x - \xi_1| + (n-1) \cdot \ln|x - \xi_2|.$$

Denote by g and $-h$, respectively the first and the second derivative of $\ln|f|$. Thus

$$g = \big(\ln|f|\big)' = \frac{1}{x - \xi_1} + \frac{n-1}{x - \xi_2}$$
$$h = -\big(\ln|f|\big)'' = \frac{1}{(x - \xi_1)^2} + \frac{n-1}{(x - \xi_2)^2}.$$

[19] Heron of Alexandria (born 10, died 70 AD) was a Greek mathematician and engineer of Roman era.

Using Eq. (6.23), the above formulas can rewritten as:

$$g = \frac{1}{d} + \frac{n-1}{D} \qquad \text{and} \qquad h = \frac{1}{d^2} + \frac{n-1}{D^2}$$

Solving the system for d, D we obtain:

$$d = \frac{n}{g \pm \sqrt{(n-1)(nh-g^2)}}, \qquad D = \frac{(n-1)\cdot d}{dg-1}.$$

In general, the square root in the denominator is a complex number. When working with floating-point numbers, in order to minimize the rounding errors, from the two possible denominators, we select the one with the larger absolute value. Now, if our 'dubious assumption' is accidentally true, then $x_i - d = \xi_1$ is a root of f. If not, then take it as the next term of our sequence and reiterate. All in all, the sequence in Laguerre's method is defined be the recursive formula

$$x_{i+1} := x_i - \frac{n}{g \pm \sqrt{(n-1)(nh-g^2)}}, \tag{6.24}$$

with some starting point (initial guess) $x_0 \in \mathbb{C}$.

Remark 59. We derived the above recursive formula using the assumption that all the roots of f but ξ_1 are equal. Since we needed only the absolute value of f, thus the condition can be somehow mitigated. We may assume instead that the all the roots (but ξ_1) are equidistant from x_i and still derive the same formula. Nevertheless, such an assumption is not much less dubious than the previous one.

Like with Newton–Raphson method, it is possible that the sequence will not converge. But if it does, then by its construction, we immediately have:

Observation 6.6.14. *If, for a given initial guess x_0, the sequence $(x_i)_{i\in\mathbb{N}}$ is convergent, then its limit is a root of f.*

Unlike with Newton–Raphson method, though, the sequence will diverge very rarely. The following two facts are known, however their proofs exceed the scope of these notes and so we omit them.

provide refs

Proposition 6.6.15. *If all the roots of a polynomial f are real, then for every initial guess $x_0 \in \mathbb{R}$, the sequence $(x_i)_{i\in\mathbb{N}}$ defined by Eq. (6.24) converges to a root of f.*

It is not an accident that the assumptions of the above proposition is the same as that of Theorem 6.3.21. On the other hand, to the best of the knowledge of the author of these notes, there is no known criterion for convergence of Laguerre's method in complex case. Yet, [68] says that the "nonconvergence is extremely unusual and, further, can almost always be fixed by a simple scheme to break a nonconverging limit cycle".

Proposition 6.6.16. *If the sequence $(x_i)_{i\in\mathbb{N}}$ converges to a simple root of f (i.e. root of multiplicity 1), then the rate of convergence equals 3.*

Finally, in order to write the algorithm explicitly, let us notice that

$$g(x_i) = \big(\ln|f(x_i)|\big)' = \frac{f'(x_i)}{f(x_i)} \qquad \text{and} \qquad h(x_i) = -g'(x_i) = \big(g(x_i)\big)^2 - \frac{f''(x_i)}{f(x_i)}.$$

Algorithm 6.6.17. *Given a polynomial $f \in R[x]$, an accuracy $\varepsilon > 0$ and some initial guess x_0, this algorithm returns an approximation of a root of f with the desired accuracy or throws an exception, if the initial guess is badly chosen.*

1. *Initialize the counter $i := 0$;*
2. *repeat the following steps until the requested accuracy of the root approximation is achieved:*
   a) *set $g := \frac{f'(x_i)}{f(x_i)}$ and $h := g^2 - \frac{f''(x_i)}{f(x_i)}$;*
   b) *set $s := \sqrt{(n-1)(nh-g^2)}$ and $v := g + s$;*
   c) *if $\mathrm{re}(\overline{g}\cdot s) < 0$, then $|g-s| > |v|$, hence update v, setting $v := v - 2s$;*
   d) *compute the next term of the sequence setting:*

$$x_{i+1} := x_i - \frac{n}{v},$$

   e) *increment the counter i;*
   f) *if i exceeded the maximal allowable number of iterations throw an exception;*
3. *return x_i.*

roots by eigenvalues, Brent's method

Quadratic Interval Refinements

The algorithms described so far: regula falsi, secant method, Netwon–Rapshon and Laguerre's method, are all designed specifically for operating on floating-point, finite-precision numbers, whereas using them with exact arithmetic may easily lead to coefficients of enormous sizes. In contrast, bisection, presented on the beginning of this section, due to its simplicity, is equally well suited to be used with floating-point and exact arithmetic. But bisection is slow. Here we present another algorithm, that has a quadratic rate of convergence (like Newton–Raphson method), but designed with exact arithmetic in mind.

The basic idea of *quadratic interval refinement* (shortly QIR) algorithm is to blend the bisection method with regula falsi. Like the bisection, this algorithm divides the interval containing a root of f, but not just to two sub-intervals (as the bisection does), but to N sub-intervals. Then, using linear interpolation (in a style of regula falsi), it tries to estimate which sub-interval contains the root. If the guess is correct, then in

the next iteration the updated interval is divided into N^2 sub-intervals, otherwise the number of subdivisions is reduced to $\sqrt{N}$, while interval remains unchanged. Like in the bisection method, the successive intervals always bracket a root. Hence this algorithm must succeed.

The main algorithm needs two sub-procedures to subdivide the interval and estimate the correct sub-interval. The first algorithm below is a general one, while the second is a fail-safe procedure for $N = 4$.

Algorithm 6.6.18 (This is an auxiliary procedure for Algorithm 6.6.20)**.** *Given a function f, integer $N > 4$ and an open interval (a, b) such that $a, b \in \mathbb{Q}$ and $f(a) \cdot f(b) < 0$, this algorithm returns one of the three possible values: either an exact root $\xi \in (a, b)$ of f, or the pair $\big(\text{SUCCESS}, (\alpha, \beta)\big)$, or a flag* FAILURE*. In the second case, α, β are rational numbers such that $a \leq \alpha < \beta \leq b$, f changes sign on (α, β) and $(b - a) = N \cdot (\beta - \alpha)$.*

1. *Subdivide the interval (a, b) into N sub-intervals of equal lengths, denoting the endpoints by $\alpha_0 := a, \alpha_1, \ldots, \alpha_N := b$, that is $\alpha_i := a + i \cdot \delta$, where $\delta := {}^{(b-a)}/_{N}$;*
2. *find the endpoint α_k that is closest to the secant of the graph of f at a and b, that is set*
$$k := \left[\frac{N \cdot f(a)}{f(a) - f(b)} \right],$$
here the square bracket means rounding to the nearest integer;
3. *evaluate f at α_k, if $f(\alpha_k) = 0$, then return the exact root α_k and quit;*
4. *if $f(\alpha_k)$ has the same sign as $f(a)$, then evaluate $f(\alpha_{k+1})$ and:*
   a) *if $f(\alpha_{k+1}) = 0$, then return the exact root α_{k+1} and quit;*
   b) *if f changes sign at (α_k, α_{k+1}) then return $\big(\text{SUCCESS}, (\alpha_k, \alpha_{k+1})\big)$, otherwise return* FAILURE*;*
   c) *quit;*
5. *if the signs of $f(\alpha_k)$ and $f(a)$ differ, then evaluate $f(\alpha_{k-1})$ and:*
   a) *if $f(\alpha_{k-1}) = 0$, then return the exact root α_{k-1} and quit;*
   b) *if f changes sign at (α_{k-1}, α_k) then return $\big(\text{SUCCESS}, (\alpha_{k-1}, \alpha_k)\big)$, otherwise return* FAILURE*;*

Algorithm 6.6.19 (This is a second auxiliary procedure for Algorithm 6.6.20)**.** *Given a function f and an open interval (a, b) such that $a, b \in \mathbb{Q}$ and $f(a) \cdot f(b) < 0$, this algorithm return one of the three possible values: either an exact root $\xi \in (a, b)$ of f, or the pair $\big(\text{SUCCESS}, (\alpha, \beta)\big)$, or a pair $\big(\text{FAILURE}, (\alpha, \beta)\big)$. In all cases, except the first one, α, β are rational numbers such that $a \leq \alpha < \beta \leq b$, $f(\alpha) \cdot f(\beta) < 0$ and $(b - a) = 4 \cdot (\beta - \alpha)$.*

1. *Subdivide the interval (a, b) into **four** sub-interval of equal length, denoting the endpoints by $\alpha_0 := a, \alpha_1, \ldots, \alpha_4 := b$, that is $\alpha_i := a + i \cdot \delta$, where $\delta := {}^{(b-a)}/_{4}$;*

2. *find the* ***left endpoint*** α_k *of the interval* (α_k, α_{k+1})*, where the secant of the graph of* f *at* a *and* b *intersects* OX*, that is set*

$$k := \left\lfloor \frac{4 \cdot f(a)}{f(a) - f(b)} \right\rfloor;$$

3. *bisect the interval* (a, b) *twice (i.e. perform two iterations of Algorithm 6.6.1) to obtain a new interval* (α, β)*, where* f *is guaranteed to change sign (or exceptionally an exact root of* f*);*
4. *if the bisection produced an exact root* ξ *of* f*, then return it and quit;*
5. *if the interval* (α, β) *outputted by the bisection is the same as* (α_k, α_{k+1})*, then return* $\big(\text{SUCCESS}, (\alpha, \beta)\big)$*;*
6. *otherwise return* $\big(\text{FAILURE}, (\alpha, \beta)\big)$.

We are now ready to present the main procedure of QIR algorithm.

Algorithm 6.6.20. *Let* a, b *be rational numbers and let* $f : [a, b] \to \mathbb{R}$ *be a continuous function such that* $f(a) \cdot f(b) < 0$*. Given an accuracy* $\varepsilon > 0$*, this algorithm approximates a root* $\xi \in (a, b)$ *of* f *with the requested accuracy.*

1. *Initialize the refinement factor* $N := 4$*;*
2. *as long as* $b - a > \varepsilon$ *repeat the following steps:*
   a) *if* $N = 4$*, then do the following:*
      i. *execute Algorithm 6.6.19;*
      ii. *if an exact root was found, then simply return it and quit;*
      iii. *if the 'refinement step' (i.e. step 2(a)i) was successful, then set* $N := 16$*;*
      iv. *update* a, b *setting them to* α, β *return by Algorithm 6.6.19;*
   b) *otherwise, when* $N > 4$*, do the following:*
      i. *execute Algorithm 6.6.18;*
      ii. *if an exact root was found, return it and quit;*
      iii. *if the step 2(b)i was successful, then update* a, b *by the returned* α, β*, and set* $N := N^2$*;*
      iv. *otherwise set* $N := \max\big(\sqrt{N}, 4\big)$*;*
3. *return* a.

Since the refinement factor N is squared after each successful subdivision, thus it does not come as a big surprise that the convergence rate of QIR is 2.

Proposition 6.6.21. *The rate of convergence of Algorithm 6.6.20 equals* 2.

Proof + example

Remark 60. Although QIR algorithm is specifically devised for using with exact arithmetic, there is a modification (known as A-QIR, that stands for Approximate Quadratic Interval Refinement) fine-tuned for use with floating-point numbers. For details see [39].

Example: QIR vs N-R/L (size of terms)
Convergence rates
Jenkins–Traub algorithm, Aberth–Ehrlich algorithm, Durand–Kerner algorithm, Schönhage's algorithm (A. Schönhage. The fundamental theorem of algebra in terms of computational complexity. Technical report, Univ. Tübingen, 1982.)
Bairstow method

Examples

In this subsection we present few examples demonstrating how the described methods behave. First we present the polynomials and the intervals on which the roots are approximated:

f_i	a	b
$f_0 = 3 - 3x - \frac{9}{2}x^2 + x^3$	0	1
$f_1 = 1 - 3x + x^3$	1	10
$f_2 = -1 + 4x - 5x^3 + x^5$	1.7	3
$f_3 = 73 - 66x - 21x^2 + 12x^3 + 3x^4$	1.6	3
$f_4 = -4031 + 3768x + 1182x^2 - 759x^3 - 177x^4 + 3x^6$	4	10
$f_5 = -1 + x^2 - 13x^3 + x^7$	1.6	2

The next table summarizes number of iterations needed by various algorithms to approximate the roots within the tolerance of $\varepsilon = 10^{-20}$. The computations were interrupted if the accuracy was not met after 1 000 iterations.

	f_0	f_1	f_2	f_3	f_4	f_5
Bisection	68	71	68	68	71	67
Regula falsi	24	>1000	179	108	231	33
Secant	8	27	14	12	15	10
Newton–Raphson	6	12	9	8	7	6
Laguerre	4	5	5	>1000	4	48
QIR	7	10	7	7	8	7

Next we approximate the roots of the same polynomials on the same intervals, but this time the request one thousand decimal digits of accuracy (i.e. $\varepsilon = 10^{-1000}$). The following table summarizes the number of iterations needed by various described algorithms (the maximum allowable number of iterations this time was 10 000).

	f_0	f_1	f_2	f_3	f_4	f_5
Bisection	3323	3327	3324	3324	3326	3322
Regula falsi	1102	>10000	8974	5371	>10000	1595
Secant	17	35	22	20	23	19
Newton–Raphson	12	17	15	14	13	12
Laguerre	>10000	9	8	8	7	2967
QIR	12	15	12	12	13	12

6.7 Higher degree solvable polynomials

6.8 Strategies for solving univariate polynomial equations

6.9 Thom's lemma

Bibliographic notes and further reading

Subsection "Generating root bounds" is based on [90, 69]. Theorem 6.3.12 is a variant of theorems presented in [2, 80, 86]. Proposition 6.3.14 is a special case of a bound introduced in [33] for *multivariate* polynomials. The linear algorithm for computing Hong's bound is based on a preliminary, unpublished version of a paper [46] (see also [60]). Algorithm 6.3.13 comes from P. Vigklas' Ph.D. thesis [86]. For a detailed analysis of time complexity of a real root isolation by a continued-fractions method (Algorithm 6.5.23) we refer the reader to [85].

Appendix

This is a Sage code computing the form of the polynomial h in Eq. (6.5).

```
# a helper function for expressing a symmetric
# polynomial s using elementary symmetric
# polynomials sigma1,...,sigma4
# Warning: this version works only for 4 variables!
def to_sigmas( s ) :
    S.<sigma1,sigma2,sigma3,sigma4> = QQ[]
    E = 0
    for (e,alpha) in list(s) :
        L = list(e)
        if all(j < 5 for j in L) :
            E += alpha*prod( S.gens()[j-1] for j in L )
    return E

# define the ring of coefficients
P.<p,q,r, b0,b1,b2,b3, a0,a1,a2,a3> = QQ[]
# and its subring
```

```
Pa = QQ[a0,a1,a2,a3]
# define the polynomial ring
R_.<x> = PolynomialRing(P, 1, order='neglex')
R.<x> = P[]
# the depressed polynomial g
g = R([r,q,p,0,1]); g_ =  R_(g)
html("<p>The depressed polynomial:"
"$$g = " + latex(g_) + "$$</p>")

# the auxiliary polynomial h
h = (x-b1^2)*(x-b2^2)*(x-b3^2); h_ = R_(h)
html("<p>The auxiliary polynomial:"
"$$h = " + latex(h_) + "$$</p>")

# the transformation matrix
M = 1/2*matrix(QQ,[
[1,1,1,1],
[1,-1,1,-1],
[1,1,-1,-1],
[1,-1,-1,1]
])
html("<p>The transformation matrix:"
"$$M = " + latex(M) + "$$</p>")
# compute the transformation
V = M*vector([a0,a1,a2,a3])
# substitute [b0,...,b3] by M*[a0,...,a3]
h = R([ hi(b0 = V[0], b1 = V[1], b2 = V[2], b3 = V[3])
    for hi in h ])
# uncomment the next line, to show h in full form
#html("$$h = " + latex(R_(h)) + "$$")

# express the coefficients of h using elementary
# symmetric polynomials sigma1,...,sigma4
S = SymmetricFunctions(QQ)
e = S.elementary()
h0 = S.from_polynomial( Pa(h[0]) )
h0 = to_sigmas( e(h0) )
h1 = S.from_polynomial( Pa(h[1]) )
h1 = to_sigmas( e(h1) )
h2 = S.from_polynomial( Pa(h[2]) )
h2 = to_sigmas( e(h2) )
html("<p>The coefficients of $h$ in terms of elementary"
" symmetric polynomials:$$\\begin{cases}"
"h_0 = " + latex( h0 ) + "\\\\"
```

```
"h_1 = " + latex( h1 ) + "\\\\"
"h_2 = " + latex( h2 ) + "\\end{cases}$$</p>")

# simplify the expressions using the fact that
# \sigma_1 = 0 (since g is depressed)
html("<p>The coefficients of $h$ after substituting "
"$\\sigma_1=0$ $$\\begin{cases}"
"h_0 = " + latex(h0(sigma1 = 0)) + "\\\\"
"h_1 = " + latex(h1(sigma1 = 0)) + "\\\\"
"h_2 = " + latex(h2(sigma1 = 0)) + "\\end{cases}$$</p>")

# substitute the coefficients of g into h
# (use Viete's formulae)
h0 = h0(sigma1 = 0, sigma2 = p, sigma3 = -q, sigma4 = r)
h1 = h1(sigma1 = 0, sigma2 = p, sigma3 = -q, sigma4 = r)
h2 = h2(sigma1 = 0, sigma2 = p, sigma3 = -q, sigma4 = r)
html("<p>Using Viete's formulae: "
"$$\\begin{cases}"
"h_0 = "+latex(h0) + "\\\\"
"h_1 = "+latex(h1) + "\\\\"
"h_2 = "+latex(h2)  + "\\end{cases}$$</p>")

# the resulting form of the polynomial h
h = h0 + h1*x +h2*x^2 + x^3; h_ = R_(h)
html("<p>The resulting form of $h$:"
"$$h = " + latex(h_) + "$$</p>")
```

7

Integration

7.1 Symbolic integration of rational functions

In this section we learn how to compute the antiderivative of a rational function. We need to begin with a special case when the function is actually a polynomial. In this case the algorithm is very similar to the one used to compute the derivative (see Algorithm 1.1.30) and as such does not require any special explanations:

Algorithm 7.1.1. *This algorithm computes an antiderivative F of a polynomial $f = f_0 + f_1 t + \cdots + f_m t^m$.*

1. *Set $F_0 := C$, where C is a new symbolic constant;*
2. *for each $0 \leq i \leq m$, take $F_{i+1} := {}^1/_{i+1} \cdot f_i$;*
3. *return $F = F_0 + F_1 t + \cdots + F_{m+1} t^{m+1}$.*

Now, we turn our attention to an arbitrary rational function defined over some (computable) subfield of $\mathbb{C}$. Recall from Section 5.4 of Chapter 5, that any rational function ${}^f/_g$ decomposes uniquely into a sum

$$\frac{f}{g} = f_0 + \frac{f_{11}}{g_1} + \frac{f_{21}}{g_2} + \frac{f_{22}}{g_2^2} + \cdots + \frac{f_{ij}}{g_i^j} + \cdots + \frac{f_{kk}}{g_k^k}, \tag{7.1}$$

where $g_1, \ldots, g_k$ are the (monic) square-free factors of the denominator g. The antiderivative is a linear operator, hence the task reduces to computing $\int \frac{a}{b^n}$, where a, b are relatively prime and b is monic, square-free. The following procedure, due to C. Hermite, let us get rid of the exponent n, reducing the problem even further. The polynomial b is square-free and so Theorem 5.3.2 asserts that b and b' are relatively prime. It follows from Bézout identity (see Theorem 1.3.4), that there are polynomials c, d such that $1 = c \cdot b + d \cdot b'$. Therefore we can write

$$\int \frac{a}{b^n} = \int \frac{acb + adb'}{b^n} = \int \frac{ac}{b^{n-1}} + \int \frac{adb'}{b^n},$$

integration by part yields:

$$= \int \frac{ac}{b^{n-1}} + \frac{ad}{(1-n)b^{n-1}} - \int \frac{(ad)'}{(1-n)b^{n-1}}$$
$$= \frac{\frac{1}{1-n}ad}{b^{n-1}} + \int \frac{ac + \frac{1}{n-1}(ad)'}{b^{n-1}}.$$

prove that num and denom are rel. prime

This way we reduced the exponent in the denominator by one. Repeating the above steps, we eventually end up with an expression:

$$\int \frac{a}{b^n} = \frac{\frac{1}{2}a_2}{b^2} + \frac{\frac{1}{3}a_3}{b^3} + \cdots + \frac{\frac{1}{n-1}a_{n-1}}{b^{n-1}} + \int \frac{B}{b},$$

for some polynomials $B, a_2, \ldots, a_{n-1}$. Put the terms preceding the integral over a common denominator and write

$$\int \frac{a}{b^n} = \frac{A}{b^{n-1}} + \int \frac{B}{b}, \tag{7.2}$$

where $A = {}^1/_2 \cdot a_2 b^{n-3} + {}^1/_3 \cdot a_3 b^{n-4} + \cdots + {}^1/_{n-1} a_{n-1}$. This way we have just proved a correctness of the following algorithm:

Algorithm 7.1.2 (Hermite's reduction). *Given a rational function ${}^a/_{b^n}$ with the square-free denominator, this algorithm returns rational functions ${}^A/_{b^{n-1}}$ and ${}^B/_b$ such that Eq.* (7.2) *holds.*

1. *Initialize $A := 0$, $B := a$ and $i := n-1$;*
2. *Using Algorithm 1.3.6, find polynomials c, d such that*
$$1 = \gcd(b, b') = c \cdot b + d \cdot b';$$
3. *as long as $i \geq 1$ proceed as follows:*
   a) *add $-{}^1/_i \cdot B \cdot d \cdot b^{n-i-1}$ to A;*
   b) *update the numerator setting $B := B \cdot c + \frac{1}{i} \cdot (B \cdot d)'$;*
   c) *decrement i;*
4. *compute the quotient q and the remainder r of B mod b;*
5. *use Algorithm 7.1.1 to compute the antiderivative Q of q;*
6. *update the numerator of the remaining integral, setting $B := r$;*
7. *update the numerator of integrated part, setting $A := A + Q \cdot b^{n-1}$;*
8. *return ${}^A/_{b^{n-1}}$ and the reduced integral $\int {}^B/_b$.*

Example 80. Take a function

$$\frac{f}{g} = \frac{240 + 72t - 72t^2 - 48t^3}{27 + 27t^2 + 9t^4 + t^6} = \frac{240 + 72t - 72t^2 - 48t^3}{(3 + t^2)^3}.$$

The algorithm then returns:

$$A = \frac{59t + 12t^2 + 7t^3 - 2t^4}{(3 + t^2)^2} \qquad \text{and} \qquad B = \frac{7}{3 + t^2}.$$

Applying the above algorithm to every term of Eq. (7.1), we eventually have to deal only with finitely many antiderivatives of functions of a form h/g, where g is monic and square-free, i.e. all the roots of g in $\mathbb{C}$ are distinct. The final step of the integration is now provided by the following proposition.

Proposition 7.1.3. *Let h be a polynomial and let $g = (t - z_0) \cdots (t - z_n)$, where $z_0, \ldots, z_n \in \mathbb{C}$ are distinct. Then*

$$\int \frac{h}{g} = \sum_{i=0}^{n} \frac{h(z_i)}{g'(z_i)} \cdot \ln(t - z_i).$$

proof

Now, at our disposal are all the tools needed to compute the antiderivative of a rational function f/g. Assume that we are able to find the roots of the square-free factors of the denominator g. The whole process boils then down to the following procedure:

Algorithm 7.1.4. *Let f, g be two relatively prime polynomials with coefficients in a subfield of $\mathbb{C}$. Assume that the roots of g are computable. One computes the antiderivative $\int f/g$ as follows:*

1. *Using Algorithm 5.4.4 compute the complete square-free PFD of f/g:*

$$\frac{f}{g} = f_0 + \frac{f_{11}}{g_1} + \frac{f_{21}}{g_2} + \frac{f_{22}}{g_2^2} + \cdots + \frac{f_{ij}}{g_i^j} + \cdots + \frac{f_{kk}}{g_k^k};$$

2. *compute the antiderivative $F_0 := \int f_0$ using Algorithm 7.1.1;*
3. *iterate over the non-polynomial terms f_{ij}/g_i^j:*
   a) *apply Algorithm 7.1.2 to f_{ij}/g_i^j to get A_{ij} and B_{ij} such that*

$$\int \frac{f_{ij}}{g_i^j} = \frac{A_{ij}}{g_i^{j-1}} + \int \frac{B_{ij}}{g_i};$$

   b) *compute the roots $z_0, \ldots, z_n$ of g_i using techniques described in Chapter 6;*

c) compute the antiderivative of f_{ij}/g_i^j *setting:*

$$H_{ij} := \frac{A_{ij}}{g_i^{j-1}} + \sum_{l=0}^{n} \frac{B_{ij}(z_l)}{g_i'(z_l)} \cdot \ln(t - z_l);$$

4. *return the antiderivative of* f/g*:*

$$\int \frac{f}{g} = F_0 + \sum_{i=1}^{k} \sum_{j=1}^{i} H_{ij}.$$

Example 81. Take a rational function:

$$\frac{f}{g} = \frac{504 + 144t + 72t^3}{-36 - 24t + 23t^2 + 27t^3 + 9t^4 + t^5}.$$

The SPFD of f/g is

$$\frac{5}{(-1)\cdot(1-t)} + \frac{42-5t}{(2+t)\cdot(3+t)} + \frac{-576-348t}{(2+t)^2\cdot(3+t)^2}.$$

Now, our algorithm yields:

$$\int \frac{f}{g} = \frac{2880 + 2892t + 696t^2}{6 + 5t + t^2} + 531\log(t+3) - 536\log(t+2) + 5\log(t-1).$$

at least: Rothstein–Trager theorem,
maybe: Trager–Lazard–Rioboo subresultant theorem,
hopefully: Risch theorem

7.2 Classical numeric integration schemes

7.3 Monte Carlo integration

8

Classical elimination theory

Eliminate, eliminate, eliminate,
Eliminate the eliminators of
elimination theory...

"Polynomials and Power Series",
S.S. Abhyankar

Now, when we have learned how to deal with univariate polynomial equations, we dive into deep water and look upon multivariate ones. In this chapter (and the two following it) we develop methods for solving systems of multivariate polynomial equation by eliminating variables. Before we dig into the details, let us consider one illustrative example. It should give a reader some insight and a chance to build an intuition on what the elimination theory is all about.

Suppose we are given a system of two polynomial equation (over $\mathbb{R}$) in variables x, y:

$$\begin{cases} 3x^2 - 3xy - 10x + 3y^2 + 5 = 0 \\ x^2 - 5xy + y^2 + 6y - 3 = 0. \end{cases} \tag{8.1}$$

Each row of this system may be viewed as an equation of a plane algebraic curve (see Figure 8.1). The points where these two curves intersect form the set of solutions to the system Eq. (8.1). As its name suggests, the elimination theory provides methods for removing variables from the system, so that one ends up with univariate polynomials only (more or less one per variable). Using methods described in subsequent sections, we may for instance eliminate variable y from the above system and obtain a single univariate polynomial equation in x, called an *eliminant*:

$$36x^4 - 258x^3 + 613x^2 - 583x + 184 = 0. \tag{8.2}$$

Its roots are precisely the abscissas of the solutions to Eq. (8.1). There are 4 of them, as the polynomial is square-free, equal approximately:

$$\{0.645, 1.44, 1.57, 3.52\}.$$

We can plug these values, one by one, into our original system and solve the resulting univariate polynomial equation(s) for y. For example, substitution of 0.645 for x in Eq. (8.1) yields

$$\begin{cases} 3.00y^2 - 1.93y - 0.201 = 0 \\ y^2 + 2.78y - 2.58 = 0. \end{cases}$$

The resulting system has a unique solution $y \approx 0.736$, so that $(x, y) \approx (0.645, 0.736)$ is one of the zeros of Eq. (8.1). One finds the other solutions in exactly the same way.

As simple as the above "back-solving" procedure is, it may easily suffer from accumulation of rounding errors (in Section 10 we discuss methods for improving accuracy of the final solutions). Even if the roots of the eliminant are approximated with a very high precision, they are still not exact, in general. Thus, after substitution we obtain polynomials which has only approximate coefficients and so their roots are not quite the values we are after. Even worse, in general, we cannot even find the roots exactly, but we must again employ a numerical root-finder and so further reduce accuracy. The situation deteriorates with each variable and it can easily get out of hand, if there are too many of variables to deal with.

Therefore, instead of substituting computed solutions for x, we may in turn eliminate this variable from Eq. (8.1), to get an equation in y alone:

$$36y^4 - 114y^3 + 73y^2 + 29y - 26 = 0.$$

The roots of this equation are the ordinates of the solutions of the original system. Approximately:

$$\{-0.536, 0.736, 0.875, 2.09\}.$$

Now we can try out (numerically) all feasible combinations of x and y to find out the solutions of Eq. (8.1):

	0.645	1.44	1.57	3.52
2.09			✓	
0.875				✓
0.736	✓			
-0.536		✓		

All in all, we can think of the elimination theory as a method of projecting solutions of polynomial systems onto the axes of the coordinate system. This is a reasonably good first step toward grasping the theory.

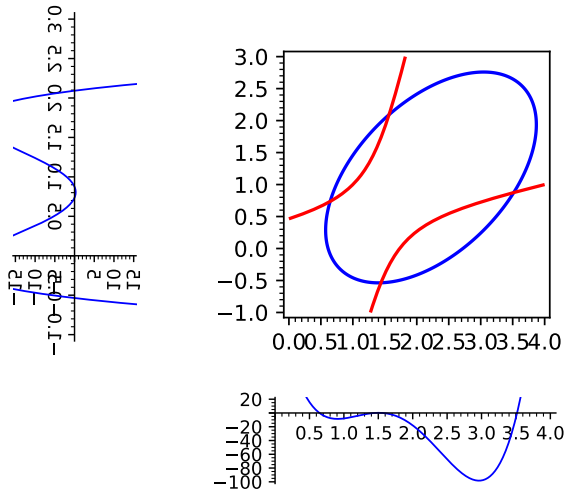


Figure 8.1: Curves described by rows of Eq. (8.1) and the graphs of the eliminants.

8.1 Labatie–Kalkbrener triangulation

In Section 1.3 of Chapter 1 we introduced a notion of polynomial pseudo-remainder sequences (prs), that we used for computing greatest common divisors of polynomials over a unique factorization domain. But this is not the only application of prs. It turns out that computation of gcd is closely related to solving a system of polynomial equations. We will observe how they are related in a couple of places in this chapter. First is a method of triangulating a system of two polynomial equations, that was first discovered by A.-G. Labatie[1] in early 19$^{\text{th}}$ century in a special case of two bivariate polynomials. It was then forgotten for a long time and independently rediscovered by M. Kalkbrener in his Ph.D. thesis in 1991, in a more general context, for systems of arbitrary many variables. (Nevertheless, this method is most efficient for polynomial systems with a few variables.)

Let R be a unique factorization domain for which we have a method of computing gcd. In practice, this will always be a ring of polynomials in one variables less than the system we are solving. Recall (cf. Proposition 1.1.15) that for every two polynomials $f_0, f_1 \in R[x]$ with coefficients in R there are unique polynomials $q, r \in R[x]$—called the pseudo-quotient and pseudo-remainder—satisfying

$$\operatorname{lc}(f_1)^{\deg f - \deg g + 1} \cdot f_0 = q \cdot f_1 + r \qquad \text{and} \qquad \deg r < \deg f_1.$$

Suppose we are given two primitive (recall that this means that their contents are trivial, see Definition 1.3.8) polynomials f_0, f_1 with coefficients in the ring R and as-

[1] Anthilde-Gabriel Labatie (born 1786, died 1866) was a French mathematician.

sume that $\deg f_0 \geq \deg f_1$. Their primitive pseudo-remainder sequence (see page 47) consists of polynomials $f_0, f_1, f_2, \ldots, f_k, 0$, constructed as follows:

$$\begin{aligned} l_1^{d_1} \cdot f_0 &= q_1 \cdot f_1 + c_1 \cdot f_2 \\ l_2^{d_2} \cdot f_1 &= q_2 \cdot f_2 + c_2 \cdot f_3 \\ &\vdots \\ l_{k-1}^{d_{k-1}} \cdot f_{k-2} &= q_{k-1} \cdot f_{k-1} + c_{k-1} \cdot f_k \\ l_k^{d_k} \cdot f_{k-1} &= q_k \cdot f_k + 0. \end{aligned} \tag{8.3}$$

Here, $l_j := \operatorname{lc}(f_j) \in R$ is the leading coefficient of f_j and $d_j := \deg f_{j-1} - \deg f_j + 1$. Next, q_j is the pseudo-quotient and $r_j = c_j f_{j+1}$ the pseudo-remainder of f_{j-1} modulo f_j, with $f_{j+1} = \operatorname{pp}(r_j)$ being the primitive part and $c_j \in R$ the content of r_j. The polynomial f_k is then the greatest common divisor of f_0 and f_1 (up to multiplication by a unit), since by assumption $\operatorname{cont} f_0 = \operatorname{cont} f_1 = 1$. In particular:

Observation 8.1.1. *The primitive polynomials f_0 and f_1 are co-prime if and only if f_k is a unit.*

We now define two sequences $g_1, \ldots, g_k$ and $e_1, \ldots, e_k$ of elements of R. First, let

$$g_1 := \gcd\left(l_1^{d_1}, c_1\right) \qquad \text{and} \qquad g_j := \gcd\left(\frac{l_1^{d_1} \cdots l_j^{d_j}}{g_1 \cdots g_{j-1}}, c_j\right), \quad \text{for } j > 1.$$

Further set $e_j := {}^{c_j}\!/_{g_j}$ for $j \in \{1, \ldots, k\}$. By definition (see p. 43) of a zero polynomial is 1 and so $c_k = g_k = e_k = 1$.

Definition 8.1.2. The sequence $\mathscr{L} = (e_1, \ldots, e_k)$ obtained above is called the *Labatie's elimination sequence* (or shortly *L-elimination sequence*) for f_0 and f_1. Elements $e_1, \ldots, e_k$ are called *Labatie's eliminants* (or shorty *L-eliminants*).

A straightforward modification of Euclidean algorithm (cf. Algorithm 1.3.16) computes Labatie's elimination sequence directly from definition:

Algorithm 8.1.3. *Let R be a ring in which* gcd *is computable. Given two polynomials f_0, f_1 with coefficients in R this algorithm returns their primitive pseudo-remainder sequence together with the Labatie's elimination sequence.*

1. *make sure that the degree of f_1 does not exceed the degree of f_0, if it is not the case swap the two polynomials;*
2. *initialize $j := 1$, $l := 1$ and $g := 1$;*
3. *as long as f_j is non-zero proceed as follows:*
   a) *compute the pseudo-quotient q_j and pseudo-remainder r_j of $f_{j-1} \bmod f_j$;*
   b) *split the pseudo-remainder into its primitive part $f_{j+1} := \operatorname{pp}(r_j)$ and its content $c_j := \operatorname{cont}(r_j)$;*

c) let l_j be the leading coefficient of f_j and $d_j := \deg f_{j-1} - \deg f_j + 1$;

d) update the product of powers of leading coefficients, setting $l := l \cdot l_j^{d_j}$;

e) set $g_j := \gcd({}^{l}/_{g}, c_j)$;

f) update the second product, setting $g := g \cdot g_j$;

g) compute the next L-eliminant setting $e_j := \frac{c_j}{g_j}$;

h) increment the index j;

4. *return the sequences $(f_0, f_1, \ldots, f_j)$ and $(e_1, \ldots, e_j)$.*

Example 82. Take the following two polynomials with integer coefficients:

$$f_0 := 3x^5 - 1, \qquad f_1 := x^4 + x^3 + x.$$

The successive pseudo-divisions yield:

$$\begin{aligned}
1^2 \cdot (3x^5 - 1) &= (3x - 3) \cdot (x^4 + x^3 + x) + 1 \cdot (3x^3 - 3x^2 + 3x - 1) \\
3^2 \cdot (x^4 + x^3 + x) &= (3x + 6) \cdot (3x^3 - 3x^2 + 3x - 1) + 3 \cdot (3x^2 - 2x + 2) \\
3^2 \cdot (3x^3 - 3x^2 + 3x - 1) &= (9x - 3) \cdot (3x^2 - 2x + 2) + 3 \cdot (x - 1) \\
1^2 \cdot (3x^2 - 2x + 2) &= (3x + 1) \cdot (x - 1) + 3 \cdot 1 \\
1^2 \cdot (x - 1) &= (x - 1) \cdot 1 + 0.
\end{aligned}$$

Hence, the primitive pseudo-remainder sequence is

$$\left(3x^5 - 1, \quad x^4 + x^3 + x, \quad 3x^3 - 3x^2 + 3x - 1, \quad 3x^2 - 2x + 2, \quad x - 1, \quad 1\right)$$

and the successive element of the first sequence equal:

$$\begin{aligned}
g_1 &= \gcd(1^2, 1) = 1 \\
g_2 &= \gcd\left(\frac{1^2 \cdot 3^2}{1}, 3\right) = 3 \\
g_3 &= \gcd\left(\frac{1^2 \cdot 3^2 \cdot 3^2}{1 \cdot 3}, 3\right) = 3 \\
g_4 &= \gcd\left(\frac{1^2 \cdot 3^2 \cdot 3^2 \cdot 1^2}{1 \cdot 3 \cdot 3}, 3\right) = 3 \\
g_5 &= \gcd\left(\frac{1^2 \cdot 3^2 \cdot 3^2 \cdot 1^2 \cdot 1^2}{1 \cdot 3 \cdot 3 \cdot 3}\right) = 1.
\end{aligned}$$

It follows that, the L-elimination sequence consists of:

$$e_1 = \frac{1}{1} = 1, \quad e_2 = \frac{3}{3} = 1, \quad e_3 = \frac{3}{3} = 1, \quad e_4 = \frac{3}{3} = 1, \quad e_5 = \frac{0}{1} = 0.$$

Example 83. A more typical example would be the following one. Let

$$f_0 = xy^3 + 10x^2y - y^3 - y = (x-1)y^3 + \left(10x^2 - 1\right)y,$$
$$f_1 = x^2y^2 + 4xy^2 - 16x^2 - 8y^2 - 9x - 1 = \left(x^2 + 4x - 8\right)y^2 - 16x^2 - 9x - 1.$$

Compute the pseudo-divisions to obtain the remaining elements of pprs:

$$f_2 = 10y, \qquad f_3 = -1600, \qquad f_4 = 0$$

The corresponding Labatie's elimination sequence is now

$$e_1 = x^4 + \frac{28}{5}x^3 - \frac{44}{5}x^2 - \frac{6}{5}x + \frac{7}{10}, \qquad e_2 = x^2 + \frac{9}{16}x + \frac{1}{16}, \qquad e_3 = 1, \qquad e_4 = 1.$$

We may now present an elimination theorem for a Labatie's elimination sequence. The reader may wish to compare it with other elimination theorems presented in this book (8.3.1,).

el. th. for Kalkrener, el. th. for Groebner, el. th. for border base

Theorem 8.1.4 (Elimination theorem). *Let K be a field. Denote $R := K[x_1, \ldots, x_n]$ and let $f, g \in R[y]$ be two relatively prime polynomials. Assume that f, g are primitive (with respect to the variable y) and $\deg_y f \geq \deg_y g$. Finally let $(f_0 = f, f_1 = g, f_2, \ldots, f_k, f_{k+1} = 0)$ be the primitive pseudo-remainder sequence for f, g and $(e_1, \ldots, e_{k-1}, e_k = 1)$ the corresponding Labatie's elimination sequence. If $\xi_1, \ldots, \xi_n, \zeta \in K^{alg}$ are such that*

$$\begin{cases} f(\xi_1, \ldots, \xi_n, \zeta) = 0 \\ g(\xi_1, \ldots, \xi_n, \zeta) = 0, \end{cases}$$

then there is $j \in \{1, \ldots, k\}$ for which

$$\begin{cases} f_j(\xi_1, \ldots, \xi_n, \zeta) = 0 \\ e_j(\xi_1, \ldots, \xi_n) = 0. \end{cases}$$

The proof of this theorem will be preceded be a series of lemmas. Keep the assumptions and notations as in the theorem. We need to introduce two auxiliary sequences of polynomials $\alpha_0, \ldots, \alpha_k$ and $\beta_0, \ldots, \beta_k$. The use of Greek letters to denote polynomials (perturbing from the notational convention of this book) serves two purposes. First we want to point out similarity between these sequences and Bézout's coefficients. Secondly, this will make it more evident that these two sequences are only auxiliary, needed for the proof of the theorem, but of little independent interest. The definition in both cases is recursive. Set $\alpha_0 := 0$, next $\alpha_1 := {}^{l_1^{d_1}}/_{g_1}$ and for $j > 1$ let

$$\alpha_j := \frac{\alpha_{j-1} \cdot q_j + \alpha_{j-2} \cdot l_j^{d_j} \cdot e_{j-1}}{g_j}.$$

Likewise, set $\beta_0 := 1$, $\beta_1 := {}^{q_1}/_{g_1}$ and

$$\beta_j := \frac{\beta_{j-1} \cdot q_j + \beta_{j-2} \cdot l_j^{d_j} \cdot e_{j-1}}{g_j}.$$

Directly by definition, we know only that α's and β's are rational functions. We need to prove that they are actually polynomials and that they additionally satisfy certain identities. Let us begin with the first sequence.

Lemma 8.1.5. *For every $j \in \{1, \dots, k\}$, the element α_j is a polynomial and the following identity holds:*

$$\frac{l_1^{d_1} \cdots l_j^{d_j}}{g_1 \cdots g_j} \cdot f_1 = \alpha_j \cdot f_j + \alpha_{j-1} \cdot f_{j+1} \cdot e_j. \tag{8.4}$$

Proof. We prove the lemma by induction on j, starting at $j = 1$. Since $\alpha_0 = 0$ and $\alpha_1 = {}^{l_1^{d_1}}/_{g_1}$, we trivially have

$$\frac{l_1^{d_1}}{g_1} \cdot f_1 = \frac{l_1^{d_1}}{g_1} \cdot f_1 + 0 = \alpha_1 \cdot f_1 + \alpha_0 \cdot f_2 \cdot e_1.$$

This shows that Eq (8.4) is true for $j = 1$.

Take now any $j \geq 1$ and assume that Eq. (8.4) holds for $j-1$. That means that we have an identity

$$\frac{l_1^{d_1} \cdots l_{j-1}^{d_{j-1}}}{g_1 \cdots g_{j-1}} \cdot f_1 = \alpha_{j-1} \cdot f_{j-1} + \alpha_{j-2} \cdot f_j \cdot e_{j-1}.$$

Multiply both sides by $l_j^{d_j}$:

$$\frac{l_1^{d_1} \cdots l_{j-1}^{d_{j-1}} \cdot l_j^{d_j}}{g_1 \cdots g_{j-1}} \cdot f_1 = \alpha_{j-1} \cdot \left(l_j^{d_j} \cdot f_{j-1}\right) + \alpha_{j-2} \cdot l_j^{d_j} \cdot f_j \cdot e_{j-1}.$$

Use Eq. (8.3) to substitute $q_j f_j + c_j f_{j+1}$ for the term in parenthesis:

$$\begin{aligned}\frac{l_1^{d_1} \cdots l_j^{d_j}}{g_1 \cdots g_{j-1}} \cdot f_1 &= \alpha_{j-1} \cdot \left(q_j \cdot f_j + c_j \cdot f_{j+1}\right) + \alpha_{j-2} \cdot l_j^{d_j} \cdot f_j \cdot e_{j-1} \\ &= \left(\alpha_{j-1} \cdot q_j + \alpha_{j-2} \cdot l_j^{d_j} \cdot e_{j-1}\right) \cdot f_j + \alpha_{j-1} \cdot c_j \cdot f_{j+1}.\end{aligned}$$

Next, recall that $g_j = \gcd\left(\frac{l_1^{d_1} \cdots l_j^{d_j}}{g_1 \cdots g_{j-1}}, c_j\right)$. Therefore, g_j divides two out of three terms in the above equation. Hence it must divide also the remaining one. The polynomial f_j is primitive and g_j is an element of the ring R. Therefore g_j divides the expression in parenthesis. It follows that

$$\alpha_j = \frac{\alpha_{j-1} q_j + \alpha_{j-2} l_j^{d_j} e_{j-1}}{g_j}$$

is a polynomial and

$$\frac{l_1^{d_1}\cdots l_j^{d_j}}{g_1\cdots g_{j-1}\cdot g_j}\cdot f_1 = \left(\frac{\alpha_{j-1}q_j+\alpha_{j-2}l_j^{d_j}e_{j-1}}{g_j}\right)\cdot f_j + \alpha_{j-1}\cdot\frac{c_j}{g_j}\cdot f_{j+1}$$
$$= \alpha_j\cdot f_j + \alpha_{j-1}\cdot e_j\cdot f_{j+1}. \qquad \square$$

Lemma 8.1.6. *For every $j\in\{1,\dots,k\}$, the element β_j is a polynomial and the following identity holds:*

$$\frac{l_1^{d_1}\cdots l_j^{d_j}}{g_1\cdots g_j}\cdot f_0 = \beta_j\cdot f_j + \beta_{j-1}\cdot f_{j+1}\cdot e_j. \tag{8.5}$$

Proof. The proof of this lemma closely mimics the proof of the previous lemma. In fact, the only difference is for $j=1$. The polynomials $f_0, f_1, \dots, f_k$ form pprs. Look at the first line of Eq. (8.3):

$$l_1^{d_1}\cdot f_0 = q_1\cdot f_1 + c_1\cdot f_2.$$

By definition, $g_1\in R$ is the greatest common divisor of $l_1^{d_1}$ and c_1, hence it divides two out of three terms of the above equality. Consequently it must divide also the remaining term q_1f_1. Now, the polynomial f_1 is primitive, therefore g_1 divides q_1 and so $\beta_1 = {}^{q_1}/_{g_1}$ is a polynomial and we have an identity

$$\frac{l_1^{d_1}}{g_1}\cdot f_0 = \frac{q_1}{g_1}\cdot f_1 + 1\cdot\frac{c_1}{g_1}\cdot f_2 = \beta_1\cdot f_1 + \beta_0\cdot e_1\cdot f_2.$$

This proves the lemma for $j=1$. The rest of the proof, namely the inductive step, is identical to the previous one. Thus we take the liberty of omitting it. $\square$

We can prove now the following result, which somehow reassembles Bézout's identity (this is why we decided to use letters α and β for elements of the two sequences).

Lemma 8.1.7. *For every $j\in\{1,\dots,k\}$, the following identity holds:*

$$(-1)^j\cdot e_1\cdots e_{j-1}\cdot f_j = \alpha_{j-1}\cdot f_0 - \beta_{j-1}\cdot f_1.$$

Proof. Using the previous two lemmas we may write

$$\begin{cases} \frac{l_1^{d_1}\cdots l_j^{d_j}}{g_1\cdots g_j}\cdot f_1 = \alpha_j f_j + \alpha_{j-1}f_{j+1}e_j \\ \frac{l_1^{d_1}\cdots l_j^{d_j}}{g_1\cdots g_j}\cdot f_0 = \beta_j f_j + \beta_{j-1}f_{j+1}e_j. \end{cases}$$

Eliminate f_{j+1} form this system, by subtracting α_{j-1} times the second equation from β_{j-1} times the first equation:

$$\frac{l_1^{d_1}\cdots l_j^{d_j}}{g_1\cdots g_j}\cdot(\beta_{j-1}f_1 - \alpha_{j-1}f_0) = (\alpha_j\beta_{j-1} - \alpha_{j-1}\beta_j)\cdot f_j. \tag{8.6}$$

Next, eliminate f_j taking the difference: β_j times the first equation minus α_j times the second one. This yields:

$$\frac{l_1^{d_1}\cdots l_j^{d_j}}{g_1\cdots g_j}\cdot(\beta_j f_1-\alpha_j f_0)=(\alpha_{j-1}\beta_j-\alpha_j\beta_{j-1})\cdot f_{j+1}\cdot e_j.$$

This holds for every j, so we change the indexing substituting $j-1$ for j:

$$\frac{l_1^{d_1}\cdots l_{j-1}^{d_{j-1}}}{g_1\cdots g_{j-1}}\cdot(\beta_{j-1}f_1-\alpha_{j-1}f_0)=-(\alpha_{j-1}\beta_{j-2}-\alpha_{j-2}\beta_{j-1})\cdot f_j\cdot e_{j-1}.$$

Finally, multiply both sides by ${l_j^{d_j}}/{g_j}$:

$$\frac{l_1^{d_1}\cdots l_j^{d_j}}{g_1\cdots g_j}\cdot(\beta_{j-1}f_1-\alpha_{j-1}f_0)=-(\alpha_{j-1}\beta_{j-2}-\alpha_{j-2}\beta_{j-1})\cdot f_j\cdot e_{j-1}\cdot\frac{l_j^{d_j}}{g_j}. \tag{8.7}$$

The left-hand-sides of Eq. (8.6) and (8.7) are the same so must be their right-hand-sides. Thus

$$(\alpha_j\beta_{j-1}-\alpha_{j-1}\beta_j)\cdot f_j=-(\alpha_{j-1}\beta_{j-2}-\alpha_{j-2}\beta_{j-1})\cdot e_{j-1}\cdot\frac{l_j^{d_j}}{g_j}\cdot f_j.$$

Cancel f_j which is nonzero for $j\le k$ and denote $\delta_j:=\alpha_j\beta_{j-1}-\alpha_{j-1}\beta_j$. The above equation provides a recursive formula for computing δ_j:

$$\begin{aligned}\delta_j&=(-1)\cdot\delta_{j-1}\cdot e_{j-1}\cdot\frac{l_j^{d_j}}{g_j}\\&=(-1)^2\cdot\delta_{j-2}\cdot e_{j-2}e_{j-1}\cdot\frac{l_{j-1}^{d_{j-1}}l_j^{d_j}}{g_{j-1}g_j}\\&\;\;\vdots\\&=(-1)^{j-1}\cdot\delta_1\cdot e_1\cdots e_{j-1}\cdot\frac{l_2^{d_2}\cdots l_j^{d_j}}{g_2\cdots g_j}.\end{aligned}$$

By definition, $\delta_1=\alpha_1\beta_0-\alpha_0\beta_1={l_1^{d_1}}/{g_1}\cdot 1-0\cdot{q_1}/{g_1}={l_1^{d_1}}/{g_1}$. Together with the previous formula this yields

$$\delta_j=(-1)^{j-1}\cdot e_1\cdots e_{j-1}\cdot\frac{l_1^{d_1}\cdots l_j^{d_j}}{g_1\cdots g_j}.$$

Substitute it, in place of $\alpha_j\beta_{j-1}-\alpha_{j-1}\beta_j$, into Eq. (8.6) to get

$$\frac{l_1^{d_1}\cdots l_j^{d_j}}{g_1\cdots g_j}\cdot(\beta_{j-1}f_1-\alpha_{j-1}f_0)=(-1)^{j-1}\cdot e_1\cdots e_{j-1}\cdot\frac{l_1^{d_1}\cdots l_j^{d_j}}{g_1\cdots g_j}\cdot f_j.$$

Cancel the common nonzero term on both sides to obtain the formula we are after. □

We are now in a position to prove the elimination theorem.

Proof of Theorem 8.1.4. Suppose that $\xi_1, \ldots, \xi_n, \zeta$ is a zero of the system of two polynomial equations $f_0 = f_1 = 0$. The previous lemma asserts that for every $j \leq k$

$$\big(e_1 \cdots e_{j-1} \cdot f_j\big)(\xi_1, \ldots, \xi_n, \zeta) = 0.$$

In particular, taking $j = k$, we have

$$\big(e_1 \cdots e_{k-1}\big)(\xi_1, \ldots, \xi_n, \zeta) \cdot f_k(\xi_1, \ldots, \xi_n, \zeta) = 0,$$

but we know that f_k, being the greatest common divisor of two relatively prime polynomials $f_0 = f$ and $f_1 = g$, is a unit. This implies that $f_k(\xi_1, \ldots, \xi_n, \zeta) \neq 0$. It follows that some e_j vanishes at $\xi_1, \ldots, \xi_n, \zeta$. However, Labatie's eliminants $e_1, \ldots, e_{k-1}$ are elements of $R = K[x_1, \ldots, x_n]$. They do not depend on y. This proves that there is $j < k$ such that $e_j(\xi_1, \ldots, \xi_n) = 0$. Take j to be the smallest such index. If $j = 1$, then trivially

$$\begin{cases} f_1(\xi_1, \ldots, \xi_n, \zeta) = 0 \\ e_1(\xi_1, \ldots, \xi_n) = 0. \end{cases}$$

and the thesis is proved. On the other hand, if $j > 1$, then by Lemma 8.1.7 we have

$$(e_1 \cdots e_{j-1})(\xi_1, \ldots, \xi_n) \cdot f_j(\xi_1, \ldots, \xi_n, \zeta) = 0.$$

The first term cannot be zero, since j was taken to be the smallest index for which e_j vanishes at $\xi_1, \ldots, \xi_n$. Consequently the second term must be zero and we have

$$\begin{cases} f_j(\xi_1, \ldots, \xi_n, \zeta) = 0 \\ e_j(\xi_1, \ldots, \xi_n) = 0, \end{cases}$$

as claimed. □

Example 84. We shall now see how to employ the elimination theorem for solving systems of polynomial equations. Consider the following system of polynomial equations :

$$\begin{cases} xy^3 + 10x^2y - y^3 - y = 0 \\ x^2y^2 + 4xy^2 - 16x^2 - 8y^2 - 9x - 1 = 0 \end{cases} \tag{8.8}$$

These are the same two polynomials that we encountered in the previous example, so we already know the primitive pseudo-remainder sequence:

$$f_0 = (x-1)\,y^3 + \big(10x^2 - 1\big)\,y, \quad f_1 = \big(x^2 + 4x - 8\big)\,y^2 - 16x^2 - 9x - 1, \quad f_2 = 10y, \quad f_3 = -1600, \quad f_4$$

and the L-eliminats:

$$e_1 = x^4 + \frac{28}{5}x^3 - \frac{44}{5}x^2 - \frac{6}{5}x + \frac{7}{10}, \qquad e_2 = x^2 + \frac{9}{16}x + \frac{1}{16}, \qquad e_3 = 1, \qquad e_4 = 1.$$

The elimination theorem says that the solutions of system Eq. (8.8) can be obtained from triangular systems:

$$\begin{cases} \left(x^2+4x-8\right)y^2-16x^2-9x-1=0 \\ x^4+\frac{28}{5}x^3-\frac{44}{5}x^2-\frac{6}{5}x+\frac{7}{10}=0 \end{cases} \qquad \begin{cases} 10y=0 \\ x^2+\frac{9}{16}x+\frac{1}{16}=0 \end{cases} \qquad \begin{cases} -1600=0 \\ 1=0. \end{cases}$$

We may skip the last one as it has no solutions. Solve the other two. The roots of e_1 are approximately:

$$\{-6.86,-0.322,0.237,1.34\}.$$

Substitute them one-by-one into f_1 and solve the resulting equations. For example, taking $x\approx -6.86$, we have to find the roots of $f_1(-6.86,y)\approx 11.6y^2-691.$ and those are: $\{-7.73,7.73\}$. In similar fashion we get all the other solutions:

- for $x\approx -0.322$, the roots are $-0.161, 0.161$,
- for $x\approx 0.237$, the roots are $-0.759i, 0.759i$,
- for $x\approx 1.34$, the roots are $-7.06i, 7.06i$.

Analogously we find the roots of e_2: $\{-0.410,-0.152\}$. The corresponding polynomial $f_2=10y$ has clearly only one root, namely 0.

This way we got a total of 10 possible pairs. The elimination theorem provides only a necessary condition, hence formally we should now (numerically) check if all these pairs are actually zeros of our system. But we will see soon (cf.) that for bivariate polynomials this is always true. Thus we have solved the system,

missing

The elimination theorem provides a necessary condition for a point $P=(\xi_1,\dots,\xi_n,\zeta)$ to be a zero of a system of polynomial equations—the point $(\xi_1,\dots,\xi_n)$ obtained by eliminating the last coordinate must be a root of an eliminant. This necessary condition has its counterpart in form of an extension theorem, presenting a sufficient condition for a root of an eliminant to be extendable to a zero of the original system. This is a rather typical situation, we shall observe similar pairs consisting of an elimination and extension theorems also for resultants (see Theorems 8.3.1 and 8.3.2) and Gröbner bases (Theorems 9.8.3 and).

missing

Theorem 8.1.8 (Extension theorem)**.** *Use the same notation and assumptions as in Theorem 8.1.4. If there are $\xi_1,\dots,\xi_n,\zeta\in K^{alg}$ such that for some $j\in\{1,\dots,k-1\}$*

$$\begin{cases} f_j(\xi_1,\dots,\xi_n,\zeta)=0 \\ e_j(\xi_1,\dots,\xi_n)=0, \end{cases}$$

then also

$$\begin{cases} f(\xi_1,\dots,\xi_n,\zeta)=0 \\ g(\xi_1,\dots,\xi_n,\zeta)=0. \end{cases}$$

The proof of this theorem would take us too far afield, hence we take the liberty to omit it referring the reader instead to [30, Theorem 4] in combination with [37, Theorem 6 on page 62]. The situation is much simpler when we have only two variables to deal with. In that case the condition in Theorem 8.1.4 is both necessary and sufficient.

Theorem 8.1.9. *Let $R = K[x]$ be a ring of univariate polynomials over a field and $f, g \in R[y]$ be two relatively prime primitive polynomials such that $\deg_y f \geq \deg_y g$. Further let $(f_0 = f, f_1 = g, f_2, \ldots, f_k, f_{k+1} = 0)$ be the primitive pseudo-remainder sequence and $e_1, \ldots, e_{k-1}, e_k = 1$ the corresponding Labatie's eliminants. If a point $P = (\xi, \zeta)$ with $\xi, \zeta \in K^{alg}$ is a zero of a system*

$$\begin{cases} f_j(x,y) = 0 \\ e_j(x,y) = 0 \end{cases}$$

for some $j \in \{1, \ldots, k-1\}$ then it is also a zero of the original system

$$\begin{cases} f(x,y) = 0 \\ g(x,y) = 0. \end{cases}$$

Proof. We have an identity (see Eq. (8.5)):

$$\frac{l_1^{d_1} \cdots l_j^{d_j}}{g_1 \cdots g_j} \cdot f_0 = \beta_j \cdot f_j + \beta_{j-1} \cdot f_{j+1} \cdot e_j.$$

Evaluate both sides at (ξ, ζ) to get

$$\left(\frac{l_1^{d_1} \cdots l_j^{d_j}}{g_1 \cdots g_j}\right)(\xi) \cdot f_0(\xi, \zeta) = 0.$$

The L-eliminant e_j was constructed in such a way that it is relatively prime to $\frac{l_1^{d_1} \cdots l_j^{d_j}}{g_1 \cdots g_j}$. Thus, ξ cannot be a root of the latter polynomial, so it follows that $f_0(\xi, \zeta) = 0$. Similar argument shows also $f_1(\xi, \zeta) = 0$, here one uses Eq. (8.4) in place of Eq. (8.5). □

As stated in the introduction to this section, the discussed method of solving systems of polynomial equations was independently discovered by A.-G. Labatie and M. Kalkbrener. The two authors proosed a slightly different definitions of eliminats, though. Kalkbrener's eliminants, that we are going to introduce next, are generally smaller than Labatie's ones (see Proposition 8.1.13). Our first step is to derive an alternative formula for computing Labatie's eliminants. To this end we define an auxiliary sequence $h_1, \ldots, h_k$ putting:

$$h_j := \frac{\prod_{i \leq j} c_i}{\gcd\left(\prod_{i \leq j} c_i, \prod_{i \leq j} l_i^{d_i}\right)}, \qquad \text{for } j \in \{1, \ldots, k\}.$$

Proposition 8.1.10. *Let $f_0, f_1 \in R[x]$ be two polynomials and $\mathscr{L} = (e_1, \ldots, e_{k-1})$ be their L-elimination sequence. Then for every $j \in \{1, \ldots, k-1\}$:*

$$e_j = \frac{h_j}{\gcd\left(h_j, \prod_{i \leq j-1} e_i\right)}. \tag{8.9}$$

Proof. Start from the right-hand-side of the above formula

$$\frac{h_j}{\gcd\Big(h_j, \prod\limits_{i\le j-1} e_i\Big)} = \frac{\prod\limits_{i\le j} c_i}{\gcd\Big(\prod\limits_{i\le j} c_i, \prod\limits_{i\le j} l_i^{d_i}\Big)\cdot \gcd\Big(\frac{\prod_{i\le j} c_i}{\gcd\Big(\prod_{i\le j} c_i, \prod_{i\le j} l_i^{d_i}\Big)}, \prod\limits_{i\le j-1} e_i\Big)}$$

Recall that the following identity holds:

$$\gcd(\alpha,\beta)\cdot\gcd\Big(\frac{\alpha}{\gcd(\alpha,\beta)},\gamma\Big) = \gcd\big(\alpha, \gcd(\alpha,\beta)\cdot\gamma\big).$$

Apply it to the previous formula to get

$$\begin{aligned}
&= \frac{\prod\limits_{i\le j} c_i}{\gcd\Big(\prod\limits_{i\le j} c_i,\ \gcd\Big(\prod\limits_{i\le j} c_i, \prod\limits_{i\le j} l_i^{d_i}\Big)\cdot \prod\limits_{i\le j-1} e_i\Big)} \\
&= \frac{\prod\limits_{i\le j} c_i}{\gcd\Big(\prod\limits_{i\le j} c_i,\ \gcd\Big(\prod\limits_{i\le j} c_i, \prod\limits_{i\le j} l_i^{d_i}\Big)\cdot \prod\limits_{i\le j-1} \frac{c_i}{g_i}\Big)} \\
&= \frac{c_j\cdot \prod\limits_{i\le j-1} c_i}{\gcd\Big(c_j\cdot \prod\limits_{i\le j-1} c_i,\ \gcd\Big(c_j\cdot \prod\limits_{i\le j-1} \frac{c_i}{g_i},\ l_j^{d_j}\cdot \prod\limits_{i\le j-1} \frac{l_i^{d_i}}{g_i}\Big)\cdot \prod\limits_{i\le j-1} c_i\Big)} \\
&= \frac{c_j}{\gcd\Big(c_j,\ \gcd\Big(c_j\cdot \prod\limits_{i\le j-1} \frac{c_i}{g_i},\ l_j^{d_j}\cdot \prod\limits_{i\le j-1} \frac{l_i^{d_i}}{g_i}\Big)\Big)}
\end{aligned}$$

Now, the identity $\gcd\big(\alpha,\gcd(\alpha\beta,\gamma)\big) = \gcd(\alpha,\gamma)$ yields

$$= \frac{c_j}{\gcd\Big(c_j,\ l_j^{d_j}\cdot \prod\limits_{i\le j-1} \frac{l_i^{d_i}}{g_i}\Big)} = \frac{c_j}{g_j} = e_j \qquad \square$$

Recall that in Section 5.3 we introduced the notion of square-free polynomials and radicals (of polynomials). This notion can be generalized to an arbitrary unique factorization domain. We shall say that an element a of a unique factorization domain R is *square-free* if it is not divisible by a square of any non-unit. If $a \in R$ is any non-invertible element, then it can be uniquely factorized into a product of primes (this is what being a u.f.d. means):

$$a = u\cdot p_1^{k_1}\cdots p_m^{k_m},$$

where $p_1, \dots, p_m \in R$ are prime (irreducible) elements of R, $u \in R$ is invertible and $k_1, \dots, k_m \in \mathbb{N}$. The element $u \cdot p_1 \cdots p_m$ is then called the *radical* of a and denoted $\operatorname{rad} a$ (see also a remark on page 220).

Remark 61. While it is very easy to compute the radical of a polynomial (cf. Observation 5.3.4), it is no longer so for an element of an arbitrary unique factorization domain. In fact, the author of these notes is not aware of any method of finding the radical of an integer that would be significantly simpler than the full-fledged factorization.

Using a formula very similar to the one in the preceding proposition, with h_j replaced by its radical, we define Kelkbrener's elimination sequence.

Definition 8.1.11. Under the above notation, the sequence $\mathscr{K} := (\hat{e}_1, \dots, \hat{e}_{k-1})$, where

$$\hat{e}_j := \frac{\operatorname{rad} h_j}{\gcd\Big(\operatorname{rad} h_j, \prod_{i \le j-1} \hat{e}_i\Big)} \tag{8.10}$$

is called the *Kalkbrener's elimination sequence* of f_0 and f_1 (or shortly *K-elimination sequence*) and ist elements are called *Kalkbrener's eliminants* (or *K-eliminants*).

Remark 62. If $j = 1$, then the products in Eq. (8.9) and (8.10) run over empty sets of indices, hence they equal 1 by definition. Thus, the formulas read as

$$e_1 = h_1 \qquad \text{and} \qquad \hat{e}_1 = \operatorname{rad} h_1.$$

An algorithm computing K-elimination sequence is again a very simple modification of Algorithm 1.3.16:

Algorithm 8.1.12. *Let R be a unique factorization domain in which* gcd *and* rad *are computable. Given two polynomials f_0, f_1 with coefficients in R this algorithm returns their primitive pseudo-remainder sequence together with the Kalkbrener's elimination sequence.*

1. *make sure that the degree of f_1 does not exceed the degree of f_0, if it is not the case swap the two polynomials;*
2. *initialize $j := 1$, $l := 1$, $c := 1$ and $e := 1$;*
3. *as long as f_j is non-zero proceed as follows:*
   a) *compute the pseudo-quotient q_j and pseudo-remainder r_j of $f_{j-1} \bmod f_j$;*
   b) *split the pseudo-remainder into its primitive part $f_{j+1} := \operatorname{pp}(r_j)$ and its content $c_j := \operatorname{cont}(r_j)$;*
   c) *let l_j be the leading coefficient of f_j and $d_j := \deg f_{j-1} - \deg f_j + 1$;*
   d) *update the products $l := l \cdot l_j^{d_j}$ and $c := c \cdot c_j$;*
   e) *set $h_j := {}^{c}/_{\gcd(c,l)}$ and compute the radical of h_j;*
   f) *compute the next K-eliminant setting $\hat{e}_j := \frac{\operatorname{rad} h_j}{\gcd(\operatorname{rad} h_j, e)}$;*
   g) *update e setting $e := e \cdot \hat{e}_j$;*

h) increment the index j;

4. *return the sequences* $(f_0, f_1, \ldots, f_j)$ *and* $(\hat{e}_1, \ldots, \hat{e}_{j-1})$.

Remark 63. In view of Proposition 8.1.10, omitting the computation of the radical in the above algorithm one would obtain Labatie's elimination sequence.

In general, K-elimination sequence does not have to be the same as L-elimination sequence, but there is a clear link between them as indicated by similarity between Eq. (8.9) and Eq. (8.10).

Proposition 8.1.13. *The terms of Kalkbrener's elimination sequence divide radicals of corresponding terms of Labatie's elimination sequence.*

Proof. The assertion holds for $\hat{e}_1$ and e_1, since the former is clearly the radical of the latter. Fix an index $j < k$ and assume the thesis to be true for all preceding indices. Take any prime element p of R dividing $\hat{e}_j$. We will show that it divides the radical of e_j. By definition, $\hat{e}_j = {}^{\operatorname{rad} h_j}/_{\gcd(\operatorname{rad} h_j, \hat{e}_1 \cdots \hat{e}_{j-1})}$, hence p divides $\operatorname{rad} h_j$ (thus also h_j) but not $\hat{e}_1 \cdots \hat{e}_{j-1}$. Since p is prime, it follows from the inductive hypothesis that p does not divide $e_1 \cdots e_{j-1}$. Consequently, p divides ${}^{h_j}/_{\gcd(h_1, e_1 \cdots e_{j-1})}$, but this is just e_j by Proposition 8.1.10. □

We are now ready to show how elimination sequences can be used for solving systems of polynomial equations. We begin with a case of two bivariate equations, where we have the following:

Theorem 8.1.14. *Let* $f_0, f_1 \in K[x, y]$ *be two bivariate polynomials over a field* K*. Assume that they are relatively prime and primitive, when treated as polynomials in* y *with coefficients in* $R = K[x]$*. Further, let* $e_1, \ldots, e_{k-1}$ *be either a Labatie's or Kalkbrener's elimination sequence for* f_0*,* f_1*. Then the set of the solutions to the system*

$$\begin{cases} f_0(x, y) = 0 \\ f_1(x, y) = 0 \end{cases}$$

is the union of sets of the solutions to (triangular) systems

$$\begin{cases} f_{i+1}(x, y) = 0 \\ e_i(x) = 0, \end{cases} \qquad \text{for} \quad i = 1, \ldots, k-1.$$

Unfinished section

8.2 Resultant

In this section we introduce one of fundamental tools of elimination theory, namely a *resultant* of two polynomials. Let us begin with a simple, nevertheless, motivating example. Take two quadratic polynomials $f = f_0 + f_1 x + f_2 x^2$ and $g = g_0 + g_1 x + g_2 x^2$.

The field of constants is not important at this moment, we can just take $\mathbb{R}$ for simplicity. We wonder whether these two polynomials have a common root in $\mathbb{C}$ (or in an algebraic closure of the field of constants of our choice). Of course, we may explicitly compute the roots using the formula from Section 6.2. However, we want to avoid doing this. We hope to find a criterion which does not involve actual roots, but instead is simpler and easily generalizable to arbitrary degree polynomials. For a moment, pretend that different powers of x are "independent variables":

$$x_0 := x^0, \quad x_1 := x^1, \quad x_2 := x^2.$$

By this move, we can consider the problem on grounds of linear algebra. We are looking for a nontrivial solution to a system of two linear equations

$$\begin{cases} f_0 x_0 + f_1 x_1 + f_2 x_2 = 0 \\ g_0 x_0 + g_1 x_1 + g_2 x_2 = 0. \end{cases}$$

This system has three "variables", but consists of only two equations. Anyway, introducing only one new "variable", we obtain two new equations. Consistently with our prior notation, let $x_4 := x^4$. Since $x \cdot f = 0 = x \cdot g$, we write

$$\begin{cases} f_0 x_0 + f_1 x_1 + f_2 x_2 = 0 \\ f_0 x_1 + f_1 x_2 + f_2 x_3 = 0 \\ g_0 x_0 + g_1 x_1 + g_2 x_2 = 0 \\ g_0 x_1 + g_1 x_2 + g_2 x_3 = 0. \end{cases}$$

For polynomials f and g to have a common root ξ, this system must have a nontrivial solution $x_0 = \xi^0 = 1$, $x_1 = \xi$, $x_2 = \xi^2$ and $x_3 = \xi^3$. On the other hand, being homogeneous, this system always has a trivial solution. It follows from Cramer's rule that the determinant of the matrix of coefficients must vanish. All in all, we have just proved that the necessary condition for two quadratic polynomials $f = f_0 + f_1 x + f_2 x^2$ and $g = g_0 + g_1 x + g_2 x^2$ to have a common roots is

$$\det \begin{pmatrix} f_0 & f_1 & f_2 & 0 \\ 0 & f_0 & f_1 & f_2 \\ g_0 & g_1 & g_2 & 0 \\ 0 & g_0 & g_1 & g_2 \end{pmatrix} = 0.$$

Later we will discover that this condition is also sufficient.

Let R be an integral domain. Denote its field of fractions by K and let $L \supset K$ be some algebraically closed field.

Definition 8.2.1. Let $f = f_0 + f_1 x + \cdots + f_m x^m$ and $g = g_0 + g_1 x + \cdots + g_n x^n$ be two nonzero polynomials with coefficients in R. Assume that they factor over L into linear terms

$$f = \operatorname{lc}(f) \cdot (x - \xi_1) \cdots (x - \xi_m), \qquad g = \operatorname{lc}(g) \cdot (x - \zeta_1) \cdots (x - \zeta_n),$$

where $\xi_1, \ldots, \xi_m, \zeta_1, \ldots, \zeta_n \in L$ are all the roots respectively of f and g (multiple roots being repeated). The *resultant* of f and g is defined as

$$\operatorname{res}_x(f,g) := \operatorname{lc}(f)^{\deg g} \cdot \operatorname{lc}(g)^{\deg f} \cdot \prod_{\substack{1\le i\le m\\ 1\le j\le n}} (\xi_i - \zeta_j). \tag{8.11}$$

For the sake of completeness, for every $f, g \in R[x]$ we define also

$$\operatorname{res}_x(0,g) = \operatorname{res}_x(f,0) = 0.$$

Observe that, if $f = c$ is a constant polynomial, then the above product runs over an empty set of indices and so equals 1, by definition. In particular we have:

Observation 8.2.2. *If $f = c$ is a constant polynomial, then $\operatorname{res}_x(c,g) = c^{\deg g}$. Analogously, if $g = c$ is a constant polynomial, then $\operatorname{res}_x(f,c) = c^{\deg f}$.*

Example 85. Take the following two polynomials with rational coefficients (we will use the same pair of polynomials for a number of examples that follow):

$$f = x^3 - x^2 + 4x - 4 = (x-1)\cdot(x-2I)\cdot(x+2I)$$
$$g = x^3 + x^2 - 16x - 16 = (x-4)\cdot(x+1)\cdot(x+4).$$

The resultant of f and g computed directly from the definition is

$$\operatorname{res}_x(f,g) = (1-4)(1+1)(1+4)(2I-4)(2I+1)(2I+4)(-2I-4)(-2I+1)(-2I+4) = -60000.$$

It is easy to derive an alternative formula for the resultant.

Proposition 8.2.3. *Let $f, g \in R[x]$ be two nonzero polynomials and $\xi_1, \ldots, \xi_m$ be all the roots of f (multiple roots being repeated). Then*

$$\operatorname{res}_x(f,g) = \operatorname{lc}(f)^{\deg g} \cdot \prod_{i=1}^{m} g(\xi_i). \tag{8.12}$$

Proof. As before, let $\zeta_1, \ldots, \zeta_n$ be all the roots of g in some algebraically closed field L, so that g factors into linear terms

$$g = \operatorname{lc}(g) \cdot \prod_{j=1}^{n} (x - \zeta_j).$$

Expanding the formula for the resultant we write

$$\begin{aligned}
\operatorname{res}_x(f,g) &= \operatorname{lc}(f)^{\deg g} \cdot \operatorname{lc}(g)^{\deg f} \cdot \prod_{i=1}^{m}\prod_{j=1}^{n} (\xi_i - \zeta_j) \\
&= \operatorname{lc}(f)^{\deg g} \cdot \prod_{i=1}^{m} \Big(\operatorname{lc}(g) \cdot \prod_{j=1}^{n} (\xi_i - \zeta_j) \Big) \\
&= \operatorname{lc}(f)^{\deg g} \cdot \prod_{i=1}^{m} g(\xi_i).
\end{aligned}$$

□

Expand the product on the right-hand-side of Eq. (8.12). Multiplication is commutative hence the result is a symmetric polynomial in ξ_i's. The fundamental theorem of symmetric polynomials (see e.g. [51, Theorem IV.6.1]), asserts that it can be expressed in terms of elementary symmetric polynomials. These in turn are elements of R, by Viète's formulæ (see Theorem 6.1.6). All in all, we have proved:

Observation 8.2.4. *If $f, g \in R[x]$, then $\operatorname{res}_x(f,g) \in R$.*

A more precise statement is the following one:

Observation 8.2.5. *Let p be the characteristic of R, then either*

$$\operatorname{res}_x(f,g) \in \mathbb{Z}[f_0,\ldots,f_m,g_0,\ldots,g_m], \qquad \text{if } p=0$$

or

$$\operatorname{res}_x(f,g) \in \mathbb{Z}_p[f_0,\ldots,f_m,g_0,\ldots,g_m], \qquad \text{if } p>0.$$

The importance of this apparently fancy formulation lays in the fact that it is a starting point for generalization of the notion of the resultant to an arbitrary number of variables. An important property of the resultant is a fact that it is closely related to the greatest common divisor. We will investigate this relation more closely later, here we just observe the following basic fact:

Proposition 8.2.6. *Let $f, g \in K[x]$ be two nonzero polynomials with coefficients in a a field K contained in some algebraically closed field L. The following conditions are equivalent:*

1. $\operatorname{res}_x(f,g) = 0$;
2. *polynomials f and g have a common root in L;*
3. *polynomials f and g have a nontrivial common factor in the ring $K[x]$.*

Proof. The equivalence of the first two conditions is trivial by the way we defined the resultant. We shall prove that 2 is equivalent to 3. Let $\xi \in L$ be a common root of f and g. Suppose a contrario that f and g are relatively prime in $K[x]$. Bézout identity (Theorem 1.3.4) asserts that there are polynomials $a, b \in K[x]$ such that

$$1 = \gcd(f,g) = a \cdot f + b \cdot g.$$

Evaluating both sides at ξ we obtain $1 = a(\xi)\cdot f(\xi) + b(\xi)\cdot g(\xi) = 0$. A contradiction.

Conversely, let $h \in K[x]$, $\deg h > 0$ be the greatest common divisor of f and g. Since h is non-constant, it has a root (denote it ξ) in L. Then ξ is a common root of both f and g. □

The resultant exhibits also a certain kind of antisymmetry.

Observation 8.2.7. *If $f, g \in R[x]$ are polynomials, then*

$$\operatorname{res}_x(g,f) = (-1)^{\deg f \cdot \deg g} \cdot \operatorname{res}_x(f,g).$$

Recall that the *reciprocal* polynomial (reverse polynomial in the power-basis representation) of a polynomial $f = f_0 + f_1 x + \cdots + f_m x^m$ is the polynomial $\operatorname{rev}(f) = f_m + f_{m-1}x + \cdots + f_0 x^m$ with the same coefficients as f but in reverse order (see Definition 1.1.1).

Proposition 8.2.8. *Let f and g be two polynomials. If the constant terms of both f and g are nonzero, then*

$$\operatorname{res}_x\big(\operatorname{rev}(f),\operatorname{rev}(g)\big) = \operatorname{res}_x(g,f).$$

Proof. By Observation 1.1.2, $\operatorname{rev}(f) = x^m \cdot f(1/x)$ and $\operatorname{rev}(g) = x^n \cdot g(1/x)$. Let $\xi_1,\dots,\xi_m$ be all the roots of f and $\zeta_1,\dots,\zeta_n$ all the roots of g, with the convention that multiple roots are repeated. It is clear that $1/\xi_1,\dots,1/\xi_m$ (respectively $1/\zeta_1,\dots,1/\zeta_n$) are all the roots of $\operatorname{rev}(f)$ (respectively $\operatorname{rev}(g)$). Therefore, by definition

$$\begin{aligned}
\operatorname{res}_x\big(\operatorname{rev}(f),\operatorname{rev}(g)\big) &= f_0^n \cdot g_0^m \cdot \prod_{\substack{1\le i\le m\\ 1\le j\le n}} \left(\frac{1}{\xi_i} - \frac{1}{\zeta_j}\right) \\
&= f_0^n \cdot g_0^m \cdot \prod_{\substack{1\le i\le m\\ 1\le j\le n}} \frac{\zeta_j - \xi_i}{\xi_i \zeta_j} \\
&= \frac{f_0^n}{\left(\prod_{i=1}^m \xi_i\right)^n} \cdot \frac{g_0^m}{\left(\prod_{j=1}^n \zeta_j\right)^n} \cdot (-1)^{nm} \cdot \prod_{\substack{1\le i\le m\\ 1\le j\le n}} (\xi_i - \zeta_j)
\end{aligned}$$

By Viète's formulæ the products in the denominators equal respectively $(-1)^m \cdot f_0/f_m$ and $(-1)^n \cdot g_0/g_n$, yielding

$$= (-1)^{nm} \cdot f_m^n \cdot g_n^m \cdot \prod_{\substack{1\le i\le m\\ 1\le j\le n}} (\xi_i - \zeta_j) = \operatorname{res}_x(g,f). \qquad \square$$

Yet another property of the resultant is its multiplicativity.

Proposition 8.2.9. *If $f, g, h \in R[x]$ are three polynomials, then*

$$\operatorname{res}_x(f,gh) = \operatorname{res}_x(f,g)\operatorname{res}_x(f,h) \qquad \textit{and} \qquad \operatorname{res}_x(fh,g) = \operatorname{res}_x(f,g)\operatorname{res}_x(h,g).$$

Proof. We prove just the first identity. The proof of the second one is fully analogous. Let $\xi_1,\dots,\xi_m$ be all the roots of f (in some algebraically closed field). Using Proposi-

tion 8.2.3 we write

$$\begin{aligned}\operatorname{res}_x(f,g) &= \operatorname{lc}(f)^{\deg(gh)} \cdot \prod_{i=1}^{m} (gh)(\xi_i) \\ &= \operatorname{lc}(f)^{\deg g + \deg h} \cdot \prod_{i=1}^{m} g(\xi_i) h(\xi_i) \\ &= \operatorname{lc}(f)^{\deg g} \cdot \prod_{i=1}^{m} g(\xi_i) \cdot \operatorname{lc}(f)^{\deg h} \cdot \prod_{i=1}^{m} h(\xi_i) \\ &= \operatorname{res}_x(f,g) \cdot \operatorname{res}_x(f,h). \end{aligned}$$

□

Combining the above proposition with Observation 8.2.2 we have:

Corollary 8.2.10. *If $f, g \in R[x]$ and $\alpha \in R$, then*

$$\operatorname{res}_x(\alpha \cdot f, g) = \alpha^{\deg g} \cdot \operatorname{res}_x(f,g) \qquad \text{and} \qquad \operatorname{res}_x(f, \alpha \cdot g) = \alpha^{\deg f} \cdot \operatorname{res}_x(f,g).$$

The advantage of using Eq. (8.11) for a definition is that we immediately see how the resultant can be used to detect common roots of two polynomials. Unfortunately, Eq. (8.11) does not provide any hints how to actually compute the resultant. Thus, our next goal is to develop a number of computationally feasible tools for this task.

Sylvester formula

The resultant possesses a number of properties normally associated with a determinant: it vanishes if any of its arguments is null and exhibits a certain form of antisymmetry (cf. Observation 8.2.7). Hence it should not be a big surprise that the resultant can be computed using a determinant. In this section we will show how to do it.

Definition 8.2.11. Let $f = f_0 + f_1 + \cdots + f_m x^m$ and $g = g_0 + g_1 x + \cdots + g_n x^n$ be two nonzero polynomials with coefficients in some integral domain R. A matrix

$$\left(\begin{array}{ccccccc} f_0 & f_1 & \cdots\cdots & f_m & & & \\ & f_0 & f_1 & \cdots\cdots & f_m & & \\ & & \ddots & \ddots & & \ddots & \\ & & & f_0 & f_1 & \cdots\cdots & f_m \\ \hdashline g_0 & g_1 & \cdots\cdots & g_n & & & \\ & g_0 & g_1 & \cdots\cdots & g_n & & \\ & & \ddots & \ddots & & \ddots & \\ & & & g_0 & g_1 & \cdots\cdots & g_n \end{array}\right) \begin{array}{l} \left.\vphantom{\begin{array}{c}a\\a\\a\\a\end{array}}\right\} n \text{ rows} \\ \left.\vphantom{\begin{array}{c}a\\a\\a\\a\end{array}}\right\} m \text{ rows} \end{array}$$

is called the *Sylvester matrix* of f, g and denoted $\operatorname{Syl}_x(f,g)$.

Let K be the field of fractions of the coefficient ring R. From now on we will treat the f, g as polynomials over K, i.e. as elements of the ring $K[x]$. We know that for every positive integer k the set $K[x]_k = \{k \in K[x] \,|\, \deg h \leq k\}$ of polynomials with degrees not exceeding k is a vector space of dimension $k+1$. It can be equipped with a power basis $\{1, x, \ldots, x^k\}$. Fix the two polynomials $f = f_0 + f_1 x + \cdots + f_m x^m$ and $g = g_0 + g_1 x + \cdots + g_n x^n$. Consider a Cartesian product

$$V = K[x]_{n-1} \times K[x]_{m-1},$$

where m, n are the respective degrees of f and g. Taking the power bases in both factors, we construct a basis of V:

$$\mathscr{B} = \big\{(1,0), (x,0), \ldots, (x^{n-1}, 0), (0,1), (0,x), \ldots, (0, x^{m-1})\big\}.$$

Let $\Phi_{f,g} : V \to K[x]_{m+n-1}$ be a linear map defined by the formula

$$\Phi_{f,g}(a,b) := f \cdot a + g \cdot b.$$

It is straightforward to show that the matrix of $\Phi_{f,g}$, with respect to the basis $\mathscr{B}$ of V and the power basis of $K[x]_{m+n-1}$, has a form

$$\left(\begin{array}{ccc:ccc} f_0 & & & g_0 & & \\ \vdots & \ddots & & \vdots & \ddots & \\ f_m & & f_0 & g_n & & g_0 \\ & \ddots & \vdots & & \ddots & \vdots \\ & & f_m & & & g_n \end{array}\right).$$

$\underbrace{\qquad\qquad}_{n \text{ columns}} \quad \underbrace{\qquad\qquad}_{m \text{ columns}}$

Observation 8.2.12. *With the above notation, the matrix of $\Phi_{f,g}$ is the transpose of the Sylvester matrix of f and g.*

We will now establish a relation between the greatest common divisor of two polynomials and their Sylvester matrix. Recall that the rank of a matrix is the number of its linearly independent rows (or equivalently columns).

Theorem 8.2.13. *If $f, g \in K[x]$ are two polynomials, then*

$$\deg\big(\gcd(f,g)\big) = \deg(f) + \deg(g) - \operatorname{rank}\big(\operatorname{Syl}_x(f,g)\big).$$

Proof. Let again $m := \deg f$ and $n := \deg g$. We are going to describe the kernel of the map $\Phi_{f,g}$ defined above. Let $h := \gcd(f,g)$ and $d := \deg h$. Further denote $f^* := {}^{f}\!/_{h}$ and $g^* := {}^{g}\!/_{h}$. Take a polynomial $c \in K[x]$. A pair $(g^* \cdot c, -f^* \cdot c) \in K[x] \times K[x]$ belongs to V, the domain of $\Phi_{f,g}$, if and only if

$$\deg(g^* \cdot c) \leq n-1 \qquad \text{and} \qquad \deg(f^* \cdot c) \leq m-1.$$

We thus have

$$n-1 \geq \deg g^* + \deg c = \deg g - \deg h + \deg c = n - d + \deg c$$

and so it follows that $\deg c < d$. The second inequality leads to the same conclusion. On the other hand, if we take any $c \in K[x]_{d-1}$, then $\Phi_{f,g}(g^*c, -f^*c) = 0$. This shows that the map $K[x]_{d-1} \to \ker \Phi_{f,g}$, which sends c to $(g^*c, -f^*c)$ is an isomorphism (the injectivity is trivial). Hence, the dimension of the kernel of $\Phi_{f,g}$ equals d. From the course of linear algebra we know that the dimensions of the kernel and the image of a linear map are related by the formula

$$\dim V = \dim \operatorname{im} \Phi_{f,g} + \dim \ker \Phi_{f,g}.$$

Now, the dimension of $V = K[x]_{n-1} \times K[x]_{m-1}$ is $n+m$, while the dimension of the image of $\Phi_{f,g}$ equals the rank of its matrix, which we know to be the transpose of $\operatorname{Syl}_x(f,g)$, by Observation 8.2.12. Hence we obtain an equality

$$n + m = \operatorname{rank} \operatorname{Syl}_x(f,g) + d,$$

which concludes the proof. □

An immediate consequence of the previous theorem is the following result.

Corollary 8.2.14. *The resultant of two polynomials f and g vanishes if and only if their Sylvester matrix is not invertible, i.e.*

$$\operatorname{res}_x(f,g) = 0 \iff \det\big(\operatorname{Syl}_x(f,g)\big) = 0.$$

Proof. Proposition 8.2.6 asserts that $\operatorname{res}_x(f,g) = 0$ if and only if f, g have a nontrivial common factor. The previous theorem states that this happens if and only if the ranks of their Sylvester matrix is strictly smaller than its dimension, which means that it is not invertible. □

Exercise 16. Prove Corollary 8.2.14 directly without relying on the previous theorem.

Corollary 8.2.15. *The resultant of two polynomials $f, g \in R[x]$ vanishes if and only if there are polynomials $a, b \in R[x]$ such that*

$$f \cdot a + g \cdot b = 0 \qquad \text{and} \qquad \deg a < \deg g, \quad \deg b < \deg f.$$

Proof. By the previous corollary $\operatorname{res}_x(f,g) = 0$ if and only if $\det \operatorname{Syl}_x(f,g) = 0$. This holds if and only if the kernel of $\Phi_{f,g}$ is not null. The last condition is equivalent to the existence of $a \in K[x]_{n-1}$ and $b \in K[x]_{m-1}$ such that $fa + gb = 0$. Clearing the denominators, we may assume that a, b have coefficients in R. □

Exercise 17. Prove the above corollary using Corollary 1.3.5.

Let us write down one more fact that follows immediately from basic properties of the determinant.

Observation 8.2.16. *For every $\alpha, \beta \in K$ one has*

$$\det\Big(\mathrm{Syl}_x(\alpha \cdot f, \beta \cdot g)\Big) = \alpha^{\deg(g)} \cdot \beta^{\deg(f)} \cdot \det\Big(\mathrm{Syl}_x(f,g)\Big).$$

We are now in a position to present the main theorem of this section. The following result is due to J. Sylvester[2]. It is commonly used as a definition of the resultant in place of Eq. (8.11).

Theorem 8.2.17 (Sylvester)**.** *If f and g are two nonzero polynomials, then*

$$\mathrm{res}_x(f,g) = (-1)^{\deg f \cdot \deg g} \cdot \det \mathrm{Syl}_x(f,g).$$

Proof. By the previous observation we may assume without loss of generality that both f and g are monic. In this case, Eq. (8.11) simplifies to

$$\mathrm{res}_x(f,g) = \prod_{\substack{1 \le i \le m \\ 1 \le j \le n}} (\xi_i - \zeta_j).$$

Regarding $\xi_1, \dots, \xi_m$ and $\zeta_1, \dots, \zeta_n$ as indeterminates we may write the coefficients $f_0, \dots, f_{m-1}$ and $g_0, \dots, g_{n-1}$ of f and g as symmetric polynomials in ξ_i's and ζ_j's, by Viète's formulæ. Consequently we can treat $r := \mathrm{res}_x$ and $s := \det\big(\mathrm{Syl}_x(f,g)\big)$ as multivariate polynomials, $r, s \in P$ where $P = \mathbb{Z}[\xi_1, \dots, \xi_m, \zeta_1, \dots, \zeta_n]$ or $P = \mathbb{Z}_p[\xi_1, \dots, \xi_m, \zeta_1, \dots, \zeta_n]$ if the characteristic of K is $\mathrm{char}\, K = p \neq 0$. We will show that $r = (-1)^{mn} \cdot s$.

Corollary 8.2.14 asserts that s vanishes if and only if r does. This happens if and only if $\xi_{i_0} = \zeta_{j_0}$ for some indices i_0, j_0, by Proposition 8.2.6. Write s as a univariate polynomial in ξ_{i_0} with coefficients in a multivariate polynomial ring. Applying Little Bézout Theorem (cf. Proposition 1.1.18) we conclude that $(\xi_{i_0} - \zeta_{j_0})$ divides s. It follows that s is divisible by r.

Now, the (total) degree of r as a multivariate polynomial equals $m \cdot n$. Let us compute the (total) degree of s. The determinant of a $n \times n$ matrix $M = [m_{ij}]$ is by definition (see Eq. (3.5))

$$\det(M) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} m_{i\sigma(i)}$$

Apply this formula to the Sylvester matrix. Look at the structure of $\mathrm{Syl}_x(f,g)$. A nonzero term in the above sum has a form

$$f_{i_1} \cdots f_{i_n} \cdot g_{j_1} \cdots g_{j_m},$$

where the indices are subject to the relation

$$(i_1 + \cdots + i_n) + (j_1 + \cdots + j_m) = n \cdot m.$$

[2] James Joseph Sylvester (born 3 September 1814, died 15 March 1897) was an English mathematician and a fellow of Royal Society of London for Improving Natural Knowledge.

Therefore the total degree of s is precisely $m \cdot n$.

By the previous two paragraphs we conclude that r divides s and they have equal degrees. Thus there is a constant c (either $c \in \mathbb{Z}$ or $c \in \mathbb{Z}_p$, depending on the characteristic of K) such that $s = c \cdot r$. We shall prove that $c = (-1)^{mn}$. It suffices to compute values of the polynomials r and s for some specific choice of variables $\xi_1, \ldots, \xi_m$ and $\zeta_1, \ldots, \zeta_n$. Take

$$\xi_1 = \cdots = \xi_m = 1 \qquad \text{and} \qquad \zeta_1 = \cdots = \zeta_n = 0.$$

This means that

$$f = (x-1)^m = x^m + \binom{m}{1} x^{m-1}(-1)^1 + \cdots + \binom{m}{k} x^{m-k}(-1)^k + \cdots + (-1)^m,$$

while $g = x^n$. It is clear that

$$r(1, \ldots, 1, 0, \ldots, 0) = (1-0)\cdots(1-0) = 1.$$

On the other hand, $\mathrm{Syl}_x(f,g)$ is an upper-triangular matrix of a form

$$\begin{pmatrix}
(-1)^m & f_1 & \cdots & f_{n-1} & 1 & & \\
 & (-1)^m & f_1 & \cdots & f_{n-1} & 1 & \\
 & & \ddots & \ddots & & \ddots & \ddots \\
 & & & (-1)^m & f_1 & \cdots & f_{n-1} & 1 \\
\hdashline
0 & 0 & \cdots & 0 & 1 & & & \\
 & 0 & 0 & \cdots & 0 & 1 & & \\
 & & \ddots & \ddots & & \ddots & \ddots & \\
 & & & 0 & 0 & \cdots & 0 & 1
\end{pmatrix}$$

The determinant of this matrix equals $(-1)^{mn}$. Hence we have $c = s(1, \ldots, 1, 0, \ldots, 0) = (-1)^{mn}$, as desired. □

Example 86. Take the same two polynomials as in the previous example:

$$f = x^3 - x^2 + 4x - 4 \qquad \text{and} \qquad g = x^3 + x^2 - 16x - 16.$$

Construct the Sylvester's matrix of f and g

$$\mathrm{Syl}_x(f,g) = \left(\begin{array}{rrrrrr}
-4 & 4 & -1 & 1 & 0 & 0 \\
0 & -4 & 4 & -1 & 1 & 0 \\
0 & 0 & -4 & 4 & -1 & 1 \\
\hline
-16 & -16 & 1 & 1 & 0 & 0 \\
0 & -16 & -16 & 1 & 1 & 0 \\
0 & 0 & -16 & -16 & 1 & 1
\end{array}\right).$$

The determinant of this matrix equals 60000 and so it follows from Sylvester's theorem that $\mathrm{res}_x(f,g) = -60000$.

Exercise 18. Reprove Proposition 8.2.8 using Theorem 8.2.17.

Exercise 19. In many textbooks, a Sylvester's matrix is defined by writing the coefficients in a descending (rather than ascending, as we did) order of indices:

$$\begin{pmatrix} f_m & f_{m-1} & \cdots\cdots & f_0 & & \\ & f_m & f_{m-1} & \cdots\cdots & f_0 & \\ & & \ddots & \ddots & & \ddots \\ & & & f_m & f_{m-1} & \cdots\cdots & f_0 \\ g_n & g_{n-1} & \cdots\cdots & g_0 & & \\ & g_n & g_{n-1} & \cdots\cdots & g_0 & \\ & & \ddots & \ddots & & \ddots \\ & & & g_n & g_{n-1} & \cdots\cdots & g_0 \end{pmatrix}.$$

Prove that the discriminant of the above matrix is $(-1)^{mn} \cdot \det\big(\mathrm{Syl}_x(f,g)\big)$, hence is exactly equal to the resultant $\mathrm{res}_x(f,g)$.

Bézout formula

Although simple and elegant, Sylvester theorem has limitations. The dimension of the Sylvester matrix equals the sum of the degrees of the involved polynomials. Hence for high degree polynomials, computation of the resultant may be very time consuming. Moreover, if the coefficients of the polynomials in question are floating-point numbers, Sylvester formula is known to be badly conditioned. In this section we describe an alternative method of computing resultants that uses smaller matrices. It is named after É. Bézout.

Take two polynomials f and g. We shall treat their indeterminates as two independent variables, say x and y. Observe that a bivariate polynomial $f(x)g(y)-f(y)g(x)$ vanishes for $x = y$. Hence, it is divisible by a binomial $x - y$. Consequently, the quotient

$$B_{f,g}(x,y) := \frac{f(x)\cdot g(y) - f(y)\cdot g(x)}{x-y} \tag{8.13}$$

is in fact a polynomial. It is clear from definition, that the degree of $B_{f,g}$ with respect to each of the two variables is $M-1$, where M is the maximum of the degrees of f and g. Denote by b_{ij} the coefficient of $x^i y^j$ in $B_{f,g}$, so that

$$B_{f,g} = \sum_{i=0}^{M-1}\sum_{j=0}^{M-1} b_{ij} x^i y^j.$$

and write the polynomial $B_{f,g}$ in a matrix form

$$B_{f,g} = \big(1, x^1, \ldots, x^{M-1}\big) \cdot \begin{pmatrix} b_{00} & b_{10} & \cdots & b_{0,M-1} \\ b_{10} & b_{11} & \cdots & b_{1,M-1} \\ \vdots & \vdots & & \vdots \\ b_{M-1,0} & b_{M-1,1} & \cdots & b_{M-1,M-1} \end{pmatrix} \cdot \begin{pmatrix} 1 \\ y^1 \\ \vdots \\ y^{M-1} \end{pmatrix}.$$

Definition 8.2.18. The matrix

$$\mathrm{Bez}_x(f,g) := \begin{pmatrix} b_{00} & \cdots & b_{0,M-1} \\ \vdots & & \vdots \\ b_{M-1,0} & \cdots & b_{M-1,M-1} \end{pmatrix}$$

of the coefficients of the polynomial $B_{f,g}$ is called the *Bézout matrix* of f and g.

Observe that swapping the variables x and y, we reverse the signs of both: the numerator and the denominator in Eq. (8.13). This means that $B_{f,g}$ is a symmetric polynomial and so $\mathrm{Bez}_x(f,g)$ is a symmetric matrix. On the other hand, swapping the polynomials f and g, we change the sign of the numerator alone, hence we have

$$\mathrm{Bez}_x(g,f) = -\mathrm{Bez}_x(f,g).$$

Before we establish a relation between the Bézout matrix and the resultant, we first derive an alternative method for constructing $\mathrm{Bez}_x(f,g)$.

Lemma 8.2.19. *Let $f = f_0 + f_1 x + \cdots + f_m x^m$ and $g = g_0 + g_1 x + \cdots + g_n x^n$ be two polynomials. Then for $j > 0$ and $i < M-1$ one has*

$$b_{ij} = b_{i+1,j-1} + \det\begin{pmatrix} f_{i+1} & f_j \\ g_{i+1} & g_j \end{pmatrix}.$$

Proof. Expand the numerator of $B_{f,g}$ as it appears in Eq. (8.13):

$$f(x)\cdot g(y) - f(y)\cdot g(x) = (x-y)\Big(\sum_{i=0}^{M-1}\sum_{j=0}^{M-1} b_{ij}x^i y^j\Big) = \sum_{i=0}^{M-1}\Big(\sum_{j=0}^{M-1} b_{ij}x^{i+1}y^j - \sum_{j=0}^{M-1} b_{ij}x^i y^{j+1}\Big).$$

Let c_{ij} be the coefficient of $x^i y^j$ in the above formula. We have

$$\det\begin{pmatrix} f_i & f_j \\ g_i & g_j \end{pmatrix} = c_{ij} = b_{i-1,j} - b_{i,j-1}.$$

Changing the indexation we obtain the desired formula $b_{ij} = b_{i+1,j-1} + c_{i+1,j}$. □

Corollary 8.2.20. *Under the assumption of the previous lemma, we have*

$$b_{ij} = \sum_{k=1}^{\min\{j+1,n-i\}} \det\begin{pmatrix} f_{i+k} & f_{j+1-k} \\ g_{i+k} & g_{j+1-k} \end{pmatrix},$$

with the convention that a coefficient is null, when its index (respectively indices) is not in the support of the polynomial.

Proof. Applying the previous lemma recursively we have

$$\begin{aligned} b_{ij} &= c_{i+1,j} + b_{i+1,j-1} \\ &= c_{i+1,j} + c_{i+2,j-1} + b_{i+2,j-2} \\ &\vdots \\ &= \sum_{k=1}^{M} c_{i+k,j+1-k} = \sum_{k=1}^{M} \det \begin{pmatrix} f_{i+k} & f_{j+1-k} \\ g_{i+k} & g_{j+1-k} \end{pmatrix}. \end{aligned}$$

□

The recursive formula for b_{ij} in Lemma 8.2.19 let us build the Bézout matrix starting from the bottom-left (and top-right by symmetry) corner and filling the successive diagonals. The following algorithm is essentially due to E.-W. Chionh, H. Zhang and R. Goldman.

Algorithm 8.2.21. *Given two polynomials $f = f_0 + f_1 x + \cdots + f_m x^m$ and $g = g_0 + g_1 x + \cdots + g_n x^n$, this algorithm constructs the Bézout matrix $\mathrm{Bez}_x(f,g)$.*

1. *Make sure that $\deg f \leq \deg g$, if it is not the case then construct the matrix $\mathrm{Bez}_x(g,f)$ exchanging the roles of the two polynomials and invert the signs of all the entries when returning it;*
2. *initialize the matrix $\mathrm{Bez}_x(f,g)$ of size $n \times n$ filling it with zeros;*
3. *loop through every row i from $n-1$ to 0 and every column j from 0 to i (i.e. loop bottom-to-top in row-major order through all the elements on and below the main diagonal):*
   a) *compute*
   $$c := \det \begin{pmatrix} f_{i+1} & f_j \\ g_{i+1} & g_j \end{pmatrix};$$
   b) *if $i+1 \leq n-1$ and $j-1 \geq 0$, which means that the entry $b_{i+1,j-1}$ to the left and down from the current position exists, then add it to c, setting $c := c + b_{i+1,j-1}$;*
   c) *update entries of $\mathrm{Bez}_x(f,g)$ symmetrically with respect to the main diagonal, setting*
   $$b_{i,j} := c \qquad \text{and} \qquad b_{j,i} := c;$$
4. *return $\mathrm{Bez}_x(f,g)$, possibly inverting the signs of all entries if the two polynomials were exchanged in the first step.*

Example 87. We demonstrate Algorithm 8.2.21 using again the same pair of polynomials

$$f = x^3 - x^2 + 4x - 4 \qquad \text{and} \qquad g = x^3 + x^2 - 16x - 16.$$

Using the above algorithm, the matrix $\mathrm{Bez}_x(f,g)$ is constructed as

$$\mathrm{Bez}_x(f,g) = \begin{pmatrix} \det\left(\begin{smallmatrix} f_1 & f_0 \\ g_1 & g_0 \end{smallmatrix}\right) & \det\left(\begin{smallmatrix} f_2 & f_0 \\ g_2 & g_0 \end{smallmatrix}\right) & \det\left(\begin{smallmatrix} f_3 & f_0 \\ g_3 & g_0 \end{smallmatrix}\right) \\ \det\left(\begin{smallmatrix} f_2 & f_0 \\ g_2 & g_0 \end{smallmatrix}\right) & \det\left(\begin{smallmatrix} f_2 & f_1 \\ g_2 & g_1 \end{smallmatrix}\right) + \det\left(\begin{smallmatrix} f_3 & f_0 \\ g_3 & g_0 \end{smallmatrix}\right) & \det\left(\begin{smallmatrix} f_3 & f_1 \\ g_3 & g_1 \end{smallmatrix}\right) \\ \det\left(\begin{smallmatrix} f_3 & f_0 \\ g_3 & g_0 \end{smallmatrix}\right) & \det\left(\begin{smallmatrix} f_3 & f_1 \\ g_3 & g_1 \end{smallmatrix}\right) & \det\left(\begin{smallmatrix} f_3 & f_2 \\ g_3 & g_2 \end{smallmatrix}\right) \end{pmatrix}$$
$$= \begin{pmatrix} -128 & 20 & -12 \\ 20 & 12+(-12) & -20 \\ -12 & -20 & 2 \end{pmatrix}.$$

On the other hand, in order to construct the Bézout matrix from definition, we would have to obtain first the polynomial $B_{f,g}$:

$$B_{f,g} = \frac{(x^3-x^2+4x-4)\cdot(y^3+y^2-16y-16)-(y^3-y^2+4y-4)\cdot(x^3+x^2-16x-16)}{x-y}$$
$$= 2x^2y^2-20x^2y-20xy^2-12x^2-12y^2+20x+20y-128.$$

The reader may compare the coefficients of the polynomial with the entries of the matrix we have constructed.

We will show that the Bézout matrix can be obtained from the Sylvester matrix, hence these two notions are closely related. Again, let $f = f_0 + f_1x + \cdots + f_mx^m$ and $g = g_0 + g_1x + \cdots + g_nx^n$. Without loss of generality we may assume that $m \leq n$. Elevate the degree of f appending zeros if needed and write

$$f = f_0 + f_1x + \cdots + f_mx^m + \cdots + f_nx^n,$$

where $f_{m+1} = \cdots = f_n = 0$, if the degree of g is strictly greater than the degree of f. Form a Sylvester-like matrix of dimension $2n \times 2n$:

$$S := \begin{pmatrix} f_0 & \cdots & f_m & \cdots & 0 & & \\ & \ddots & & \ddots & & \ddots & \\ & & f_0 & \cdots & f_m & \cdots & 0 \\ g_0 & \cdots & \cdots & \cdots & g_n & & \\ & \ddots & & & & \ddots & \\ & & g_0 & \cdots & \cdots & \cdots & g_n \end{pmatrix}.$$

Lemma 8.2.22. *The determinant of S equals*

$$\det S = \mathrm{lc}(g)^{\deg g - \deg f} \cdot \det \mathrm{Syl}_x(f,g).$$

Proof. The assertion is trivial if the degrees of the two polynomials coincide, since then $S = \mathrm{Syl}_x(f,g)$. On the other hand, if $m = \deg f$ is strictly smaller than $n = \deg g$, then the only nonzero entries in the last $n-m$ columns are the coefficients of g and so they all lie on and below the main diagonal. Thus it suffices to expand the determinant $n-m$ times with respect to the last column using Laplace formula (Theorem 3.4.2). □

Subdivide the matrix S into four $n \times n$ submatrices, denoted F_1, F_2 and G_1, G_2:

$$S = \left(\begin{array}{c:c} F_1 & F_2 \\ \hdashline G_1 & G_2 \end{array}\right) = \left(\begin{array}{cccc:cccc} f_0 & \cdots & f_{n-1} & & f_n & & & \\ & \ddots & \vdots & & \vdots & \ddots & & \\ & & f_0 & & f_1 & \cdots & f_n & \\ \hdashline g_0 & \cdots & g_{n-1} & & g_n & & & \\ & \ddots & \vdots & & \vdots & \ddots & & \\ & & g_0 & & g_1 & \cdots & g_n & \end{array}\right).$$

All four cells are triangular Toeplitz matrices. Cells F_1 and G_1 are upper-triangular, while F_2 and G_2 are lower-triangular. A product of two lower-triangular matrices is again lower-triangular. We claim that in our case the matrices F_2 and G_2 commute, due to their special structure. Indeed, their products equal

$$F_2 \cdot G_2 = G_2 \cdot F_2 = \begin{pmatrix} d_0 & & & & \\ d_1 & \ddots & & & \\ & \ddots & \ddots & & \\ \vdots & & \ddots & \ddots & \\ d_{n-1} & \cdots & d_1 & d_0 \end{pmatrix}, \qquad \text{where} \qquad d_k = \sum_{i=0}^{k} f_{n-k+i} g_{n-i} = \sum_{j=0}^{k} f_{n-j} g_{n-k+j}.$$

In particular we have $G_2F_2 - F_2G_2 = 0$. This yields an equality:

$$\begin{pmatrix} G_2 & -F_2 \\ 0 & I \end{pmatrix} \cdot S = \begin{pmatrix} G_2 & -F_2 \\ 0 & I \end{pmatrix} \cdot \begin{pmatrix} F_1 & F_2 \\ G_1 & G_2 \end{pmatrix} = \begin{pmatrix} G_2F_1 - F_2G_1 & 0 \\ G_1 & G_2 \end{pmatrix}. \tag{8.14}$$

We need to investigate the product of cells with mixed indices in the top-left corner of the matrix on the right.

Lemma 8.2.23. *Let B denote a $n \times n$ matrix obtained from $\mathrm{Bez}_x(g, f)$ by reversing the order of its rows. Then*

$$B = G_2F_1 - F_2G_1.$$

Proof. Denote the right-hand-side of the above formula by A. Compute an element a_{ij} in the i-th row and j-th column of A:

$$\begin{aligned} a_{ij} &= \big(g_{n-i+1} \cdot f_{j-1} + g_{n-i+2} \cdot f_{j-2} + \cdots\big) - \big(f_{n-i+1} \cdot g_{j-1} + f_{n-i+2} \cdot g_{j-2} + \cdots\big) \\ &= \sum_{k=1}^{\min\{i,j\}} \big(g_{n-i+k} f_{j-k} - f_{n-i+k} g_{j-k}\big). \end{aligned}$$

Substitute $n - i'$ for i and $j' + 1$ for j. Corollary 8.2.20 yields

$$= \sum_{k=1}^{\min\{n-i', j'+1\}} \big(f_{j'+1-k} g_{i'+k} - f_{i'+k} g_{j'+1-k}\big) = -b_{i',j'}.$$

Therefore, the matrix A is constructed from $\operatorname{Bez}_x(f,g)$ by reversing the order of rows an inverting the signs of all the coefficients. □

In Theorem 8.2.13 we observed that the rank deficiency of the Sylvester matrix is the same as the degree of the greatest of the greatest common divisor. This is true also for the Bézout matrix, thank to the identity Eq. (8.14).

Proposition 8.2.24. *Let $f, g \in K[x]$ be two nonzero polynomials, then*

$$\deg\bigl(\gcd(f,g)\bigr) = \max\bigl\{\deg(f),\deg(g)\bigr\} - \operatorname{rank}\bigl(\operatorname{Bez}_x(f,g)\bigr).$$

Proof. As before, without loss of generality we may assume that $\deg f \leq \deg g$. The matrix G_2 is lower-triangular with nonzero entries on the main diagonal, hence it is invertible. Consequently the block matrix

$$\begin{pmatrix} G_2 & -F_2 \\ 0 & I \end{pmatrix}$$

is invertible, too. Thus the rank of the product

$$\begin{pmatrix} G_2 & -F_2 \\ 0 & I \end{pmatrix} \cdot S$$

is just the rank of S. By the means of Eq. (8.14) we can write

$$\operatorname{rank} S = \operatorname{rank}(G_2F_1 - F_2G_1) + \operatorname{rank} G_2.$$

We have $\operatorname{rank} S = \operatorname{rank}\operatorname{Syl}_x(f,g) + (n-m)$ and so Lemma 8.2.23 yields

$$\operatorname{rank}\operatorname{Syl}_x(f,g) + (n-m) = \operatorname{rank}\operatorname{Bez}_x(f,g) + n.$$

Apply Theorem 8.2.13 to the left-hand-side to get

$$(n+m) - \deg\gcd(f,g) + (n-m) = \operatorname{rank}\operatorname{Bez}_x(f,g) + n.$$ □

We are now ready to show the main theorem of this section which establishes a link between the Bézout matrix and the resultant.

Theorem 8.2.25. *Let $f = f_0 + f_1x + \cdots + f_mx^m$ and $g = g_0 + g_1x + \cdots + g_nx^n$ be two polynomials. Assume that $\deg f \leq \deg g$. Then*

$$\det\bigl(\operatorname{Bez}_x(f,g)\bigr) = \begin{cases} g_n^{n-m} \cdot \operatorname{res}_x(f,g), & \text{if } n \equiv 0,1 \pmod 4 \\ -g_n^{n-m} \cdot \operatorname{res}_x(f,g), & \text{if } n \equiv 2,3 \pmod 4. \end{cases}$$

Proof. Compute the determinants of both sides of Eq. (8.14)

$$\det G_2 \cdot \det S = \det G_2 \cdot \det\bigl(G_2F_1 - F_2G_1\bigr).$$

The determinant of G_2, being a power of the leading coefficients of g, is nonzero. Canceling the common terms and using Lemma (8.2.22) together with Sylvester theorem we arrive at a formula:

To be checked. Where this -1^{n^2} is from?

$$(-1)^{n^2} \cdot g_n^{n-m} \cdot \operatorname{res}_x(f,g) = \det(G_2F_1 - F_2G_1). \tag{8.15}$$

Lemma 8.2.23 states that $G_2F_1 - F2G_1$ is obtained from $\operatorname{Bez}_x(f,g)$ by reversing the order of rows an inverting the signs of all the coefficients. These operations amount to $n + \lfloor n/2 \rfloor$ sign changes of the determinant. Using Eq. (8.15) we derive the desired identity

$$\det \operatorname{Bez}_x(f,g) = (-1)^{n^2} \cdot (-1)^{n+\lfloor \frac{n}{2} \rfloor} \cdot g_n^{n-m} \cdot \operatorname{res}_x(f,g) = \begin{cases} g_n^{n-m} \cdot \operatorname{res}_x(f,g), & \text{if } n \equiv 0,1 \pmod 4 \\ -g_n^{n-m} \cdot \operatorname{res}_x(f,g), & \text{if } n \equiv 2,3 \pmod 4. \end{cases}$$

□

Example 88. In the previous example we constructed the Bézout matrix of the polynomials

$$f = x^3 - x^2 + 4x - 4 \qquad \text{and} \qquad g = x^3 + x^2 - 16x - 16.$$

Now, Theorem 8.2.25 asserts that $\operatorname{res}_x(f,g) = -\det \operatorname{Bez}_x(f,g) = -60000$.

The notion of the Bézout matrix provides also another method for counting real roots of a univariate polynomial. Recall that in Chapter we defined the *signature* $\operatorname{sgn}(M)$ of a symmetric matrix M as the difference between the number of positive and negative eigenvalues.

missing

Proposition 8.2.26. *If f, g are two nonzero univariate polynomials in an indeterminate x, then the signature of $\operatorname{Bez}_x(f, g \cdot f')$ equals the difference between the number of distinct roots of f on which g is positive and the number of distinct roots of f on which g is negative. Rigorously*

$$\operatorname{sgn}\big(\operatorname{Bez}_x(f, g \cdot f')\big) = \sharp\big\{\xi \in \mathbb{R} \mid f(\xi) = 0 \wedge g(\xi) > 0\big\} - \sharp\big\{\xi \in \mathbb{R} \mid f(\xi) = 0 \wedge g(\xi) < 0\big\}.$$

Proof - BPR03, Corollary 9.21

Important special cases of the last proposition are the following ones:

Corollary 8.2.27.

1. *The number of distinct real roots of f equals*

$$\operatorname{sgn}\big(\operatorname{Bez}_x(f, f')\big).$$

2. *Let a be a fixed real number such that $f(a) \neq 0$. The number of roots of f greater than a equals*

$$\frac{1}{2}\Big(\operatorname{sgn}\big(\operatorname{Bez}_x\big(f, (x-a) \cdot f'\big)\big) + \operatorname{sgn}\big(\operatorname{Bez}_x\big(f, f'\big)\big)\Big),$$

while the number of roots of f smaller than a equals

$$\frac{1}{2}\Big(\operatorname{sgn}\big(\operatorname{Bez}_x\big(f,(a-x)\cdot f'\big)\big)+\operatorname{sgn}\big(\operatorname{Bez}_x\big(f,f'\big)\big)\Big).$$

3. *Let $a<b$ be two real numbers such that $f(a)\neq 0$ and $f(b)\neq 0$. The number of distinct roots of f in the interval (a,b) equals*

$$\frac{1}{2}\Big(\operatorname{sgn}\big(\operatorname{Bez}_x\big(f,(x-a)\cdot(b-x)\cdot f'\big)\big)+\operatorname{sgn}\big(\operatorname{Bez}_x\big(f,f'\big)\big)\Big).$$

missing Recall (see) that if M is a symmetric matrix and $c_M=\det(M-x\cdot I)$ is the characteristic polynomial of M, then the signature of M can be computed as the difference between the number of sign changes in f and $f(-x)$. We can rephrase the corollary in form of an algorithm.

Algorithm 8.2.28. *Given a nonzero polynomial f and $a,b\in\mathbb{R}\sup\{\pm\infty\}$, $a<b$, one can count the number of distinct roots of f on (a,b) as follows:*

1. *initialize $g:=1$;*
2. *if $a\neq-\infty$, the update g setting $g:=x-a$;*
3. *if $b\neq\infty$, then update g setting $g:=(b-x)\cdot g$;*
4. *calculate the derivative f' of f using Algorithm 1.1.30;*
5. *using Algorithm 8.2.21, construct the Bézout matrices $B_0:=\operatorname{Bez}_x(f,f')$ and $B_1:=\operatorname{Bez}_x(f,gf')$;*
6. missing *use Algorithm to compute the signatures of B_0 and B_1;*
7. *return $1/2\cdot\big(\operatorname{sgn}(B_0)+\operatorname{sgn}(B_1)\big)$.*

In case, when we need to know only the number of all distinct real roots of f, it is enough to compute a single Bézout matrix and so the above algorithm simplifies to:

Algorithm 8.2.29. *Given a nonzero polynomial f, one can count the number of real roots of f as follows:*

1. *calculate the derivative f' of f using Algorithm 1.1.30;*
2. *construct the Bézout matrix $\operatorname{Bez}_x(f,f')$ using Algorithm 8.2.21;*
3. missing *use Algorithm to compute the signature of $\operatorname{Bez}_x(f,f')$;*
4. *return the computed signature.*

Resultant via companion matrix

Recall (see Section 3.8 of Chapter 3) that the *companion matrix* C_f of a monic polynomial $f=f_0+f_1x+\cdots+f_{m-1}x^{m-1}+x^m$ has a form

$$C_f=\begin{pmatrix}0&0&\cdots&0&-f_0\\1&0&\cdots&0&-f_1\\0&1&\cdots&0&-f_2\\\vdots&&\ddots&&\vdots\\0&0&\cdots&1&-f_{m-1}\end{pmatrix}.$$

It satisfies the condition $\det(xI - C_f) = f$ and so the eigenvalues of C_f are precisely the roots of f. Here we develop a method for computing the resultant of two polynomials, which utilizes the companion matrices.

Take two nonzero polynomials $f = f_0 + f_1 x + \cdots + f_m x^m$ and $g = g_0 + g_1 x + \cdots + g_n x^n$ with coefficients in some integral domain R. It is not necessary, but convenient to assume that $\deg f \leq \deg g$. If it is not the case we may swap the two polynomials using Observation 8.2.7. As we did before, by K denote the field of fractions of the coefficient ring R. We will treat f and g as elements of an euclidean ring $K[x]$. Normalize f setting $\overline{f} := {}^1/_{\mathrm{lc}(f)} \cdot f$.

Theorem 8.2.30. *If $C_{\overline{f}}$ is the companion matrix of $\overline{f}$ then*

$$\mathrm{res}_x(f, g) = \mathrm{lc}(f)^{\deg g} \cdot \det g(C_{\overline{f}}).$$

Proof. Assume that $\zeta_1, \ldots, \zeta_n$ are all the roots of g in some big enough field L, containing K. As always in this chapter, the multiple roots are repeated. The polynomial g can be factored over L into linear terms $g = \mathrm{lc}(g) \cdot (x - \zeta_1) \cdots (x - \zeta_n)$. It follows that

$$g(C_{\overline{f}}) = \mathrm{lc}(g) \cdot (C_{\overline{f}} - \zeta_1 I) \cdots (C_{\overline{f}} - \zeta_n I).$$

Here I is the identity matrix of size $m \times m$, where $m = \deg f$ is the degree of f. Compute the determinants of both sides

$$\det g(C_{\overline{f}}) = \mathrm{lc}(g)^m \cdot (-1)^m \cdot \overline{f}(\zeta_1) \cdots (-1)^m \cdot \overline{f}(\zeta_n) = (-1)^{mn} \cdot \mathrm{res}_x(g, \overline{f}) = \mathrm{res}_x(\overline{f}, g).$$

The assertion follows now immediately from Corollary 8.2.10. □

Theorem 8.2.30 may be directly converted into an algorithm.

Algorithm 8.2.31. *Given two nonzero polynomials $f, g \in R[x]$, this algorithm computes their resultant* $\mathrm{res}_x(f, g)$.

1. *Construct the companion matrix $C_{\overline{f}}$ of $\overline{f}$;*
2. *using Horner's rule compute $g(C_{\overline{f}})$;*
3. *return* $\mathrm{lc}(f)^{\deg(g)} \cdot \det g(C_{\overline{f}})$.

Time complexity analysis. Let $m := \deg f$ and $n := \deg g$. The computation of $g(C_{\overline{f}})$ using Horner's rule needs n multiplications (and the same number of additions) of $m \times m$ matrices. If we use standard matrix arithmetic this gives a total of $n \cdot m^3$ scalar multiplications. The computation of the determinant in step 3 has complexity $O(m^3)$ id we use either Gauss (Algorithm 3.4.6) or Jordan–Bareiss (Algorithm 3.4.7) algorithm. Therefore Algorithm 8.2.31 needs $O\big((\deg f)^3 \cdot \deg g\big)$ scalar operations. Observe that this complexity is not symmetric with respect to f and g. This is why in the first paragraph we suggested that it is profitable to make sure that it is f which has a lower degree.

Example 89. Take again $f = x^3 - x^2 + 4x - 4$ and $g = x^3 + x^2 - 16x - 16$. Construct the companion matrix of $\overline{f}$:

$$C_{\overline{f}} = \begin{pmatrix} 0 & 0 & 4 \\ 1 & 0 & -4 \\ 0 & 1 & 1 \end{pmatrix}.$$

Compute

$$g(C_{\overline{f}}) = \begin{pmatrix} 0 & 0 & 4 \\ 1 & 0 & -4 \\ 0 & 1 & 1 \end{pmatrix}^3 + \begin{pmatrix} 0 & 0 & 4 \\ 1 & 0 & -4 \\ 0 & 1 & 1 \end{pmatrix}^2 - 16 \cdot \begin{pmatrix} 0 & 0 & 4 \\ 1 & 0 & -4 \\ 0 & 1 & 1 \end{pmatrix}^1 - 16 \cdot \begin{pmatrix} 0 & 0 & 4 \\ 1 & 0 & -4 \\ 0 & 1 & 1 \end{pmatrix}^0 = \begin{pmatrix} -12 & \\ -20 & \\ 2 & \end{pmatrix}$$

Theorem 8.2.30 asserts that $\operatorname{lc}(f)^3 \cdot \det C_{\overline{f}} = -60000$ is the resultant $\operatorname{res}_x(f, g)$.

We observed earlier that the Sylvester matrix of two polynomials can be interpreted as a matrix of a certain linear map. Its kernel is closely related to the greatest common divisor of these polynomials (see Theorem 8.2.13). The same is true in the situation discussed here. Let $R := {}^{K[x]}/_{\langle f \rangle}$ be the quotient ring of $K[x]$ modulo the ideal generated by f. The ring R can be identified with the set of remainders modulo f, hence it consists of all the polynomials of degree strictly less than m:

$$R \cong \{h \in K[x] \mid \deg h \leq m-1\} = K[x]_{m-1}.$$

We know, that this is a vector space over K of dimension m, with a basis $\mathscr{B} = \{1, x, \ldots, x^{m-1}\}$. Consider a linear endomorphism $\varphi_g \in \operatorname{End}_K R$ induced by a multiplication of elements of R by g (rigorously speaking: by the class of g modulo $\langle f \rangle$):

$$\varphi_g(h) := \big((g \cdot h) \bmod f\big).$$

Proposition 8.2.32. *The matrix of φ_g with respect to the power basis equals $g(C_{\overline{f}})$.*

Proof. Denote the matrix of φ_g by M_g. First assume that $g = x$. Then for $j < m-1$, we have

$$\varphi_x\big(x^j\big) = \big(x^{j+1} \bmod f\big) = x^{j+1}.$$

On the other hand, if $j = m-1$, the trivial congruence $f \equiv 0 \pmod f$ yields

$$\varphi_x\big(x^{m-1}\big) = -\frac{f_0}{f_m} - \frac{f_1}{f_m} \cdot x - \cdots - \frac{f_{m-1}}{f_m} \cdot x^{m-1}.$$

It follows that the matrix of φ_x has a form

$$M_x = \begin{pmatrix} 0 & 0 & \cdots & 0 & {}^{-f_0}/_{f_m} \\ 1 & 0 & \cdots & 0 & {}^{-f_1}/_{f_m} \\ 0 & 1 & \cdots & 0 & {}^{-f_2}/_{f_m} \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & 1 & {}^{-f_{m-1}}/_{f_m} \end{pmatrix}.$$

Hence it is nothing else but the companion matrix $C_{\overline{f}}$ of $\overline{f}$.

It is easy to observe that the map $g \mapsto \varphi_g$ is a ring homomorphism from $K[x]$ to $\mathrm{End}_K R$. Thus, for $g = g_0 + g_1 x + \cdots + g_n x^n$ we have

$$M_g = g_0 \cdot I + g_1 \cdot M_x + \cdots + g_n \cdot (M_x)^n = g(C_{\overline{f}}),$$

where I is the identity matrix. □

Using Proposition 8.2.32 we obtain an alternative version of Algorithm 8.2.31.

Algorithm 8.2.33. *Given two nonzero polynomials $f, g \in K[x]$ over a field K, this algorithm computes their resultant* $\mathrm{res}_x(f, g)$.

1. *For every $j \in \{0, \ldots, \deg(f) - 1\}$ compute the remainder r_j of $g \cdot x^j$ modulo f;*
2. *form a matrix M_g, whose columns consist of the coefficients of $r_0, \ldots, r_{m-1}$;*
3. *return* $\mathrm{lc}(f)^{\deg(g)} \cdot \det(M_g)$.

Time complexity analysis. Denote again $m := \deg f$ and $n := \deg g$. Division with a remainder has the same asymptotic complexity as multiplication. The loops in the first step of the algorithm is executed m times and each iteration needs one division. (The multiplication $g \cdot x^j$ can be ignored, since in the power basis representation it boils down to appending zeros to the list of coefficients.) Hence the complexity of the first step is $O(m^2 \cdot n)$, when using standard arithmetic, and $O\big(m \cdot d \cdot \lg d\big)$, where $d = \max\{m, n\}$, when using fast Fourier transform. The computation of the determinant in the last step has the same complexity as in Algorithm 8.2.31 and so the time complexity of the whole algorithm is bounded by $O(d^3)$, $d = \max\{\deg f, \deg g\}$.

Example 90. Use the same two polynomials as in the last few examples. The successive remainders computed by the above algorithms are

$$\begin{aligned} r_0 &= (g \bmod f) = 2x^2 - 20x - 12, \\ r_1 &= \big((g \cdot x) \bmod f\big) = -18x^2 - 20x + 8, \\ r_2 &= \big((g \cdot x^2) \bmod f\big) = -38x^2 + 80x - 72. \end{aligned}$$

Write their coefficients side-by-side to make up the matrix

$$M_g = \begin{pmatrix} -12 & 8 & -72 \\ -20 & -20 & 80 \\ 2 & -18 & -38 \end{pmatrix}.$$

This is the same matrix that we obtained in the previous example.

Proposition 8.2.34. *Use the above notation, then*

$$\deg\big(\gcd(f, g)\big) = \deg f - \operatorname{rank} M_g.$$

Proof. Denote $d := \deg\gcd(f,g)$ and write f, g in a form

$$f = f^* \cdot \gcd(f,g), \qquad g = g^* \cdot \gcd(f,g),$$

where f^*, g^* are some relatively prime polynomials. We claim that the kernel of φ_g consists of polynomials $h \in K[x]_{m-1}$ divisible by f^*. Indeed, if $h = f^* \cdot h^*$ for some $h^* \in K[x]$, then

$$h \cdot g = f^* \cdot h^* \cdot g^* \cdot \gcd(f,g) = h^* \cdot g^* \cdot f \equiv 0 \pmod{f}.$$

Therefore $\varphi_g(h) = 0$. Conversely, if $\varphi_g(h) = 0$, then f divides $h \cdot g$. Hence there is a polynomial $e \in K[x]$ such that $f \cdot e = g \cdot h$. Consequently

$$f^* \cdot \gcd(f,g) \cdot e = g^* \cdot \gcd(f,g) \cdot h.$$

Then $f^* \cdot e = g^* \cdot h$ nut f^* and g^* are relatively prime and so f^* divides h. This proves the claim.

From the course of linear algebra we know that

$$\dim K[x]_{m-1} = \dim \operatorname{im} \varphi_g + \dim \ker \varphi_g. \tag{8.16}$$

The dimension of the image of φ_g equals the rank of the associated matrix. On the other hand, we have proved that $\ker \varphi_g = \{h \in K[x]_{m-1} \mid f^* \text{ divides } h\}$. Therefore $\dim \ker \varphi_g = \dim K[x]_{m-1} - \deg f^*$. Substituting these values to Eq. (8.16) we obtain

$$\dim K[x]_{m-1} = \operatorname{rank} M_g + \dim K[x]_{m-1} - \deg f^*.$$

The assertion follows now from the fact that $\deg f^* = \deg f - \deg\gcd(f,g)$. □

Exercise 20. There is one more way to compute the resultant using companion matrices. Let $f, g \in R$ be two polynomials. Denote $m = \deg f$, $n = \deg g$ and let $\overline{f} := {}^1/_{\operatorname{lc}(f)} \cdot f$, $\overline{g} := {}^1/_{\operatorname{lc}(g)} \cdot g$ be monic polynomials obtained from f, g. Prove the following formula:

$$\operatorname{res}_x(f,g) = \operatorname{lc}(f)^{\deg g} \cdot \operatorname{lc}(g)^{\deg f} \cdot \det(C_{\overline{f}} \otimes I_n - I_m \otimes C_{\overline{g}}).$$

Here $\otimes$ is the Kronecker product and I_m, I_n are identity matrices. What is the time complexity of this method of computing the resultant?

Euclidean-like algorithm for resultant

The resultant of two polynomials vanishes if and only it they have a nontrivial common factor (cf. Proposition 8.2.6). Hence, there must be a link between the resultant and the greatest common divisor. It turns out that the resultant can be computed using a variant of the Euclidean algorithm. The key ingredient is the following proposition.

Proposition 8.2.35.

1. *Let f, g be two polynomials with coefficients in a field K. If r is the remainder of f modulo g, then*

$$\operatorname{res}_x(f,g) = (-1)^{\deg f \deg g} \cdot \operatorname{lc}(g)^{\deg f - \deg r} \cdot \operatorname{res}_x(g,r).$$

2. *Let f, g be two polynomials with coefficients in a unique factorization domain R. If r is the pseudo-remainder of f modulo g, then*

$$\operatorname{res}_x(f,g) = (-1)^{\deg f \deg g} \cdot \operatorname{lc}(g)^{M} \cdot \operatorname{res}_x(g,r),$$

where

$$M = \begin{cases} (\deg g - \deg f)(\deg g - 1) - \deg r, & \text{if } \deg f < \deg g, \\ \deg f - \deg r, & \text{if } \deg f \geq \deg g. \end{cases}$$

Proof. Compute the quotient q and remainder r of f modulo g:

$$f = g \cdot q + r, \qquad \text{where} \quad \deg r < \deg g.$$

Let $\zeta_1, \ldots, \zeta_n$ be all the roots of g (as always in this section, multiple roots are repeated). Express the resultant of f and g using Proposition 8.2.3:

$$\begin{aligned} \operatorname{res}_x(f,g) &= (-1)^{\deg f \deg g} \cdot \operatorname{res}_x(g,f) \\ &= (-1)^{\deg f \deg g} \cdot \operatorname{lc}(g)^{\deg f} \cdot \prod_{i=1}^{n} f(\zeta_i) \\ &= (-1)^{\deg f \deg g} \cdot \operatorname{lc}(g)^{\deg f} \cdot \prod_{i=1}^{n} \Big(\underbrace{g(\zeta_i) \cdot q(\zeta_i)}_{=0} + r(\zeta_i) \Big) \\ &= (-1)^{\deg f \deg g} \cdot \operatorname{lc}(g)^{\deg f - \deg r} \cdot \operatorname{lc}(g)^{\deg r} \cdot \prod_{i=1}^{n} r(\zeta_i) \\ &= (-1)^{\deg f \deg g} \cdot \operatorname{lc}(g)^{\deg f - \deg r} \cdot \operatorname{res}_x(g,r). \end{aligned}$$

This proves the first assertion. We leave the second one as an easy exercise to the reader. □

Remark 64. In the second case of the above proposition, the exponent M can easily happen to be negative. Since $\operatorname{res}_x(f,g)$ always lays in R, by Observation 8.2.4, this means that $\operatorname{res}_x(g,r)$ is divisible in R by a power of the leading coefficient of f.

Proposition 8.2.35 gives rise to a recursive method for computing the resultant, that closely resembles the Euclidean algorithm.

Corollary 8.2.36.

1. *Let $f = f_0 + f_1 x + \cdots + f_m x^m$ and $g = g_0 + g_1 x + \cdots + g_n x^n$ be two polynomials over a field K, then*

$$\operatorname{res}_x(f,g) = \begin{cases} 0, & \text{if } f = 0 \text{ or } g = 0, \\ f_0^n, & \text{if } f = f_0 \equiv \text{const}, \\ g_0^m, & \text{if } g = g_0 \equiv \text{const}, \\ (-1)^{mn} \cdot g_n^{m-k} \cdot \operatorname{res}_x(g,r), & \text{if } r \text{ is the remainder of } f \text{ modulo } g \text{ and } k = \deg r. \end{cases}$$

2. *Let $f = f_0 + f_1 x + \cdots + f_m x^m$ and $g = g_0 + g_1 x + \cdots + g_n x^n$ be two polynomials over a unique factorization domain R, then*

$$\operatorname{res}_x(f,g) = \begin{cases} 0, & \text{if } f = 0 \text{ or } g = 0, \\ f_0^n, & \text{if } f = f_0 \equiv \text{const}, \\ g_0^m, & \text{if } g = g_0 \equiv \text{const}, \\ (-1)^{mn} \cdot g_n^M \operatorname{res}_x(g,r), & \text{if } r \text{ is the pseudo-remainder of } f \text{ modulo } g, \\ & k = \deg r \text{ and } M \text{ is as in Proposition 8.2.35.} \end{cases}$$

To improve the clarity of exposition, let us rephrase the first point of this corollary in form of a recursive algorithm (the second point can be rephrased analogously).

Algorithm 8.2.37. *Given two polynomials $f, g \in K[x]$ with coefficients in a field K, this algorithm returns their resultant.*

1. *If any of the two polynomials is null, then return* 0 *and quit;*
2. *if $g = c$ is a constant polynomial, then return $c^{\deg f}$ and quit;*
3. *analogously, if $f = c$ is a constant polynomial, then return $c^{\deg g}$ and quit;*
4. *compute the remainder r of f modulo g;*
5. *execute this algorithm recursively to compute the resultant* $\operatorname{res}_x(g,r)$*;*
6. *return* $(-1)^{\deg f \deg g} \cdot \operatorname{lc}(g)^{\deg f - \deg r} \cdot \operatorname{res}_x(g,r)$.

Exercise 21. Convert the above algorithm to an iterative (rather than recursive) form.

Exercise 22. Adjust the above algorithm to work with polynomials over a unique factorization domains. Use the second point of Corollary 8.2.36.

Example 91. Take the same two polynomials as in last few examples: $f = x^3 - x^2 + 4x - 4$ and $g = x^3 + x^2 - 16x - 16$. Compute their resultant using the above algorithm:

$$\begin{aligned} \operatorname{res}_x(x^3 - x^2 + 4x - 4, x^3 + x^2 - 16x - 16) &= (-1)^9 \cdot 1^1 \cdot \operatorname{res}_x(x^3 + x^2 - 16x - 16, -2x^2 + 20x + 12) \\ &= -1 \cdot (-1)^6 \cdot -2^2 \cdot \operatorname{res}_x(-2x^2 + 20x + 12, 100x + 50) \\ &= -4 \cdot (-1)^2 \cdot 100^2 \cdot \operatorname{res}_x(100x + 50, \frac{3}{2}) \\ &= -40000 \cdot \frac{3}{2}^1 = -60000 \end{aligned}$$

Recall that Bézout's identity (see Theorem 1.3.4) expresses the greatest common divisor of two elements as their linear combination. There is an analogue of Bézouts identity for resultants, that establishes yet another link between resultants and greatest common divisors.

Proposition 8.2.38. *If $f, g \in K[x]$ are two polynomials with coefficients in a field K, then there exist such polynomials $a, b \in K[x]$ that*

$$a \cdot f + b \cdot g = \operatorname{res}_x(f, g).$$

Proof. Without loss of generality we may assume that f and g are both nonzero, since otherwise $\operatorname{res}_x(f,g) = 0 = 0 \cdot f + 0 \cdot g$. Proceed by induction on $N := \max\{\deg f, \deg g\}$. For $N = 0$, both f nad g must be constant, say $f = c$ and $g = d$, $c, d \in K$. It follows from Observation 8.2.2 that $\operatorname{res}_x(f,g) = 1 = {}^1\!/_c \cdot f + 0 \cdot g$.

Suppose that we have proved the thesis for polynomials of degrees strictly less than N. Further assume that $N = \deg f \geq \deg g$ (otherwise, if $\deg f < \deg g$, exchange the roles of f and g using Observation 8.2.7). Compute the quotient q and the remainder r of f modulo g:

$$f = g \cdot q + r, \qquad \deg r < \deg g \leq N. \tag{8.17}$$

Using Proposition 8.2.35 express the resultant of f and g in terms of g and r:

$$\operatorname{res}_x(f,g) = (-1)^{\deg f \deg g} \cdot \operatorname{lc}(g)^{\deg f - \deg r} \cdot \operatorname{res}_x(g, r).$$

If the degree of g is strictly less than N, we may apply the inductive hypothesis to $\operatorname{res}_x(g,r)$. Therefore, there are polynomials a_1, b_1 such that

$$\operatorname{res}_x(g,r) = a_1 \cdot g + b_1 \cdot r.$$

Together with Eq. (8.17) this yields

$$\begin{aligned}
\operatorname{res}_x(f,g) &= \pm \operatorname{lc}(g)^{\deg f - \deg r} \cdot \big(a_1 \cdot g + b_1 \cdot (f - gq)\big) \\
&= \Big(\pm \operatorname{lc}(g)^{\deg f - \deg r} \cdot b_1\Big) \cdot f + \Big(\pm \operatorname{lc}(g)^{\deg f - \deg r} \cdot (a_1 - b_1 \cdot q)\Big) \cdot g \\
&= a \cdot f + b \cdot g.
\end{aligned}$$

On the other hand, if $\deg g \geq N$, we divide g modulo r and write $g = q_1 \cdot r + r_1$ for some $\deg r_1 < \deg r < N$. By inductive hypothesis there are a_1, b_1 such that $\operatorname{res}_x(r, r_1) = a_1 \cdot r + b_1 \cdot r_1$. Consequently

$$\begin{aligned}
\operatorname{res}_x(f,g) &= \pm \operatorname{lc}(g)^{\deg f - \deg r} \cdot \operatorname{res}_x(g, r) \\
&= \pm \operatorname{lc}(g)^{\deg f - \deg r} \cdot \operatorname{lc}(r)^{\deg g - \deg r_1} \cdot \operatorname{res}_x(r, r_1) \\
&= \pm \operatorname{lc}(g)^{\deg f - \deg r} \cdot \operatorname{lc}(r)^{\deg g - \deg r_1} \cdot \Big(a_1 \cdot (f - gq) + b_1 \cdot \big(-f q_1 + g \cdot (1 + q \cdot q_1)\big)\Big) \\
&= \pm \operatorname{lc}(g)^{\deg f - \deg r} \cdot \operatorname{lc}(r)^{\deg g - \deg r_1} \cdot \big(a_1 - b_1 q_1\big) \cdot f \\
&\quad \pm \operatorname{lc}(g)^{\deg f - \deg r} \cdot \operatorname{lc}(r)^{\deg g - \deg r_1} \cdot \big(b_1 + b_1 q q_1 - a_1 q\big) \cdot g \\
&= a \cdot f + b \cdot g.
\end{aligned}$$

□

Remark 65. It can be proved (see [22, Chaper 3, § 5]) that the coefficients of a and b are polynomials in coefficients of f and g, hence the above proposition holds also for polynomials over an arbitrary integral domain.

Modular resultant computation

In previous chapters we have observed—more than once—advantages of modular approach to computation of various "global" objects. In this section we present algorithms for computing resultant of (multivariate) polynomials, due to G.E. Collins, which are modular in nature. Recall that the basic idea behind modular methods is to find one or (usually) more homomorphisms from a given ring (over which the computation is expensive) to some "simpler" rings, in which we can easily compute partial answers. Those are subsequently combined together to reconstruct the final result.

Let P,R be two rings. Every ring homomorphism $\varphi : P \to R$ induces a homomorphism of the polynomial rings over P and R, which maps $\sum_{0\leqslant i\leqslant n} f_i x^i$ to $\sum_{0\leqslant i\leqslant n} \varphi(f_i)x^i$. Abusing the notation harmlessly, we will denote the later homomorphism again by φ.

Theorem 8.2.39 (Collins, 1971)**.** *Let P,R be two rings and $\varphi : P \to R$ be a ring homomorphism. Further let $f, g \in P[x]$ be two polynomials. If $\deg \varphi(f) = \deg f$, then*

$$\varphi\big(\operatorname{res}_x(f,g)\big) = \varphi\big(\operatorname{lc}(f)\big)^{\deg g - \deg \varphi(g)} \cdot \operatorname{res}_x\big(\varphi(f), \varphi(g)\big).$$

Proof. Let $f = f_0 + f_1 x + \cdots + f_m x^m$ and $g = g_0 + g_1 x + \cdots + g_n x^n$. Set $d := \deg \varphi(g)$. We have $\varphi(g) = \varphi(g_0) + \varphi(g_1)x + \cdots + \varphi(g_d)x^d + 0x^{d+1} + \cdots + 0x^n$. Take the Sylvester matrix $\operatorname{Syl}_x(f,g)$ of f, g and map all its entries to R. Denote the resulting matrix by $\varphi\big(\operatorname{Syl}_x(f,g)\big)$. We have

$$\varphi\big(\operatorname{Syl}_x(f,g)\big) = \begin{pmatrix} \varphi(f_0) & \cdots\cdots\cdots & \varphi(f_m) & & \\ & \ddots & & \ddots & \\ & & \varphi(f_0) & \cdots\cdots\cdots & \varphi(f_m) \\ \varphi(g_0) & \cdots & \varphi(g_d) & \cdots\quad 0 & \\ & \ddots & & \ddots & \ddots \\ & & \varphi(g_0) & \cdots\quad \varphi(g_d) & \cdots\quad 0 \end{pmatrix}.$$

Let us compute the determinant of $\varphi\big(\operatorname{Syl}_x(f,g)\big)$. Laplace formula (Theorem 3.4.2) applied $(n-d)$ times to the last column yields:

$$\det \varphi\big(\operatorname{Syl}_x(f,g)\big) = (-1)^{m\cdot(n-d)} \cdot \varphi(f_m)^{n-d} \cdot \det \operatorname{Syl}_x\big(\varphi(f), \varphi(g)\big).$$

The assertion follows now from Sylvester theorem (Theorem 8.2.17). □

The most straightforward application of modular method is to compute the resultant of two polynomials over $\mathbb{Z}$, using the canonical maps $\mathbb{Z} \to \mathbb{Z}/p_i\mathbb{Z}$, for some prime numbers $p_1, \ldots, p_k$. Let $f, g \in \mathbb{Z}[x]$ be two polynomials and p a prime number. As

in Section 2.5, by $f_{(p)}$ (respectively $g_{(p)}$) we denote a polynomial obtained from f (resp. g) by reducing every coefficient modulo p. Hence, if $f = f_0 + f_1 x + \cdots + f_m x^m$, then $f_{(p)} = (f_0 \bmod p) + (f_1 \bmod p)x + \cdots + (f_m \bmod p)x^m$. If p does not divide the leading coefficients of f and g, then $\deg f_{(p)} = \deg f$ and $\deg g_{(p)} = \deg g$. It follows from Theorem 8.2.39 that

$$\operatorname{res}_x(f_{(p)}, g_{(p)}) = \big(\operatorname{res}_x(f,g) \bmod p\big).$$

Thus, if we have **enough** distinct primes $p_1, \ldots, p_k$, we may reconstruct the resultant of f, g from its modular images $r_1 = \operatorname{res}_x(f_{p_1}, g_{p_1}), \ldots, r_k = \operatorname{res}_x(f_{p_k}, g_{p_k})$ by the means of Chinese remainder theorem (see Theorem 2.2.2). We only need to find how many primes is enough. Recall that $\|f\|_\infty$ is the maximum of the absolute values of the coefficients of f, that is

$$\|f_0 + f_1 x + \cdots + f_m x^m\|_\infty := \max\big\{|f_0|, |f_1|, \ldots, |f_m|\big\}.$$

Corollary 3.4.22 gives us a rough bound for the absolute value of the resultant.

Observation 8.2.40. *Let $f, g \in \mathbb{Z}[x]$ be two polynomials with integer coefficients of degrees m and n, respectively. Denote $B := \max\big\{\|f\|_\infty, \|g\|_\infty\big\}$. Then*

$$|\operatorname{res}_x(f,g)| \le B^{m+n} \cdot (m+n)^{\frac{m+n}{2}}.$$

Although the above bound often suffices, we may do better. The resultant is the determinant of the Sylvester matrix, which has a rather specific structure. Exploiting this structure, we get a slightly better estimate.

Lemma 8.2.41. *If f, g are two univariate polynomials with integer coefficients, then*

$$|\operatorname{res}_x(f,g)| \le (\deg f + \deg g)! \cdot \|f\|_\infty^{\deg g} \cdot \|g\|_\infty^{\deg f}.$$

Proof. Denote $m := \deg f$ and $n := \deg g$. Expand the determinant of the Sylvester matrix of f, g. It is a sum of $(m+n)!$ terms, where each term is a product of n coefficients of f and m coefficients of g. Bounding the absolute value of each coefficient by the respective norm, we arrive at the above estimate. □

The previous discussion proves the correctness of the following procedure.

Algorithm 8.2.42. *Given two polynomials $f, g \in \mathbb{Z}[x]$ with integer coefficients, this algorithm computes their resultant.*

1. *Compute the bound*

$$B := (\deg f + \deg g)! \cdot \|f\|_\infty^{\deg g} \cdot \|g\|_\infty^{\deg f}.$$

2. *Find distinct prime numbers $p_1, \ldots, p_k$ that do not divide the leading coefficients of f and g and such that*

$$p_1 \cdots p_k \ge 2B.$$

3. *Using one of the other algorithms described in this chapter, for every prime p_i compute the resultant*
$$r_i := \operatorname{res}_x(f_{(p_i)}, g_{(p_i)}).$$
4. *Using Chinese remainder theorem, find $r \in \{-B, \ldots, B\}$ such that*
$$r \equiv r_i \pmod{p_i} \qquad \text{for every} \quad i \leq k.$$
5. *Return* $\operatorname{res}_x(f, g) = r$.

Example 92. We will compute the resultant of the same two polynomials that we have encountered a few times before:
$$f = x^3 - x^2 + 4x - 4, \qquad g = x^3 + x^2 - 16x - 16.$$
We have $\|f\|_\infty = 4$ and $\|g\|_\infty = 16$. Hence Lemma 8.2.41 gives us the bound for the absolute value of the resultant
$$B = 188743680.$$
(On the other hand, the bound obtained by Observation 8.2.40 is $B' = 3623878656$.) We will use prime moduli: $47, 53, 59, 61, 67$. Their product equals 600662303 and exceeds $2B$. We shall compute the resultants modulo each prime. For example, modulo 47 we have
$$f_{(47)} = x^3 + 46x^2 + 4x + 43 \qquad \text{and} \qquad g_{(47)} = x^3 + x^2 + 31x + 31.$$
Hence the resultant equals $\operatorname{res}_x(f_{(47)}, g_{(47)}) = 19$. Analogously we compute the resultant modulo other primes. Eventually we obtain the following system of congruences:
$$\begin{cases} r \equiv 19 \pmod{47} \\ r \equiv 49 \pmod{53} \\ r \equiv 3 \pmod{59} \\ r \equiv 24 \pmod{61} \\ r \equiv 32 \pmod{67} \end{cases}$$
Using Chinese remainder theorem we obtain a solution $r = -60000$ of the system. This is the sought resultant of f and g.

We shall now study the case of two multivariate polynomials over a field. Thus we have $f, g \in K[x_1, \ldots, x_k, y]$. Although K may be an arbitrary field, the algorithm, we will present, was originally devised as a sub-procedure for Algorithm 8.2.46 (discussed next) and so it is mostly intended to be used in case $K = \mathbb{F}_p$. For an element $\alpha \in K$ and a polynomial $h \in K[x_1, \ldots, x_k]$ by $\varphi_\alpha(h)$ we denote the polynomial $h(x_1, \ldots, x_{k-1}, \alpha)$ in $(k-1)$ variables, obtained by substituting α for x_k. It is clear that $\varphi_\alpha : K[x_1, \ldots, x_k] \to K[x_1, \ldots, x_{k-1}]$ is a ring homomorphism. As we did before, we extend it to a homomorphism $\varphi_\alpha : K[x_1, \ldots, x_k][y] \to K[x_1, \ldots, x_{k-1}][y]$.

Let $\mathrm{lc}_y(f), \mathrm{lc}_y(g)$ denote the leading coefficients of f and g with respect to y. If, for a given $\alpha \in K$, we have

$$\varphi_\alpha\big(\mathrm{lc}_y(f)\big) \neq 0 \qquad \text{and} \qquad \varphi_\alpha\big(\mathrm{lc}_y(g)\big) \neq 0, \tag{8.18}$$

then Theorem 8.2.39 asserts that

$$\varphi_\alpha\big(\mathrm{res}_y(f,g)\big) = \mathrm{res}_y\big(\varphi_\alpha(f), \varphi_\alpha(g)\big).$$

Assume that d is the degree of $\mathrm{res}_y(f,g)$ with respect to the variable x_k. If $\alpha_0, \ldots, \alpha_d$ are distinct elements of K satisfying condition 8.18 and $r_i := \mathrm{res}_y(\varphi_{\alpha_i}(f), \varphi_{\alpha_i}(g))$, then $\mathrm{res}_y(f,g)$ can be reconstructed from $r_0, \ldots, r_d$ by polynomial interpolation. (Lagrange theorem says that the resulting interpolating polynomial belongs to $K(x_1, \ldots, x_{k-1})[x_k]$, but we know a priori that $\mathrm{res}_y(f,g)$ is in fact an element of $K[x_1, \ldots, x_k]$.) Before we write an explicit algorithm, we still have to find a bound for d. To this end we prove:

Proposition 8.2.43. *Let $f, g \in K[x_1, \ldots, x_k, y]$ be two polynomials, the degree of their resultant with respect to the variable x_k can be bounded as*

$$\deg_{x_k} \mathrm{res}(f,g) \leq \deg_y f \cdot \deg_{x_k} g + \deg_y g \cdot \deg_{x_k} f.$$

Proof. Expanding the determinant of the Sylvester matrix $\mathrm{Syl}_y(f,g)$ we obtain a sum of products of a form $f_{i_1} \cdots f_{i_n} \cdot g_{j_1} \cdots g_{j_m}$, where $m = \deg_y f$ and $n = \deg_y g$. It is clear that the degree of such a product does not exceed

$$m \cdot \max\{\deg_{x_k} f_i \mid i \leq m\} + n \cdot \max\{\deg_{x_k} g_j \mid j \leq n\}$$

and this proves the assertion. □

We may now summarize our discussion in form of an algorithm. The presented algorithm is recursive in nature. Each recursion reduces the number of variables by one. The final univariate case must be resolved by other methods.

Algorithm 8.2.44 (Collins, 1971)**.** *Given two multivariate polynomials $f, g \in K[x_1, \ldots, x_k, y]$ this algorithm computes their resultant $\mathrm{res}_y(f,g)$ with respect to the variable y.*

1. *If f, g are univariate polynomials (i.e. $k = 0$), then compute the resultant $\mathrm{res}_y(f,g)$ using any of the other algorithm described in this chapter; return $\mathrm{res}_y(f,g)$ and quit.*
2. *Let $m := \deg_y f$, $m' := \deg_{x_k} f$, $n := \deg_y g$ and $n' := \deg_{x_k} g$.*
3. *Compute the upper bound for the degree of the resultant, set*

$$d := m \cdot n' + m' \cdot n.$$

4. *Find $d+1$ distinct elements $\alpha_0, \ldots, \alpha_d$ of the field K satisfying condition (8.18), i.e.*

$$\deg_y \varphi_{\alpha_j}(f) = \deg_y f \quad \text{and} \quad \deg_y \varphi_{\alpha_j}(g) = \deg_y g \qquad \text{for} \quad j \in \{0, \ldots, d\},$$

where $\varphi_{\alpha_j}(f) := f(x_1, \ldots, x_{k-1}, \alpha_j, y)$ and $\varphi_{\alpha_j}(g) := g(x_1, \ldots, x_{k-1}, \alpha_j, y)$. If such elements do not exist, then throw an exception and quit.

5. *Calling this algorithm recursively, compute the resultants*

$$r_j := \operatorname{res}_y\big(\varphi_{\alpha_j}(f), \varphi_{\alpha_j}(g)\big) \qquad \text{for} \quad j \in \{0, \ldots, d\}.$$

6. *Using methods described in Section 4.1, find an interpolating polynomial* $r \in K[x_1, \ldots, x_k]$ *such that*

$$\varphi_{\alpha_j}(r) = r_j \qquad \text{for} \quad j \in \{0, \ldots, d\}.$$

7. *Return* $\operatorname{res}_y(f, g) = r$.

Example 93. Take two bivariate polynomials

$$f = 2y^2x + 35y^2 + 37yx + 12y + 2x^2 + 35x, \qquad g = 37y^3x + yx + 3x^2$$

over a finite field $\mathbb{F}_{41}$. We have

$$m = 2, \quad m' = 2, \quad n = 3, \quad n' = 2,$$

hence $d = 10$. Scanning the successive element of the field $\mathbb{F}_{41}$, we may select elements $\alpha_0, \ldots, \alpha_{10}$ satisfying condition (8.18):

$$\alpha_0 = 1, \quad \alpha_1 = 2, \quad \alpha_2 = 4, \quad \alpha_3 = 5, \quad \alpha_4 = 6, \quad \alpha_5 = 7, \quad \alpha_6 = 8, \quad \alpha_7 = 9, \quad \alpha_8 = 10, \quad \alpha_9 = 11,$$

The corresponding reduced polynomials $\varphi_{\alpha_j}(f)$ and $\varphi_{\alpha_j}(g)$ are:

$$\begin{aligned}
f(x, \alpha_0) &= 37y^2 + 8y + 37 & g(x, \alpha_0) &= 37y^3 + y + 3 \\
f(x, \alpha_1) &= 39y^2 + 4y + 37 & g(x, \alpha_1) &= 33y^3 + 2y + 12 \\
f(x, \alpha_2) &= 2y^2 + 37y + 8 & g(x, \alpha_2) &= 25y^3 + 4y + 7 \\
f(x, \alpha_3) &= 4y^2 + 33y + 20 & g(x, \alpha_3) &= 21y^3 + 5y + 34 \\
f(x, \alpha_4) &= 6y^2 + 29y + 36 & g(x, \alpha_4) &= 17y^3 + 6y + 26 \\
f(x, \alpha_5) &= 8y^2 + 25y + 15 & g(x, \alpha_5) &= 13y^3 + 7y + 24 \\
f(x, \alpha_6) &= 10y^2 + 21y + 39 & g(x, \alpha_6) &= 9y^3 + 8y + 28 \\
f(x, \alpha_7) &= 12y^2 + 17y + 26 & g(x, \alpha_7) &= 5y^3 + 9y + 38 \\
f(x, \alpha_8) &= 14y^2 + 13y + 17 & g(x, \alpha_8) &= y^3 + 10y + 13 \\
f(x, \alpha_9) &= 16y^2 + 9y + 12 & g(x, \alpha_9) &= 38y^3 + 11y + 35 \\
f(x, \alpha_{10}) &= 18y^2 + 5y + 11 & g(x, \alpha_{10}) &= 34y^3 + 12y + 22
\end{aligned}$$

We now compute the resultants of the reduced polynomials. The companion matrix of $\varphi_{\alpha_0}(f)$ is $\left(\begin{smallmatrix} 0 & 40 \\ 1 & 2 \end{smallmatrix}\right)$, hence

$$r_0 = \operatorname{res}_y(\varphi_{\alpha_0}(f), \varphi_{\alpha_0}(g)) = \det\Big(\varphi_{\alpha_0}(g)\begin{pmatrix} 0 & 40 \\ 1 & 2 \end{pmatrix}\Big) = \det\begin{pmatrix} 11 & 11 \\ 30 & 30 \end{pmatrix} = 0.$$

Analogously, we compute $r_1, \ldots, r_{10}$ to get

$$r_1 = 6, \quad r_2 = 13, \quad r_3 = 10, \quad r_4 = 40, \quad r_5 = 34, \quad r_6 = 31, \quad r_7 = 16, \quad r_8 = 33, \quad r_9 = 9, \quad r_{10} =$$

Build the interpolating polynomial $r = \mathrm{res}_y(f, g)$:

$$r = 5x^8 + 11x^7 + 21x^6 - x^5 + 39x^4 + 7x^3.$$

Now, equipped with an efficient tool to deal with multivariate polynomials over finite fields, we may return to the polynomials over the integers and generalize Algorithm 8.2.42 to work with polynomials having more than one variable. First we generalize Lemma 8.2.41.

Lemma 8.2.45. *If $f, g \in \mathbb{Z}[x_1, \ldots, x_k][y]$ are two polynomials, then the absolute values of the coefficients the resultant $\mathrm{res}_y(f, g)$ do not exceed*

$$(\deg_y f + \deg_y g)! \cdot \|f\|_{infty}^{\deg_y g} \cdot \|g\|_{\infty}^{\deg_y f}.$$

Here the norm of a multivariate polynomial is defined recursively. If $k > 0$ and $f = f_0 + f_1 \cdot y + \cdots + f_m y^m$, with $f_i \in \mathbb{Z}[x_1, \ldots, x_k]$, then we set

$$\|f\|_\infty := \max\bigl\{\|f_0\|_\infty, \ldots, \|f_m\|_\infty\bigr\}.$$

The proof is fully analogous to the proof of Lemma 8.2.41. We leave the details to the reader. The following procedure is a straightforward generalization of Algorithm 8.2.42 to the multivariate case.

Algorithm 8.2.46 (Collins, 1971)**.** *Given two polynomials $f, g \in \mathbb{Z}[x_1, \ldots, x_k][y]$ with integer coefficients, this algorithm computes their resultant $\mathrm{res}_y(f, g)$ with respect to the variable y.*

1. *Compute the bound*

$$B := (\deg_y f + \deg_y g)! \cdot \|f\|_{\infty}^{\deg_y g} \cdot \|g\|_{\infty}^{\deg_y f}.$$

2. *Find distinct prime numbers $p_1, \ldots, p_k$ such that $p_1 \cdots p_k \geq 2B$ and*

$$\deg_y f_{(p_i)} = \deg_y f, \qquad \deg_y g_{(p_i)} = \deg_y g.$$

3. *For every prime p_i compute the resultant of the reduced polynomial*

$$r_i := \mathrm{res}_y(f_{(p_i)}, g_{(p_i)}),$$

   using Algorithm 8.2.44.

4. *Using Chinese remainder theorem for each coefficient, find $r \in \mathbb{Z}[x_1, \ldots, x_k]$ such that*

$$r \equiv r_i \pmod{p_i} \qquad \textit{for every} \quad i \leq k.$$

5. *Return $\mathrm{res}_y(f, g) = r$.*

8.3 Applications of resultants

Solving systems of multivariate polynomial equations

Definitely, the first and foremost application of resultants is for solving systems of multivariate polynomial equations. The basic idea is to eliminate all but one variables from the system, reducing the problem to that of solving a univariate equation. The key result here is the following one:

Theorem 8.3.1 (Elimination theorem)**.** *Let $f, g \in K[x_1, \ldots, x_k, y]$ be two polynomials with coefficients in a field K and $L \supset K$ be some algebraically closed field. Denote the resultant with respect to the last variable by $r := \operatorname{res}_y(f, g)$. If $\xi_1, \ldots, \xi_k, \zeta \in L$ are such that*

$$\begin{cases} f(\xi_1, \ldots, \xi_k, \zeta) = 0 \\ g(\xi_1, \ldots, \xi_k, \zeta) = 0, \end{cases}$$

then also $r(\xi_1, \ldots, \xi_k) = 0$.

Proof. The proof follows immediately from Proposition 8.2.38. Indeed, since

$$\operatorname{res}_y(f, g) = a \cdot f + b \cdot g,$$

for some polynomials a, b with coefficients in $K(x_1, \ldots, x_r)$, thus evaluating both sides at $\xi_1, \ldots, \xi_k, \zeta$ we have

$$r(\xi_1, \ldots, \xi_k) = a(\xi_1, \ldots, \xi_k, \zeta) \cdot f(\xi_1, \ldots, \xi_k, \zeta) + b(\xi_1, \ldots, \xi_k, \zeta) \cdot g(\xi_1, \ldots, \xi_k, \zeta) = 0.$$

□

Example 94. We will show how to apply the above proposition by solving the following system of equations (we have already encountered it earlier in the introduction to this chapter):

$$\begin{cases} 3x^2 - 3xy - 10x + 3y^2 + 5 = 0 \\ x^2 - 5xy + y^2 + 6y - 3 = 0. \end{cases} \tag{8.19}$$

Denote the polynomial in the first equation by f and the one in the second equation by g. Compute the resultant of these two polynomials using Sylvester's method:

$$\begin{aligned} \operatorname{res}_y(f, g) &= \det \begin{pmatrix} 3x^2 - 10x + 5 & -3x & 3 & 0 \\ 0 & 3x^2 - 10x + 5 & -3x & 3 \\ x^2 - 3 & -5x + 6 & 1 & 0 \\ 0 & x^2 - 3 & -5x + 6 & 1 \end{pmatrix} \\ &= 144x^4 - 1032x^3 + 2452x^2 - 2332x + 736. \end{aligned}$$

This way we eliminated y from the system. The resulting polynomial is a constant multiple of the one appearing in Eq. (8.2). Theorem 8.3.1 asserts that the roots of $\operatorname{res}_y(f, g)$ are precisely the abscissas of the solutions to the original system. They are approximately equal: $0.645, 1.44, 1.57, 3.52$. Now, as explained in the introduction,

we can either back-substitute these values to the system or choose to eliminate the other variable. The latter amounts to computing the resultant

$$\begin{aligned}\operatorname{res}_x(f,g) &= \det\begin{pmatrix} 3y^2+5 & -3y-10 & 3 & 0 \\ 0 & 3y^2+5 & -3y-10 & 3 \\ y^2+6y-3 & -5y & 1 & 0 \\ 0 & y^2+6y-3 & -5y & 1 \end{pmatrix} \\ &= 144y^4-456y^3+292y^2+116y-104.\end{aligned}$$

This is again a constant multiple of a polynomial we saw in the introduction. By Theorem 8.3.1 its roots are the ordinates of the solutions to the system Eq. (8.19). Approximately:

$$-0.536, 0.736, 0.875, 2.09.$$

We can now check all possible pair whether they actually solve the system. The valid pairs are:

$$(0.645, 0.736), (1.44, -0.536), (1.57, 2.09), (3.52, 0.875).$$

Example 95. For a second example take a trivariate system

$$\begin{cases} x^2-yz+z^2+1=0 \\ xy+xz-yz=0 \\ x^2+2y^2-xz=0. \end{cases} \tag{8.20}$$

Denote the polynomials by f, g and h. We reduce the system to a bivariate one by eliminating the variable z from it:

$$\begin{cases} \operatorname{res}_z(f,g)=0 \\ \operatorname{res}_z(g,h)=0. \end{cases}$$

This time let us compute the resultants using companion matrices (see Algorithm 8.2.31). Now f treated as a polynomial in a variable in z over $\mathbb{Q}[x,y]$ is monic and its companion matrix equals

$$C_f = \begin{pmatrix} 0 & -x^2-1 \\ 1 & y \end{pmatrix}.$$

Therefore

$$M_g = g(C_f) = \begin{pmatrix} xy & -x^3+x^2y-x+y \\ x-y & 2xy-y^2 \end{pmatrix}.$$

Consequently

$$\operatorname{res}_z(f,g) = \operatorname{lc}(f)^{\deg g} \det M_g = x^4-2x^3y+3x^2y^2-xy^3+x^2-2xy+y^2.$$

Analogously we compute $\operatorname{res}_z(g,h)$:

$$\operatorname{res}_z(g,h) = (x-y)^1 \cdot \det\begin{pmatrix} \frac{x^3+2xy^2-2y^3}{x-y} \end{pmatrix} = x^3+2xy^2-2y^3.$$

The resulting system (in variables x and y) is

$$\begin{cases} x^4 - 2x^3y + 3x^2y^2 - xy^3 + x^2 - 2xy + y^2 = 0 \\ x^3 + 2xy^2 - 2y^3 = 0. \end{cases}$$

Next we eliminate y from it obtaining a single polynomial in x;

$$r_x := \operatorname{res}_y\big(\operatorname{res}_z(f,g), \operatorname{res}_z(g,h)\big) = 25x^{12} + 48x^{10} + 32x^8 + 2x^6.$$

The roots of this polynomial are possible x-coordinates of the solutions of Eq. (8.20). Analogously we form univariate polynomials in y and z:

$$\begin{aligned} r_y &:= \operatorname{res}_x\big(\operatorname{res}_z(f,g), \operatorname{res}_z(g,h)\big) = 50y^{12} - 16y^{10} + 6y^8 + y^6 \\ r_z &:= \operatorname{res}_x\big(\operatorname{res}_y(f,g), \operatorname{res}_y(g,h)\big) = 25z^{12} + 63z^{10} + 46z^8 + 10z^6 + 2z^4. \end{aligned}$$

Of course, we could have eliminated variables in different order, too. Denote the set of roots of r_x by $\mathscr{X}$, of r_y by $\mathscr{Y}$ and of r_z by $\mathscr{Z}$. Approximately

$$\begin{aligned} \mathscr{X} &\approx \{0.000, -0.272 - 1.00i, -0.272 + 1.00i, -0.264i, 0.264i, 0.272 - 1.00i, 0.272 + 1.00i\} \\ \mathscr{Y} &\approx \{0.000, -0.562 - 0.312i, -0.562 + 0.312i, -0.342i, 0.342i, 0.562 - 0.312i, 0.562 + 0.312i\} \\ \mathscr{Z} &\approx \{0.000, -0.272 - 0.415i, -0.272 + 0.415i, -1.15i, -1.00i, 1.00i, 1.15i, 0.272 - 0.415i, 0.272 + \end{aligned}$$

Try out every possible triple $(\xi, \zeta, \eta) \in \mathscr{X} \times \mathscr{Y} \times \mathscr{Z}$ to find all the solutions to Eq. (8.20):

$$\begin{aligned} &\{(0.000, 0.000, -1.00i), (0.000, 0.000, 1.00i), (-0.272 - 1.00i, -0.562 + 0.312i, 0.272 - 0.415i), \\ &(-0.272 + 1.00i, -0.562 - 0.312i, 0.272 + 0.415i), (-0.264i, -0.342i, -1.15i), (0.264i, 0.342i, 1.1 \\ &(0.272 - 1.00i, 0.562 + 0.312i, -0.272 - 0.415i), (0.272 + 1.00i, 0.562 - 0.312i, -0.272 + 0.415i)\} \end{aligned}$$

Notice that although 0 is in $\mathscr{Z}$, it does not extend to any solution of the system!

It is evident from the previous example, that although the vanishing of the resultant is a necessary condition for an element of L to be a projection of a solution of the system of polynomial equations, it is not a sufficient one. Hence, in general the implication in Theorem 8.3.1 cannot be reversed. However, a partial conversion is still possible and is provided by the following theorem (see also Theorem in Chapter 9 for an analogous result for Gröbner bases).

missing

Theorem 8.3.2 (Extension theorem). *Let $f, g \in K[x_1, \ldots, x_k][y]$ be two multivariate polynomials with coefficients in a field K. Denote by r the resultant $\operatorname{res}_y(f,g)$ and by $\operatorname{lc}_y(f)$, $\operatorname{lc}_y(g)$ the leading coefficients of f and g, with respect to the variable y. If $\xi_1, \ldots, \xi_n$ are elements of some algebraically closed field $L \supset K$ such that $r(\xi_1, \ldots, \xi_k) = 0$ and both $\big(\operatorname{lc}_y(f)\big)(\xi_1, \ldots, \xi_k)$ and $\big(\operatorname{lc}_y(g)\big)(\xi_1, \ldots, \xi_k)$ are nonzero, then there is $\zeta \in L$ such that*

$$\begin{cases} f(\xi_1, \ldots, \xi_k, \zeta) = 0 \\ g(\xi_1, \ldots, \xi_k, \zeta) = 0. \end{cases}$$

In a nutshell, this theorem says that a partial solution $(\xi_1, \dots, \xi_k)$ extends to a solution $(\xi_1, \dots, \xi_k, \zeta)$ to the system $f = g = 0$ providing that the leading coefficients of f and g do not vanish on this partial solution.

Proof. In order to simplify notation write Ξ for $\xi_1, \dots, \xi_n$. By assumption, $\big(\mathrm{lc}_y(f)\big)(\Xi)$ is nonzero, hence the degree of the univariate polynomial $f(\Xi, y) \in L[y]$ is the same as the degree $\deg_y f$ of the multivariate polynomial f with respect to the variable y. Likewise, we have $\deg g(\Xi, y) = \deg_y g$. It follows from Theorem 8.2.39 that

$$\mathrm{res}_y\big(f(\Xi, y), g(\Xi, y)\big) = \big(\mathrm{res}_y(f, g)\big)(\Xi) = r(\Xi) = 0.$$

But now $f(\Xi, y)$ and $g(\Xi, y)$ are two univariate polynomials with coefficients in L. From the fact that their resultant vanishes we infer (see Proposition8.2.6) that there is a non-constant polynomials $h \in L[y]$ which divides both of them simultaneously. Every non-constant polynomial over an algebraically closed field has a root. Thus there is $\zeta \in L$ such that $h(\zeta) = 0$. It follows that $f(\Xi, \zeta) = g(\Xi, \zeta) = 0$. □

Example 96. Let us continue the last example and see how the extension theorem works here. We have observed that $\eta = 0$ is one of the roots of

$$r_z := \mathrm{res}_x\big(\mathrm{res}_y(f, g), \mathrm{res}_y(g, h)\big),$$

but it does not extend to any solution of Eq. (8.20). Let us denote $F := \mathrm{res}_y(f, g)$ and $G := \mathrm{res}_y(g, h)$. We have

$$\begin{aligned} F &= -x^3 + zx^2 + \big(-2z^2 - 1\big)x + z^3 + z, \\ G &= x^4 - 3zx^3 + 5z^2x^2 - z^3x. \end{aligned}$$

The leading coefficients (with respect to x) are $\mathrm{lc}_x F = -1$ and $\mathrm{lc}_x G = 1$. They do not vanish at zero, hence Theorem 8.3.2 asserts that there is a solution $(\xi, 0)$ above 0. Indeed substituting $z = 0$ in formulas for F and G we obtain a system

$$\begin{cases} -x^3 - x = 0 \\ x^4 = 0. \end{cases}$$

It is clear that both polynomials vanish at $\xi = 0$. Thus $(0, 0)$ is a solution (in fact a unique solution) to the system

$$\begin{cases} F = 0 \\ G = 0. \end{cases}$$

Let us see what happens if we try to extend it to a solution of Eq. (8.20). With respect to y, we have

$$\mathrm{lc}_y f = -z \qquad \text{and} \qquad \mathrm{lc}_y g = x - z$$

They both vanish simultaneously at $(0,0)$ and so we may not apply the extension theorem! In fact, if we try to substitute $x=0$ and $z=0$ into the system (8.20) we get

$$\begin{cases} 1=0 \\ 0=0 \\ 2y^2=0. \end{cases}$$

Obviously, this system has no solutions.

Discriminant

Take a quadratic polynomial $f = ax^2+bx+c$. In secondary school we learned that the roots of f are $x_1 = {}^1\!/_{2a}\cdot(-b+\sqrt{\Delta})$ and $x_2 = {}^1\!/_{2a}\cdot(-b-\sqrt{\Delta})$, where $\Delta := b^2-4ac$ is a certain scalar, called the *discriminant*, associated to f. We may express Δ in terms of roots of f as

$$\Delta = a^2\cdot(x_1-x_2)^2.$$

Our goal is to generalize the notion of the discriminant to polynomials of arbitrary degree. The term was coined in 1851 by J. Sylvester.

Definition 8.3.3. Let $f\in K[x]$ be a polynomial with coefficients in a field K and $L\supset K$ be some algebraically closed field. If $\xi_1,\dots,\xi_n\in L$ are all the roots of f (multiple roots being repeated), then we define the *discriminant* of f as

$$\Delta_f := \operatorname{lc}(f)^{2\cdot\deg f-2}\cdot\prod_{1\le i<j\le n}(\xi_i-\xi_j)^2.$$

An alternative notation is $\operatorname{disc}(f)$.

Recall that the Vandermonde matrix of $\xi_1,\dots,\xi_n$ is

$$V_{\xi_1,\dots,\xi_n} := \begin{pmatrix} 1 & \xi_1 & \xi_1^2 & \cdots & \xi_1^{n-1} \\ 1 & \xi_2 & \xi_2^2 & \cdots & \xi_2^{n-1} \\ \vdots & \vdots & & \vdots & \\ 1 & \xi_n & \xi_n^2 & \cdots & \xi_n^{n-1} \end{pmatrix}$$

We know (see Theorem 3.8.3) that

$$\det V_{\xi_1,\dots,\xi_n} = \prod_{1\le i<j\le n}(\xi_j-\xi_i).$$

Thus we can alternatively define the discriminant as a the determinant of the Vandermonde matrix of its roots.

Observation 8.3.4. $\Delta_f = \operatorname{lc}(f)^{2\deg f-2}\cdot\left(\det V_{\xi_1,\dots,\xi_n}\right)^2$.

An immediate consequence of the definition is that a polynomial has a multiple root if and only if its discriminant vanishes. In other words:

Observation 8.3.5. *Let f be a polynomial, then f is square-free if and only if $\Delta_f \neq 0$.*

In Section 5.3 of Chapter 5 we saw that a polynomial (over a field of characteristic zero) is square-free if and only if it is relatively prime to its derivative. On the other hand, in this chapter we learned that two polynomials are relatively prime if and only if their resultant is nonzero. This let us suspect that the notions of the discriminant and resultant are somehow related. Indeed they are.

Theorem 8.3.6. *Let f be a polynomial defined over a field of characteristic zero, then*

$$\Delta_f = \frac{(-1)^{\frac{1}{2}\deg f\cdot(\deg f-1)}}{\operatorname{lc} f}\cdot \operatorname{res}_x(f,f').$$

Proof. As usual let L be an algebraically closed field containing K. Further assume that $f = f_0 + f_1 x + \cdots + f_n x^n$ and $\xi_1,\ldots,\xi_n \in L$ are all the roots of f. Factor the polynomial f into linear terms

$$f = f_n \cdot \prod_{i=1}^{n}(x-\xi_i),$$

where in presence of multiple roots we allow some terms to be repeated. Compute the derivative of f

$$f' = f_n \cdot \sum_{i=1}^{n}\prod_{\substack{j=1\\ j\neq i}}^{n}(x-\xi_j).$$

Substitute a root ξ_k into it

$$\begin{aligned} f'(\xi_k) &= f_n \cdot \Big(0+\cdots+0+\prod_{\substack{j=1\\ j\neq k}}^{n}(\xi_k-\xi_j)+0+\cdots+0\Big)\\ &= f_n \cdot \prod_{\substack{j=1\\ j\neq k}}^{n}(\xi_k-\xi_j). \end{aligned}$$

Proposition 8.2.3 yields

$$\begin{aligned}\frac{(-1)^{n\cdot(n-1)/2}}{f_n}\cdot \mathrm{res}_x(f,f') &= \frac{(-1)^{n\cdot(n-1)/2}}{f_n}\cdot f_n^{n-1}\cdot\prod_{k=1}^{n} f'(\xi_k)\\ &= (-1)^{n\cdot(n-1)/2}\cdot f_n^{n-2}\cdot\prod_{k=1}^{n}\left(f_n\cdot\prod_{\substack{j=1\\ j\neq k}}^{n}(\xi_k-\xi_j)\right)\\ &= (-1)^{n\cdot(n-1)/2}\cdot f_n^{2n-2}\cdot\prod_{k=1}^{n}\prod_{\substack{j=1\\ j\neq k}}^{n}(\xi_k-\xi_j) = \Delta_f. \qquad \square\end{aligned}$$

It follows immediately from the above theorem and Observation 8.2.4 that the discriminant of a polynomial belongs to the base ring of this polynomial. In particular we have:

Corollary 8.3.7. *If $f\in\mathbb{Z}[x]$ is a square-free polynomial with integer coefficients, then $|\Delta_f|\geq 1$.*

Example 97. Take a quadratic polynomial $f=ax^2+bx+c$, then $f'=2ax+b$. Compute the discriminant of f using Sylvester's method. We have

$$\begin{aligned}\Delta_f &= \frac{(-1)^1}{a}\cdot\det\begin{pmatrix} c & b & a\\ b & 2a & 0\\ 0 & b & 2a\end{pmatrix}\\ &= -\frac{1}{a}\cdot\left(4a^2c+ab^2-2ab^2\right)\\ &= b^2-4ac.\end{aligned}$$

Exercise 23. Using a computer algebra system of your choice, prove the following discriminant formulas:

1. if $f=x^3+px+q$, then $\Delta_f=-4p^3-27q^2$;
2. if $f=x^3+ax^2+bx+c$, then $\Delta_f=a^2b^2-4a^3c-4b^3+18abc-27c^2$;
3. if $f=x^4+px^2+qx+r$, then $\Delta_f=-4p^3q^2+16p^4r-27q^4+144pq^2r-128p^2r^2+256r^3$.

Just by definition it is clear that the determinant Δ_f of f is a symmetric polynomial in the roots of f. It follows from Viète's formulæ that Δ_f is a polynomial in coefficients of f. For example, the discriminant $\Delta=b^2-4ac$ of a quadratic polynomial is a homogeneous polynomial of degree 2 in variables a,b,c, hence an element of $\mathbb{Z}[a,b,c]$. Unfortunately, for a generic polynomial f, the number of terms of Δ_f grows exponentially with the degree of f. Therefore, for high degree polynomials such closed-form formulas, as the ones presented in the previous exercise, do not have much practical value. At the moment of writing, the closed-form discriminant formulas were known

(see [40]) for polynomials up to degree 17, where the Δ_f for $\deg f = 17$ consists of... 21 976 689 397 terms!

Remark 66. Any irreducible algebraic curve of degree 3 over $\mathbb{C}$ can be transformed by a certain change of variables (formally speaking: by a projective map), into a so called *Weierstraß*[3] *normal form* (for details see e.g. [29, Chapter 15]):

$$y^2 = x^3 + px + q.$$

One can detect that the curve does not have any *singular points* (cusps or self-intersections) computing the discriminant of the right-hand-side. The curve is *non-singular* (i.e. does not have singular points) if and only if the discriminant is nonzero.

At secondary school we learned that a discriminant $\Delta = b^2 - 4ac$ of a quadratic polynomial $f = ax^2 + bx + c$ is null if and only if f has a double root (necessarily real by Observation 6.1.5), Δ is positive if f has two distinct real roots and negative if it has no real roots (i.e. it has two conjugated complex zeros). We have already generalized the first of this properties to polynomials of arbitrary degrees. The next proposition describes the correspondence between the sign of the discriminant and a number of real roots.

Proposition 8.3.8. *Let f be a square-free polynomial with real coefficients. Denote the number of real roots of f by s, then*

- $\Delta_f > 0$ *if and only if* $\deg f \equiv s \pmod 4$*;*
- $\Delta_f < 0$ *if and only if* $\deg f \equiv s + 2 \pmod 4$.

Proof. Let $\xi_1, \dots, \xi_s \in \mathbb{R}$ be all the real roots of f and $(\zeta_1, \overline{\zeta}_1), \dots, (\zeta_k, \overline{\zeta}_k)$ be all the pairs of conjugated non-real complex roots, where $s + 2k = \deg f$. Observe that for every i, the difference $\zeta_i - \overline{\zeta}_i$ is a purely imaginary number, hence

$$\left(\zeta_1 - \overline{\zeta}_1\right)^2 \cdots \left(\zeta_k - \overline{\zeta}_k\right)^2 = (-1)^k \cdot \text{some positive real number.}$$

At the same time, for every $i < j$, $(\xi_i - \xi_j)^2$ is strictly positive, as a square of a real number. Likewise, the products

$$(\xi_i - \zeta_j)(\xi_i - \overline{\zeta}_j) = \big((\xi_i - \operatorname{re}\zeta_j) - i \cdot \operatorname{im}\zeta_j\big)\big((\xi_i - \operatorname{re}\zeta_j) + i \cdot \operatorname{im}\zeta_j\big)$$

and

$$(\zeta_i - \zeta_j)(\zeta_i - \overline{\zeta}_j)(\overline{\zeta}_i - \zeta_j)(\overline{\zeta}_i - \overline{\zeta}_j)$$

are sums of squares of reals and so are positive, too. All in all

$$\operatorname{sgn}\Delta_f = \operatorname{sgn}\prod_{i=1}^{k}(\zeta_i - \overline{\zeta}_i)^2 = (-1)^k$$

and it follows that Δ_f is negative if and only if k is odd, which means that $s \equiv \deg f - 2$. □

[3] Karl Theodor Wilhelm Weierstraß (born 31 October 1815, died 19 February 1897) was a German mathematician.

Example 98. If f is a real cubic polynomial, then:

- $\Delta_f > 0$ if and only if f has 3 distinct real roots;
- $\Delta_f = 0$ if and only if f has a multiple root (either one triple root or one double root and one simple root);
- $\Delta_f < 0$ if and only if f has a unique real root and a pair of conjugated non-real complex roots.

We shall now prove some further properties of the discriminant.

Proposition 8.3.9. *If $f, g \in K[x]$ are two nonzero polynomials, then*

$$\Delta_{fg} = \Delta_f \cdot \Delta_g \cdot \operatorname{res}_x(f,g)^2.$$

Proof. Write f and g as products of linear terms over an algebraic closed field L containing K:

$$f = f_m \cdot (x-\xi_1)\cdots(x-\xi_m), \qquad g = g_n \cdot (x-\zeta_1)\cdots(x-\zeta_n).$$

By Theorem 8.3.6:

$$\begin{aligned}\Delta_{fg} &= (f_m g_n)^{2(n+m)-2} \cdot \prod_{1\le i<j\le m}(\xi_i-\xi_j)^2 \cdot \prod_{1\le k<l\le n}(\zeta_k-\zeta_l)^2 \cdot \prod_{\substack{1\le i\le m\\ 1\le k\le n}}(\xi_i-\zeta_k)^2 \\ &= \left(f_m^{2m-2}\cdot\prod_{1\le i<j\le m}(\xi_i-\xi_j)^2\right)\cdot\left(g_n^{2n-2}\cdot\prod_{1\le k<l\le n}(\zeta_k-\zeta_l)^2\right)\cdot\left(f_m^n\cdot g_n^m\cdot\prod_{\substack{1\le i\le m\\ 1\le k\le n}}(\xi_i-\zeta_k)\right)^2 \\ &= \Delta_f\cdot\Delta_g\cdot\operatorname{res}_x(f,g)^2.\end{aligned}$$

□

Recall (see Definition 1.1.1) that if $f = f_0 + f_1 x + \cdots + f_n x^n$ is a polynomial, then $\operatorname{rev}(f) := f_n + f_{n-1}x + \cdots + f_0 x^n$ is called its reciprocal polynomial.

Proposition 8.3.10. *Let f be a polynomial. If its constant term is nonzero, then $\Delta_f = \Delta_{\operatorname{rev}(f)}$.*

Proof. The proof is very similar to that of Proposition 8.2.8. Let $\xi_1, \ldots, \xi_n$ be all the roots of f. By Observation 1.1.2 $\operatorname{rev}(f) = x^n \cdot f(1/x)$, hence $1/\xi_1, \ldots, 1/\xi_n$ are all the roots of $\operatorname{rev}(f)$. Theorem 8.3.6 yields

$$\begin{aligned}\Delta_{\operatorname{rev}(f)} &= f_0^{2n-2}\cdot\prod_{1\le i<j\le n}\left(\frac{1}{\xi_i}-\frac{1}{\xi_j}\right)^2 \\ &= f_0^{2n-2}\cdot\prod_{1\le i<j\le n}\frac{(\xi_j-\xi_i)^2}{\xi_i^2\xi_j^2} \\ &= \frac{f_0^{2n-2}}{\left(\prod_{k=1}^n \xi_k\right)^2}\cdot\prod_{1\le i<j\le n}(\xi_i-\xi_j)^2\end{aligned}$$

Now, $\xi_1 \cdots \xi_m = {}^{f_0}/_{f_n}$ by Viète's formulas, hence the above expression simplifies to

$$= f_n^{2n-2} \cdot \prod_{1 \le i < j \le n} (\xi_i - \xi_j)^2 = \Delta_f. \qquad \square$$

Distance between roots of a univariate polynomial

Let $f \in K[x]$ be a univariate polynomial with coefficients in a subfield K of $\mathbb{C}$ and let $\xi_1 \dots, \xi_n$ be all its roots, where the root of multiplicity k is repeated k times. Recall (see Section 6.4) that we denote

$$\operatorname{sep}(f) = \min\{|\xi_i - \xi_j| \mid i \neq j\},$$
$$\operatorname{rsep}(f) = \min\{|\xi_i - \xi_j| \mid \xi_i, \xi_j \in \mathbb{R},\ i \neq j\}.$$

respectively: the minimal distance between complex and real roots of f. In Chapter 6 we presented explicit bounds for sep and rsep. As we observed in Example , the bounds are very pessimistic in general. Here we develop a method for obtaining a very tight estimate. The next proposition will be the main ingredient of the method.

missing

Proposition 8.3.11. *Let $f \in \mathbb{C}[x]$ be a polynomial of degree $\deg f = n \ge 2$ and $\xi_1, \dots, \xi_n \in \mathbb{C}$ be all its roots. Set*

$$h := \operatorname{rad}\left(\frac{\operatorname{res}_y\big(f(y-x), f(y)\big)}{x^n} \right) \in \mathbb{C}[x]$$

for some new variable y. Then the polynomial h is even, square-free and its roots are precisely the differences between distinct roots of f, i.e.

$$h(\delta) = 0 \iff \delta = \xi_i - \xi_j \quad \text{for some } i \neq j.$$

Proof. The polynomial f factors over $\mathbb{C}$ into a product of linear terms:

$$f = \operatorname{lc}(f) \cdot \prod_{1 \le i \le n} (x - \xi_i).$$

Denote $r := \operatorname{res}_y\big(f(y-x), f(y)\big)$. Directly from the definition of the resultant we infer

$$r = \operatorname{lc}(f)^{2n} \cdot \prod_{1 \le i,j \le n} \big(x - (\xi_i - \xi_j)\big) = \operatorname{lc}(f)^{2n} \cdot x^n \cdot \prod_{i \neq j} \big(x - (xi_i - \xi_j)\big).$$

Let $\delta_1, \dots, \delta_k \in \mathbb{C}$ be all the differences between roots of f, i.e.

$$\{\delta_1, \dots, \delta_k\} = \{\xi_i - \xi_j \mid i \neq j\}.$$

Further let m_j denote the number of pairs (ξ_i, ξ_j) such that the difference $\xi_i - \xi_j$ equals δ_j. Therefore we have

$$r = \operatorname{lc}(f)^{2n} \cdot x^n \cdot (x - \delta_1)^{m_1} \cdots (x - \delta_k)^{m_k}.$$

It follows that $\delta_1, \dots, \delta_k$ are all the roots of $\operatorname{rad}({}^{r}/_{x^n})$. $\square$

Let $h = h_0 + h_1 x + \cdots + h_k x^k$ be the polynomial constructed by the means of the above proposition. Recall (see Definition 1.1.1) that the polynomial $\mathrm{rev}(f) = h_k + h_{k-1}x + \cdots + h_0 x^k$ is called the reciprocal of h. Observation 6.3.1 says that a lower bound for the absolute value of non-zero roots of h is obtained by inverting an a upper bound for roots of $\mathrm{rev}(h)$. Combining this fact with the previous proposition we have:

Corollary 8.3.12. *Let f, h be as in Proposition 8.3.11.*

1. *If M is an upper bound for the absolute values of complex roots of* $\mathrm{rev}(h)$*, then*
$$\mathrm{sep}(f) \geq \frac{1}{M}.$$
2. *If M is an upper bound for the absolute values of real roots of* $\mathrm{rev}(h)$*, then*
$$\mathrm{rsep}(f) \geq \frac{1}{M}.$$

The corollary can be directly converted into an algorithm.

Algorithm 8.3.13. *Let $M : \mathbb{Z}[x] \to (0, \infty)$ be a subprocedure computing an upper bound for absolute values of complex* (*respectively real*) *roots of a polynomial. Given a polynomial $f \in \mathbb{Z}[x]$ of degree $n \geq 2$ with integral coefficients this algorithm computes a lower bound for* $\mathrm{sep}(f)$ (*respectively for* $\mathrm{rsep}(f)$).

1. *Using methods described in Section 8.2, compute the resultant $r := \mathrm{res}_y\big(f(y-x), f(y)\big)$.*
2. *Using Observation 5.3.4 compute the radical $h := \mathrm{rad}\big({}^{r}/_{x^n}\big)$.*
3. *Get an upper bound $m := M\big(\mathrm{rev}(h)\big)$ for the absolute values of roots of the reciprocal of h.*
4. *Return ${}^{1}/_{m}$.*

Example 99. Take the same polynomial $f = x^5 - x^3 - 1000x^2 + 10 \in \mathbb{Z}[x]$ that we have already encountered in Example 74. Recall that $\mathrm{sep}(f) \approx 0.20$. We have

$$f(y-x) = y^5 - 5xy^4 + \big(10x^2 - 1\big)y^3 + \big(-10x^3 + 3x - 1000\big)y^2 + \big(5x^4 - 3x^2 + 2000x\big)y - x^5 + x^3 - 100$$

Compute the resultant $r = \mathrm{res}_y\big(f(y-x), f(y)\big)$ using e.g. Collins' algorithm (Algorithm 8.2.46):

$$r = x^5 \cdot (x^{20} - 10x^{18} + 39x^{16} + 24999920x^{14} - 93999905x^{12} + 122062434x^{10} - 53000054574975x^8 + 94$$

The polynomial $h := {}^{r}/_{x^5}$ is radical and its reciprocal equals

$$\mathrm{rev}(h) = 5.$$

We may know compute the Fujiwara upper bound (see Proposition 6.3.9) for roots of $\mathrm{rev}\, h$:

$$M = 2 \cdot \max\Big\{|1|^{\frac{1}{20}}, |-10|^{\frac{1}{18}}, |39|^{\frac{1}{16}}, |24999920|^{\frac{1}{14}}, |-93999905|^{\frac{1}{12}}, |122062434|^{\frac{1}{10}}, |-53000054574975$$

It follows that $\mathrm{sep}(f) \geq {}^{1}/_{M} \approx 0.10$. This is a much tighter bound than the ones we obtained in Chapter 6.

Implicitization of parametric curves

Another application of the resultant is to find an implicit formula for a rational parametric curve. Suppose we are given a pair of rational functions f/h and g/h in a single variable t. We shall interpret them respectively as an abscissa and ordinate of a point $P = P(t)$ on a parametric curve C:

$$C: \ t \mapsto P(t) = \Big(\tfrac{f(t)}{h(t)}, \tfrac{g(t)}{h(t)}\Big).$$

Our goal is to find an implicit equation of C. We are looking for a bivariate polynomial $F \in \mathbb{C}[x,y]$ such that $F\big(P(t)\big) \equiv 0$. In other words, we want to express C as an *algebraic curve*

$$C = \big\{(x,y) \in \mathbb{C}^2 \,|\, F(x,y) = 0\big\}.$$

Without loss of generality, we may assume that f, g, h are relatively prime. Since $f(t)/h(t)$ is the abscissa and $g(t)/h(t)$ is the ordinate of a point $P(t) = (x,y)$, we can write

$$\begin{cases} x \cdot h(t) - f(t) = 0 \\ y \cdot h(t) - g(t) = 0. \end{cases}$$

What we want to do is to eliminate variable t from the above system. Treat $x \cdot h - f$ and $y \cdot h - g$ as polynomials from $\mathbb{C}[x,y,t]$. Theorem 8.3.1 asserts that the resultant of these two polynomial with respect to t vanishes on all points of the curve. Hence

$$F = \operatorname{res}_t(x \cdot h - f, y \cdot h - g)$$

is the desired polynomial.

Example 100. Write a unit circle as a parametric curve $(x,y) = (\cos\varphi, \sin\varphi)$. Now basic trigonometry let us express sine and cosine using tangent:

$$\cos\varphi = \frac{1-\tan^2\varphi}{1+\tan^2\varphi}, \qquad \sin\varphi = \frac{2\tan\varphi}{1+\tan^2\varphi}.$$

The values of tangent spans the entire real line, hence setting $t := \tan\varphi$ we obtain a rational parametrization of the unit circle

$$t \mapsto \Big(\frac{-t^2+1}{t^2+1}, \frac{2t}{t^2+1}\Big).$$

The polynomials $f = -t^2+1$, $g = 2t$ and $h = t^2+1$ are relatively prime. Compute

$$\begin{aligned} F &= \operatorname{res}_t\big((x+1)\,t^2 + x - 1, y\,t^2 - 2t + y\big) \\ &= (-1)^4 \cdot \det \begin{pmatrix} x-1 & 0 & x+1 & 0 \\ 0 & x-1 & 0 & x+1 \\ y & -2 & y & 0 \\ 0 & y & -2 & y \end{pmatrix} \\ &= 4 \cdot (x^2 + y^2 - 1), \end{aligned}$$

to get the well known equation of the unit circle $x^2 + y^2 - 1 = 0$.

Minimal polynomials of algebraic numbers

Resultants can be used to find minimal polynomials of algebraic numbers or, in fact, more generally minimal polynomials of algebraic elements over any base field. Let $L \supset K$ be two fields. Recall that an element $\alpha \in L$ is said to be *algebraic over* K (see Section 2.5), if it is a root of a nonzero polynomial with coefficients in K. Complex numbers algebraic over $\mathbb{Q}$ are called *algebraic numbers*. The polynomial ring $K[x]$, being euclidean, is a principal ideal domain, hence the ideal

$$I_\alpha = \langle f \in K[x] \,|\, f(\alpha) = 0 \rangle,$$

consisting of all ideals vanishing at α has a unique monic generator f_α, called the *minimal polynomial* of α. The minimal polynomial is necessarily irreducible.

Remark 67. All the results in this section are formulated for algebraic numbers (i.e. over $\mathbb{Q}$), nevertheless they hold over any field K of characteristic zero. We restrict ourselves just to $\mathbb{Q}$ for two reasons. First of all, to make the theory easier for a reader to digest. Secondly, because this is probably the most important case, anyway.

Theorem 8.3.14. *Let $\alpha, \beta \in \mathbb{C}$ be two algebraic numbers and $f_\alpha, g_\beta \in \mathbb{Q}[x]$ be their minimal polynomials.*

1. *The minimal polynomial of $\alpha + \beta$ is an irreducible factor of*
$$\operatorname{res}_t\big(f_\alpha(x-t), g_\beta(t)\big).$$
2. *The minimal polynomial of $\alpha - \beta$ is an irreducible factor of*
$$\operatorname{res}_t\big(f_\alpha(x+t), g_\beta(t)\big).$$
3. *The minimal polynomial of $\alpha \cdot \beta$ is an irreducible factor of*
$$\operatorname{res}_t\big(t^{\deg f} f_\alpha(\tfrac{x}{t}), g_\beta(t)\big).$$
4. *Assume that $\beta \neq 0$. The minimal polynomial of ${}^{\alpha}/_{\beta}$ is an irreducible factor of*
$$\operatorname{res}_t\big(f_\alpha(x \cdot t), g_\beta(t)\big).$$

Proof. We prove just the first point. The remaining ones are fully analogous. Denote the degree of f_α by m and the degree of g_β by n. Let $\alpha = \alpha_1, \ldots, \alpha_m \in \mathbb{C}$ be all the roots of f_α. They are all distinct, since f_α, being irreducible, is square-free. Likewise, let $\beta = \beta_1, \ldots, \beta_n \in \mathbb{C}$ be all the roots of g_β. The polynomial f_α factors over $\mathbb{C}$ into a product of linear polynomials $f_\alpha = (x - \alpha_1) \cdots (x - \alpha_m)$. It follows that

$$f_\alpha(x-t) = (-1)^n \cdot \big(t - (x - \alpha_1)\big) \cdots \big(t - (x - \alpha_m)\big).$$

Compute the resultant

$$h = \operatorname{res}_t\big(f_\alpha(x-t), g_\beta(t)\big) = (-1)^{mn} \cdot \prod_{\substack{1 \le i \le m \\ 1 \le j \le n}} (x - \alpha_i - \beta_j).$$

It is now clear that $h(\alpha+\beta)=0$. Thus, if h itself is irreducible, then $(-1)^{mn}\cdot h$ is the minimal polynomial of $\alpha+\beta$. On the other hand, if h is irreducible, then the minimal polynomial of $\alpha+\beta$ is one of its factors. □

For the sake of clarity of an exposition, we rephrase the first point of the previous theorem in form of an algorithm. We let the reader do the same with the other points.

Algorithm 8.3.15. *Let $\alpha,\beta\in\mathbb{C}$ be two algebraic numbers, each represented as a pair consisting of the minimal polynomial (denoted respectively f_α and g_β) and a floating point approximation of one of its roots (denoted $\tilde{\alpha}$ and $\tilde{\beta}$). This algorithm computes the minimal polynomial of $\alpha+\beta$.*

1. *Compute the resultant*
$$r := \operatorname{res}_t\big(f_\alpha(x-t), g_\beta(t)\big)$$
using any of the methods described in this chapter;
2. *factor r into a product $r = c\cdot p_1^{k_1}\cdots p_l^{k_l}$ of monic irreducible polynomials using methods described in Chapter 5;*
3. *find $i\in\{1,\dots,l\}$ such that $|p_i(\tilde{\alpha}+\tilde{\beta})|$ is minimal;*
4. *return p_i.*

Example 101. Take $\alpha=\sqrt{2}$ and $\beta=\sqrt{3}$. Then

$$f_\alpha = x^2-2, \qquad \tilde{\alpha}=1.4142, \qquad g_\beta = x^2-3, \qquad \tilde{\beta}=1.7320.$$

Compute the resultant using Algorithm 8.2.31:

$$\begin{aligned} h &= \operatorname{res}_t\big(t^2-2xt+x^2-2, t^2-3\big)\\ &= \det\left(\begin{pmatrix} 0 & -x^2+2\\ 1 & 2x\end{pmatrix}^2 - 2\begin{pmatrix}1&0\\0&1\end{pmatrix}\right)\\ &= x^4-10x^2+1. \end{aligned}$$

This polynomial is monic and irreducible and so it is the minimal polynomial of $\sqrt{3}+\sqrt{2}$.

Example 102. Consider a slightly more advanced example. Let $\alpha=2\sqrt{3}+3\sqrt{2}+1$ and $\beta=\sqrt{6}$. We have

$$f_\alpha = x^4-4x^3-54x^2+116x-23, \qquad \tilde{\alpha}=8.7067, \qquad g_\beta=x^2-6, \qquad \tilde{\beta}=2.4495.$$

We again compute the resultant

$$\begin{aligned} h &= \operatorname{res}_t\big(t^4+(-4x+4)t^3+\big(6x^2-12x-54\big)t^2+\big(-4x^3+12x^2+108x-116\big)t+x^4-4x^3-54x^2\\ &= x^8-8x^7-116x^6+808x^5+2518x^4-15608x^3-15956x^2+65368x+45937. \end{aligned}$$

This times however h is reducible. It factors into two quartics:

$$h = (x^4 - 4x^3 - 66x^2 - 148x - 71) \cdot (x^4 - 4x^3 - 66x^2 + 428x - 647).$$

Denote the two factors by h_1 and h_2. We need to recognize which one is the minimal polynomial of $\alpha + \beta$. To this end we compute

$$h_1(\tilde{\alpha} + \tilde{\beta}) \approx 0.012085, \qquad h_2(\tilde{\alpha} + \tilde{\beta}) \approx 5850.0.$$

It follows that the minimal polynomial of $\sqrt{6} + 2\sqrt{3} + 3\sqrt{2} + 1$ is h_1.

Let $\alpha \in \mathbb{C}$ be an algebraic number with the minimal polynomial f_α. Recall (see Section 2.5 of Chapter 1) that elements of $\mathbb{Q}(\alpha)$ are naturally represented in form $h(\alpha)$, where $h \in \mathbb{Q}[x]$ is a polynomial of degree strictly less that $\deg f_\alpha$. Since every element of this form is again an algebraic number, we may wish to find its minimal polynomial.

Proposition 8.3.16. *Let $\alpha \in \mathbb{C}$ be an algebraic number with the minimal polynomial f_α and let $L = \mathbb{Q}(\alpha)$ be the number field generated by α. If $\beta \in L$ is an element of a form $\beta = h(\alpha)$ for some polynomial $h \in \mathbb{Q}[x]$, then the minimal polynomial of β is a monic irreducible factor of*

$$\operatorname{res}_t\big(x - h(t), f_\alpha(t)\big).$$

Proof. Consider a system of polynomial equations

$$\begin{cases} x - h(t) = 0 \\ f_\alpha(t) = 0 \end{cases}$$

in variables x and t. A pair (β, α) is clearly one of solution to this system. compute the resultant $r := \operatorname{res}_t\big(x - h(t), f_\alpha(t)\big)$. Theorem 8.3.1 asserts that $r(\beta) = 0$, therefore the minimal polynomial of β must divide r. □

Example 103. Let $\alpha \in \mathbb{C}$ be a root of $f = x^4 - 10x^2 + 1$ and $\beta = \frac{1}{2}\alpha^3 - \frac{9}{2}\alpha$. Compute the minimal polynomial of β using Proposition 8.3.16. We have

$$\begin{aligned} r &= \operatorname{res}_t\big(-\tfrac{1}{2}t^3 + \tfrac{9}{2}t + x, t^4 - 10t^2 + 1\big) \\ &= x^4 - 4x^2 + 4 \\ &= (x^2 - 2)^2. \end{aligned}$$

Thus, the minimal polynomial of β is $x^2 - 2$ or in other words $\beta = \sqrt{2}$.

Finally, we show how to use resultants to construct minimal (absolute) polynomials in relative field extensions. Let $\alpha \in \mathbb{C}$ be an algebraic number and $f_\alpha \in \mathbb{Q}[y]$ its minimal polynomial. Further, let $K = \mathbb{Q}(\alpha)$ be the algebraic number field generated by α, i.e. the minimal extension of $\mathbb{Q}$ containing α. Denote the degree of f_α (equivalently the degree of the field extension $K/\mathbb{Q}$) by m.

Assume that $\beta \in \mathbb{C}$ is a root of a (nonzero) polynomial $g = g_0 + g_1 x + \cdots + g_n x^n$ with coefficients in K. Since $K \cong \mathbb{Q}[y]/\langle f_\alpha \rangle$, the elements $g_0, \ldots, g_n$ are classes of polynomials modulo f_α. We can think of them as polynomials in α of degree strictly less than m:

$$g_i = g_i(\alpha) = g_{i,0} + g_{i,1} \cdot \alpha + \cdots + g_{i,m-1} \cdot \alpha^{m-1},$$

where $g_{i,j} \in \mathbb{Q}$. Let $\hat{g}_i := g_i(y) \in \mathbb{Q}[y]$, for $0 \leq i \leq n$, be the corresponding polynomials—formally speaking, these are preimages of $g_0, \ldots, g_n$ with respect to the canonical epimorphism from $\mathbb{Q}[y]$ onto $\mathbb{Q}[y]/\langle f_\alpha \rangle$. Define a bivariate polynomial

$$G := \sum_{i=1}^{n} \hat{g}_i(y) \cdot x^i \in \mathbb{Q}[x, y]. \tag{8.21}$$

This is nothing else, but just the polynomial g where all the occurrences of α are replaced by the indeterminate y. Consider a system of polynomial equations

$$\begin{cases} f_\alpha(y) = 0 \\ G(x, y) = 0 \end{cases} \tag{8.22}$$

in x, y and denote by h the resultant $\operatorname{res}_y(f_\alpha, G)$. The pair (β, α) is one of the solutions, therefore $h(\beta) = 0$ by Theorem 8.3.1. It follows that we have just proved the following result:

Lemma 8.3.17. *Keep the above notation. The number β is algebraic over $\mathbb{Q}$ and its minimal polynomial is an irreducible factor of*

$$h = \operatorname{res}_y\big(f_\alpha(y), G(x, y)\big).$$

This lemma is very similar to Theorem 8.3.14 and Proposition 8.3.16, but in this case we can actually show more.

Lemma 8.3.18. *The polynomial h is a power of some irreducible polynomial.*

Proof. The proof requires a reader to be at least rudimentarily familiar with field theory or algebraic number theory. We know by the previous lemma that the minimal polynomial of β is an irreducible factor of h, denote it by p. Suppose a contrario that p is not the unique irreducible factor of h, thus there is another monic irreducible polynomial q that divides h. Let $\gamma \in \mathbb{C}$ be a root of q.

The polynomial G, treated as a univariate polynomial in y over $\mathbb{Q}[x]$, is monic since its leading coefficient is the same as that of G, which is minimal polynomial of β over K. The extension theorem (cf. Theorem 8.3.2) asserts that there is $\alpha_1 \in \mathbb{C}$ such that (γ, α_1) is a solution to Eq. (8.22). In particular $f_\alpha(\alpha_1) = 0$ and so α_1 is a conjugate of α. Therefore $K_1 := \mathbb{Q}(\alpha_1)$ is isomorphic to K. Denote by $\varphi : K \xrightarrow{\sim} K_1$ an isomorphism of K and K_1 mapping α to α_1. The restriction $\varphi\big|_{\mathbb{Q}}$ is an identity and φ can be extended to an isomorphism of polynomial ring $K[x] \xrightarrow{\sim} K_1[x]$ defined by the formula

$$a_0 + a_1 x + \cdots + a_k x^k \mapsto \varphi(a_0) + \varphi(a_1) x + \cdots + \varphi(a_k) x^k.$$

Abusing the notation slightly, denote this extension by φ again.

We have $g = G(x, \alpha)$, let $g_1 := \varphi(g) = G(x, \alpha_1)$. The polynomial g is monic and irreducible and so is its image g_1. Since $g_1(\gamma) = G(\gamma, \alpha_1) = 0$, g_1 is the minimal polynomial of γ over K_1. It follows that the isomorphism $\varphi : K \xrightarrow{\sim} K_1$ can be extended to an isomorphism of

$$K(\beta) = {}^{K[x]}\!/_{\langle g \rangle} \qquad \text{and} \qquad K_1(\gamma) = {}^{K_1[x]}\!/_{\langle g_1 \rangle}.$$

We claim that this implies that γ is a root of p (i.e. that β and γ are conjugated). Indeed, let $\Phi : K(\beta) \xrightarrow{\sim} K_1(\gamma)$ be an isomorphism defined by the formula $\Phi\big(a + \langle g \rangle\big) = \varphi(a) + \langle g_1 \rangle$. Then Φ restricted to $\mathbb{Q}$ is an identity and $\Phi(\beta) = \gamma$. Hence

$$0 = \Phi\big(p(\beta)\big) = \Phi\big(p_0 + p_1\beta + \cdots + p_{k-1}\beta^{k-1} + \beta^k\big) = p_0 + p_1\gamma + \cdots + p_{k-1}\gamma^{k-1} + \gamma^k = p(\gamma).$$

Thus γ is a root of p as claimed. This implies that q, being the minimal polynomial of γ, divides p. But p and q are both monic and irreducible, therefore $q = p$. □

Recall that the radical of a polynomial h (see Section 5.3 of Chapter 5) is the square-free polynomial $\operatorname{rad} h$ having the same roots as h but all of multiplicity 1. Combining the last two lemmas, we arrive at the following result.

Theorem 8.3.19. *Let $\alpha \in \mathbb{C}$ be an algebraic number with the minimal polynomial $f_\alpha \in \mathbb{Q}[y]$. Denote $K := \mathbb{Q}(\alpha)$ and let $g \in K[x]$ be a minimal polynomial (over K) of another algebraic number $\beta \in \mathbb{C}$. Then the minimal polynomial of β over $\mathbb{Q}$ equal*

$$\operatorname{rad}\Big(\operatorname{res}_y\big(f_\alpha(y), G(x, y)\big)\Big),$$

where G is the polynomial defined in Eq. (8.21).

For the sake of completeness we rewrite the theorem in form of an algorithm.

Algorithm 8.3.20. *Let $K = \mathbb{Q}(\alpha)$ be a number field specified by the minimal polynomial f_α of its generator α and let $g \in K[x]$ be a monic irreducible polynomial over K. This algorithm compute an (absolute) minimal polynomial g_β over $\mathbb{Q}$ of a root β of g.*

1. *Construct a bivariate polynomial $G \in \mathbb{Q}[x, y]$ replacing each occurrence of α in g by x;*
2. *using one of the algorithm presented in this chapter compute the resultant*

$$h = \operatorname{res}_y\big(f_\alpha(y), G(x, y)\big)$$

3. *compute the radical of h using Observation 5.3.4:*

$$\operatorname{rad} h = \frac{h}{\gcd(h, h')};$$

4. *return* $\operatorname{rad} h$.

Example 104. Take a number field $K = \mathbb{Q}(\alpha)$, where α is a root of $f_\alpha = y^4 - 2y^2 - 1 \in \mathbb{Q}[y]$ an let $\beta \in \mathbb{C}$ be a root of $g = x^2 - \alpha^2 - 1$. We shall compute the minimal polynomial of β over $\mathbb{Q}$. To this end construct the polynomial G by replacing α by a new variable x, hence $G = x^2 - y^2 - 1$. Compute the resultant using Bézout formula:

$$\begin{aligned} h &= \operatorname{res}_y\left(y^4 - 2y^2 - 1, x^2 - y^2 - 1\right) \\ &= \det \begin{pmatrix} 0 & -2x^2+1 & 0 & x^2-1 \\ -2x^2+1 & 0 & x^2-1 & 0 \\ 0 & x^2-1 & 0 & -1 \\ x^2-1 & 0 & -1 & 0 \end{pmatrix} \\ &= x^8 - 8x^6 + 20x^4 - 16x^2 + 4 = (x^4 - 4x^2 + 2)^2. \end{aligned}$$

Thus, $\operatorname{rad}(x^8 - 8x^6 + 20x^4 - 16x^2 + 4) = x^4 - 4x^2 + 2$ is the minimal polynomial of β.

8.4 Subresultants

Combine with subsection on Euclidean algorithm?

Computing a resultant using Corollary 8.2.36, we have to construct a polynomial (pseudo-)remainder sequence. As seen in Chapter 1, this can lead to a tremendous growth of coefficients in intermediate steps. Fortunately, we are now equipped with tools to effectively deal with this problem.

Let $f, g \in K[x]$ be two polynomials of degrees respectively m and n. Recall that Sylvester's formula for the resultant was derived by considering a linear map $\Phi_{f,g}$ on the Cartesian product $K[x]_{n-1} \times K[x]_{m-1}$ sending a pair of polynomials a, b to their linear combination $f \cdot a + g \cdot b \in K[x]_{m+n-1}$, where as usually, $K[x]_k$ denotes the space of polynomials of degree at most k. We shall extend this idea a bit. The linear space $K[x]_k$ has a basis $\{1, x \ldots, x^k\}$. It follows that the quotient space $K[x]^k_j := {}^{K[x]_k}/_{K[x]_j}$, for $j < h$, can be equipped with a basis $\{x^{j+1}, x^{j+2}, \ldots, x^k\}$. This space is isomorphic to $K[x]_{k-j}$. Denote by π^k_j the canonical epimorphism $K[x] \to K[x]^k_j$. Consider a linear map $\Phi^j_{f,g} : K[x]_{n-j-1} \times K[x]_{m-j-1} \to K[x]_{n+m-j}$ defined literally by the same formula as $\Phi_{f,g}$, namely

$$\Phi^j_{f,g}(a, b) := f \cdot a + g \cdot b.$$

The matrix of this homomorphism, with respect to the power bases of all spaces involved, is

$$M := \left(\begin{array}{ccc:ccc} f_0 & & & g_0 & & \\ \vdots & \ddots & & \vdots & \ddots & \\ f_m & & f_0 & g_n & & g_0 \\ & \ddots & \vdots & & \ddots & \vdots \\ & & f_m & & & g_n \end{array}\right).$$

$$\underbrace{\qquad\qquad}_{n-j \text{ columns}} \quad \underbrace{\qquad\qquad}_{m-j \text{ columns}}$$

It has $m+n-2j$ columns and $m+n-j$ rows. It is obtained from the transpose of the Sylvester matrix $\mathrm{Syl}_x(f,g)$ by deleting:

- first j columns of coefficients of f;
- first j columns of coefficients of g;
- first j rows, that anyway contain contain only zeros, after the previous two operations.

The dimension of the domain of $\Phi^j_{f,g}$ is $n+m-2j$ and in general is smaller than the dimension of the co-domain which is $n+m-j$. In particular, for $j>0$ the map $\Phi^j_{f,g}$ cannot be surjective. To remedy this, we shall compose $\Phi^j_{f,g}$ with the canonical epimorphism of $K[x]_{m+n-j-1}$ onto $K[x]^{m+n-j-1}_{j-1}$. The resulting matrix, denoted by $S^j_j(f,g)$ (the use of a double index j will become clear below), is obtained from M one by deleting the first j rows, hence

$$S^j_j(f,g) := \left(\begin{array}{ccc:ccc} f_j & \cdots & f_{2j-n+1} & g_j & \cdots & g_{2j-m+1} \\ \vdots & & \vdots & \vdots & & \vdots \\ f_m & & \vdots & g_n & & \vdots \\ & \ddots & \vdots & & \ddots & \vdots \\ & & f_m & & & g_n \end{array}\right).$$

As always, polynomial coefficients with negative indices are null, by definition.

Definition 8.4.1. The j-th *subresultant coefficient* of f, g is

$$\mathrm{sr}_j(f,g) := (-1)^{d-j}\cdot\det\big(S^j_j(f,g)\big),$$

where $d := {}^1\!/_2\cdot(n+m)\cdot(n+m-1)$.

Observation 8.4.2. $\mathrm{res}_x(f,g) = \mathrm{sr}_0(f,g)$.

It is clear that $\mathrm{sr}_j(f,g)\neq 0$ if and only if the composition $\pi^{n+m-j-1}_{j-1}\circ\Phi^j_{f,g}$ is an isomorphism. This can be rephrased as

Observation 8.4.3. *The subresultant coefficient* $\mathrm{sr}_j(f,g)$ *is zero if and only if there are polynomials* a,b *such that*

$$\deg(fa+gb)<1 \qquad \textit{and} \qquad \deg a<\deg g-j,\quad \deg b<\deg f-j.$$

Example 105. Let $f=3x^2-4x+1$ and $g=x^3+2x^2-5$ be two polynomials with integer coefficients, then

$$\mathrm{sr}_1(f,g) = (-1)^9\cdot\det\begin{pmatrix} -4 & 1 & 0 \\ 3 & -4 & 2 \\ 0 & 3 & 1 \end{pmatrix} = -37$$
$$\mathrm{sr}_2(f,g) = (-1)^8\cdot\det\begin{pmatrix} 3 \end{pmatrix} = 3.$$

We shall need the following rather technical lemma.

Lemma 8.4.4. *Let A be an invertible matrix (with coefficients in some ring) and v be a vector. Then the product $v\cdot^{-1}$ is a vector of a form*

$$\left[\dots, \frac{\det B_i}{\det A_i}, \dots\right],$$

where B_i is a matrix obtained by substituting v for the i-th row of A.

The proof of this lemma follows by inspection of the closed-form formula for the inverse matrix. We let the reader fill in the necessary details.

Fix $j \geq 0$ and assume that $\mathrm{sr}_j(f,g)$ is nonzero. This means that the composition $\pi_{j-1}^{m+n-j-1} \circ \Phi_{j,g}^{j}$ is surjective. In particular, there are polynomials a, b such that

$$h := f\,a + g\,b = x^j + \text{lower terms}.$$

Let us find the exact form of h. Since $S_j^j(f,g)$ is the matrix of the relevant isomorphism and x_j is the first element of the power basis of the quotient space, the coordinates of the pair (a,b) are

$$\left(S_j^j(f,g)^{-1}\right) \cdot \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

It follows that the coefficients of h (i.e. its coordinates in power basis) are computed by the formula

$$M \cdot \left(S_j^j(f,g)^{-1}\right) \cdot \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

The previous lemma yields

$$h = \left(1, x, \dots, x^{m+n-j-1}\right) \cdot \begin{pmatrix} a_{0,0} & \cdots & a_{0,n+m-2j} \\ \vdots & & \vdots \\ a_{j-1,0} & \cdots & a_{j-1,n+m-2j} \\ 1 & & \\ & \ddots & \\ & & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

where $a_{i,k}$ equals ${}^1/_{\mathrm{sr}_j(f,g)}$ times the determinant of the matrix constructed by replacing the row k of $S_j^j(f,g)$ by the row i of M. Therefore

$$h = \frac{1}{\mathrm{sr}_j(f,g)} \cdot \sum_{k=0}^{j} x^k \cdot \det \left(\begin{array}{ccc:ccc} f_k & \cdots & f_{k+j-n+1} & g_k & \cdots & g_{k+j-m+1} \\ f_{j+1} & \cdots & f_{2j-n+2} & g_{j+1} & \cdots & g_{2j-m+2} \\ \vdots & & \vdots & \vdots & & \vdots \\ f_m & & \vdots & g_n & & \vdots \\ & \ddots & \vdots & & \ddots & \vdots \\ & & f_m & & & g_n \end{array} \right),$$

with are usual convention that coefficients with negative indices are null. Denote the above matrix by $S_k^j(f,g)$. This is consistent with with the notation $S_j^j(f,g)$, where the row j is replaced by itself (i.e. nothing is changed).

Definition 8.4.5. If subresultant coefficient $\mathrm{sr}_j(f,g)$ is nonzero, we define the j-th *subresultant* of f and g to be the polynomial

$$\mathrm{sres}_j(f,g) := \sum_{k=0}^{j} \det\Big(S_k^j(f,g)\Big) \cdot x^k.$$

It is clear from definition and the previous discussion that the degree of $\mathrm{sres}_j(f,g)$ equals j and $\mathrm{sr}_j(f,g)$ is its leading coefficient. For the sake of completeness, when $\mathrm{sr}_j(f,g)$ is null, we set $\mathrm{sres}_j(f,g) := 0$.

Example 106. Take the same two polynomials as in the previous examples

$$f = 3x^2 - 4x + 1 \qquad \text{and} \qquad g = x^3 + 2x^2 - 5.$$

We have

$$\mathrm{sres}_1(f,g) = x \cdot \det \begin{pmatrix} -4 & 1 & 0 \\ 3 & -4 & 2 \\ 0 & 3 & 1 \end{pmatrix} + \det \begin{pmatrix} 1 & 0 & -5 \\ 3 & -4 & 2 \\ 0 & 3 & 1 \end{pmatrix} = 37x - 55.$$

missing

Now we need to derive a number of properties of subresultants. We encourage the reader to compare them with corresponding properties of resultants. In Lemmas$f = f_0 + f_1 x + \cdots + f_m x^m$ and $g = g_0 + g_1 x + \cdots + g_n x^n$ are two polynomials over some unique factorization domain R.

Lemma 8.4.6. $\mathrm{sres}_j(f,g) = (-1)^{(m-j)(n-j)} \cdot \mathrm{sres}_j(f,g)$.

Proof. This is immediate from Definition 8.4.5 and the fact that one gets $S_k^j(g,f)$ from $S_k^j(f,g)$ by shifting the first $n-j$ columns (i.e. those consisting of coefficients of f) to the back. This can be achieved by $(n-j)(m-j)$ swapping of columns. □

Lemma 8.4.7. *For any $\alpha \in R$,*

$$\operatorname{sres}_j(\alpha f, g) = \alpha^{n-j} \operatorname{sres}_j(f,g) \qquad \text{and} \qquad \operatorname{sres}_j(f, \alpha g) = \alpha^{m-j} \operatorname{sres}_j(f,g).$$

Proof. We have

$$\operatorname{sres}_j(f,g) = \sum_{k=0}^{j} \det\Big(S_k^j(\alpha f, g)\Big) \cdot x^k = \sum_{k=0}^{j} \alpha^{n-j} \det\Big(S_k^j(f,g)\Big) \cdot x^k = \alpha^{n-j} \operatorname{sres}_j(f,g).$$

The other case is fully analogous. □

Lemma 8.4.8. *If h is a polynomial of degree $\deg h < \deg f - \deg g$, then*

$$\operatorname{sres}_j(f + gh, g) = \operatorname{sres}_j(f,g).$$

Proof. Take any $a \in K[x]_{n-j-1}$ and $b \in K[x]_{m-j-1}$. We have $\deg h < m-n$, hence also $\deg(ah+b) \leq ax\{n-j-1+m-n, m-j-1\} = m-j-1$ and in particular $ah+b \in K[x]_{m-j-1}$. It follows that the map $\lambda(a,b) := (a, ah+b)$ is an automorphism of the linear space $K[x]_{n-j-1} \times K[x]_{m-j-1}$. Its matrix (with respect to the power bases of both subspaces) is

$$\begin{pmatrix} I & 0 \\ * & I \end{pmatrix}$$

and so the determinant of λ is 1. Thus, we have

$$\Phi^j_{f+gh,g}(a,b) = (f+gh)\cdot a + g \cdot b = f \cdot a + g \cdot (ah+b) = \big(\Phi^j_{f,g} \circ \lambda\big)(a,b).$$

It follows that $\operatorname{sr}_j(f+gh, g) = \det\lambda \cdot \operatorname{sr}_j(f,g)$ and $\operatorname{sres}_j(f+gh,g) = \operatorname{sres}_j(f,g)$, as claimed. □

Lemma 8.4.9. *Assume that $m \deg f \geq \deg g = n$ and let r be the pseudo-remainder of f modulo g. If $r = r_0 + r_1 x + \cdots + r_l x^l$, then*

- $g_n^{(m-n+1)(n-j)} \cdot \operatorname{sres}_j(f,g) = (-1)^{(m-j)(n-j)} \cdot g_n^{m-l} \cdot \operatorname{sres}_j(g,r)$ *for every* $j \in \{0,\ldots,l-1\}$*;*
- $g_n^{(m-n+1)(n-l)} \cdot \operatorname{sres}_l(f,g) = (-1)^{(m-l)(n-l)} \cdot g_n^{m-l} \cdot r_l^{n-l-1} \cdot r$*;*
- $\operatorname{sres}_j(f,g) = 0$ *for every* $j \in \{l+1,\ldots,n-2\}$*;*
- $\operatorname{sres}_{n-1}(f,g) = (-1)^{m-n+1} \cdot r$.

Unfinished

Bibliographic notes and further reading

TO DO

Section 8.1 is based on [9, 30, 37].

The proof of Theorem 8.2.25 and Algorithm 8.2.21 are based on [17, 18]

Although classical, the problem of computing resultants is still vivid and draws attention of many researchers. In [87] the reader may find a summary of recent progress.

9

Gröbner bases

In this chapter we present another technique for solving systems of multivariate polynomial equations through elimination of variables. The approach presented here is more modern and more abstract than the one we encountered previously. The evident advantage is its greater universality, as the methods of Gröbner bases are applicable to a wider range of problems and generalize to other contexts.

9.1 Algebraic prerequisites

Before we begin, we need to recall some facts from the standard course of abstract algebra. In this section we generally omit proofs of quoted results as they would take us to far afield. Instead we just point a reader to a relevant source. In previous chapters, when working with polynomials of one variable, we often took advantage of the fact that a ring of univariate polynomials over a field is a principal ideal domain. That means that every ideal is generated by just a single element. This no longer holds in the realm of multivariate polynomials. Nevertheless, one should not despair, since the second best thing after principal ideals are ideals generated by finitely many elements.

Theorem 9.1.1. *Let R be a ring. The following conditions are equivalent:*

- *every ideal $I \lhd R$ is finitely generated (i.e. there are finitely many elements $a_1, \ldots, a_k \in I$ such that $I = \langle a_1, \ldots, a_k \rangle$);*
- *every strictly increasing chain $I_1 \subsetneq I_2 \subsetneq \cdots \subsetneq I_k \subsetneq \cdots$ of ideal of R is finite;*
- *every family of R-ideals contains a maximal element (with respect to inclusion).*

For a proof we refer a reader e.g. to [51, Chapter X, § 1]. A ring is called *Noetherian* if it satisfies any (hence all) of the above conditions.

Theorem 9.1.2 (Hilbert[1] basis theorem)**.** *If R is a Noetherian ring, then so is the ring of polynomials over R.*

[1] David Hilbert (born 23 January 1862, died 14 February 1943) was a German mathematician.

Proof no. 1, wikipedia?

We again omit a proof, that can be found e.g. in [51, Theorem IV.4.1]. Since every principal ideal domain is clearly Noetherian, thus in particular $\mathbb{Z}$, $\mathbb{Z}[x]$ and $K[x]$ are Noetherian (here, as usually, K is a field). Applying Hilbert basis theorem inductively, we see that a multivariate polynomial ring $K[x_1,\ldots,x_n]$ is Noetherian, too. To simplify further references, let us explicitly write down the following:

Corollary 9.1.3. *Every ideal $I \lhd K[x_1,\ldots,x_n]$ of a multivariate polynomial ring (over a field) is finitely generated.*

Corollary 9.1.4. *Every strictly increasing chain of ideals in $K[x_1,\ldots,x_n]$ is finite.*

In this chapter we are dealing with systems of multivariate polynomials, like for example

$$\begin{cases} f_1(x_1,\ldots,x_n) = 0 \\ \cdots \\ f_m(x_1,\ldots,x_n) = 0, \end{cases}$$

where $f_1,\ldots,f_m \in K[x_1,\ldots,x_n]$ and K is a fixed field. Let A be the set of solutions of this system over an algebraic closure K^{alg} of K. Consider the ideal $I = \langle f_1,\ldots,f_m\rangle$ generated by the above polynomials. From the algebra course we know that

$$I = \big\{f_1g_1 + \cdots + f_mg_m \,|\, g_1,\ldots,g_m \in K[x_1,\ldots,x_n]\big\}.$$

Observe that A is precisely the set, where all the polynomials from I vanish simultaneously. Informally speaking, A is the set of solutions of an infinite system of equations $f = 0$ for all $f \in I$. Indeed, if $f \in I$, then $f = f_1g_1 + \cdots + f_mg_m$, for some $g_1,\ldots,g_m \in K[x_1,\ldots,x_n]$. Since all f_i's vanish on A, then so does f. Conversely, if $P \in \big(K^{\mathrm{alg}}\big)^n$ is such a point that $f(P) = 0$ for every $f \in I$, then in particular $f_1(P) = \cdots = f_m(P) = 0$ and so P is one of the solutions of the original system. Hence $P \in A$, as desired. Thus, when thinking about systems of polynomial equations, we may use the language of ideals and consequently the tools of commutative algebra.

Take now an ideal $I \lhd K[x_1,\ldots,x_n]$. The set of all the points (with coordinates in the algebraic closure K^{alg} of K), where all the polynomials form I vanish is called the *algebraic set* associated to I and denoted $\mathscr{Z}(I)$. Conversely, given any subset $A \subseteq \big(K^{\mathrm{alg}}\big)^n$, consider a set $\mathscr{I}(A)$ of all the polynomials, which vanish on A:

$$\mathscr{I}(A) = \big\{f \in K[x_1,\ldots,x_n] \,|\, f(P) = 0 \text{ for all } P \in A\big\}.$$

It is straightforward to check that $\mathscr{I}(A)$ is an ideal. We leave it as an exercise to the reader. One may not expect the operators $\mathscr{Z}$ and $\mathscr{I}$ to be inverses of each other. For instance, $\mathscr{I}\big(\mathbb{C}\setminus\{0\}\big) = \{0\} \lhd \mathbb{C}[x_1,\ldots,x_n]$, while $\mathscr{Z}\big(\{0\}\big) = \mathbb{C}$. Nevertheless some interrelations are easy to prove.

Observation 9.1.5. *If A is a set, then $A \subseteq \mathscr{Z}\big(\mathscr{I}(a)\big)$.*

9.2 Monomial orders

We begin by introducing a certain class of relations that will be fundamental for what follows. Recall that a *linear order* is a relation $\preceq$ on a set A satisfying the following axioms:

- *reflexivity*: $a \preceq a$ for every $a \in A$;
- *antisymmetry*: for every $a, b \in A$, if $a \preceq b$ and $b \preceq a$, then $a = b$;
- *transitivity*: for every $a, b, c \in A$, if $a \preceq b$ and $b \preceq c$, then $a \preceq c$;
- *totality*: for every $a, b \in A$, either $a \preceq b$ or $b \preceq a$.

Here, for a didactic purpose, we explicitly stated four axioms, but it is easy to observe that in fact reflexivity follows from totality. Given a "weak" relation $\preceq$, we will always associate to it a "strong" relation $\prec$ in an obvious way: $a \prec b$ if $a \preceq b$ and $a \neq b$. Take now a finite Cartesian power $\mathbb{N}_0^n = \mathbb{N}_0 \times \cdots \times \mathbb{N}_0$ of non-negative integers. This is an additive monoid, that means: an algebraic structure with an associative binary operation together with an identity element $0 := (0, \ldots, 0)$. We wish to distinguish those linear orders $\preceq$ on $\mathbb{N}_0^n$ that satisfy the following additional conditions:

AO$_1$. $0 \preceq \alpha$ for every $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{N}_0^n$;

AO$_2$. every nonempty subset $S \subseteq \mathbb{N}_0^n$ has a minimum;

AO$_3$. the relation is compatible with addition in the sense that

$$\alpha \preceq \beta \Longrightarrow \alpha + \gamma \preceq \beta + \gamma,$$

for every $\alpha, \beta, \gamma \in \mathbb{N}_0^n$.

Take a ring $K[x_1, \ldots, x_n]$ of multivariate polynomials over a field K. Hereafter, we shall denote it $K[\mathbb{X}]$ for short. Any element (polynomial) of this ring is a sum of *terms* of the form

$$c_\alpha \cdot \mathbb{X}^\alpha := c_{\alpha_1, \ldots, \alpha_n} \cdot x_1^{\alpha_1} \cdots x_n^{\alpha_n},$$

where $c_\alpha \in K$ and $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{N}_0^n$. The set of monic terms, that we shall hereafter call just *monomials*[2], will be denoted

$$\mathbb{M}^n := \left\{ \mathbb{X}^\alpha \mid \alpha \in \mathbb{N}_0^n \right\}.$$

Clearly, this is again a monoid, isomorphic to $\mathbb{N}_0^n$. Hence every relation on the letter set induces a relation on the set of monomials. A relation induced by a linear order satisfying **AO**$_1$–**AO**$_3$ is called a monomial order (other names: *admissible order*, *term order*). Stating explicitly, the conditions translate as follows:

Definition 9.2.1. A linear relation $\preceq$ on the set $\mathbb{M}^n$ is called a *monomial order* if

MO$_1$. $1 \preceq \mathbb{X}^\alpha$ for every $\mathbb{X}^\alpha \in \mathbb{M}^n$;

[2] This is a non-standard terminology. Usually "monomial" and "term" are synonyms and objects called monomials here are called "power products" or "monic terms/monomials".

MO$_2$. every nonempty subset $S \subseteq \mathbb{M}^n$ has a minimum;

MO$_3$. the relation is compatible with multiplication in the sense that

$$\mathbb{X}^\alpha \preceq \mathbb{X}^\beta \Longrightarrow \mathbb{X}^\alpha \cdot \mathbb{X}^\gamma \preceq \mathbb{X}^\beta \cdot \mathbb{X}^\gamma,$$

for every $\mathbb{X}^\alpha, \mathbb{X}^\beta, \mathbb{X}^\gamma \in \mathbb{M}^n$.

The choice of a particular monomial order not only let us fix a sequence of terms, when presenting polynomials, but more importantly, it influences the speed of certain computations and determines which operations are allowed (for example elimination of variables requires the monomial order to be an *elimination order*—see Section 9.8).

Now, we introduce some classical monomial orders:

- *Lexicographic order* (lex): let $\alpha = (\alpha_1, \ldots, \alpha_n)$ and $\beta = (\beta_1, \ldots, \beta_n)$ be two distinct elements of $\mathbb{N}_0^n$, then $\mathbb{X}^\alpha \prec_{\text{lex}} \mathbb{X}^\beta$ if there is an index $k \leq n$ such that $\alpha_j = \beta_j$ for all $j < k$ and $\alpha_k < \beta_k$.
- *Graded lexicographic order* (glex): for a tuple $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{N}_0^n$, denote $|\alpha| := \alpha_1 + \cdots + \alpha_n$ and define $\mathbb{X}^\alpha \prec_{\text{glex}} \mathbb{X}^\beta$, if either $|\alpha| < |\beta|$ or $|\alpha| = |\beta|$ and $\mathbb{X}^\alpha \prec_{\text{lex}} \mathbb{X}^\beta$.
- *Reverse lexicographic order* (revlex). This is a direct opposite of lex—one reverses both: the direction of the inequality and the sequence of inspecting variables. To be more precise, we have $\mathbb{X}^\alpha \prec_{\text{revlex}} \mathbb{X}^\beta$, when there is an index $k \leq n$ such that $\alpha_j = \beta_j$ for all $j > k$ and $\alpha_k > \beta_k$.
- *Graded reverse lexicographic order* (grevlex) is obtained from the reverse lexicographic order in the same way as glex is obtained from lex. Namely, $\mathbb{X}^\alpha \prec_{\text{grevlex}} \mathbb{X}^\beta$, when either $|\alpha| < |\beta|$ or $|\alpha| = |\beta|$ and $\mathbb{X}^\alpha \prec_{\text{revlex}} \mathbb{X}^\beta$.

The graded lexicographic and graded reverse lexicographic orders are alternatively called *degree lexicographic* (deglex) and *degree reverse lexicographic* (degrevlex) orders.

Example 107. Take a polynomial $f = 2xy^2z + 3xyz^3 + 5x^2z^2$. If we write it with a decreasing order of terms with respect to the lexicographic order (with $x \succ y \succ z$), then we have

$$f = 5x^2z^2 + 2xy^2z + 3xyz^3.$$

On the other hand, using glex we would write

$$f = 3xyz^3 + 5x^2z^2 + 2xy^2z.$$

Still, reverse lexicographic order yields:

$$f = 2xy^2z + 5x^2z^2 + 3xyz^3,$$

while grevlex forces yet another order of terms:

$$f = 3xyz^3 + 2xy^2z + 5x^2z^2.$$

Suppose we are given a monomial order $\preceq$ and a multivariate polynomial

$$f = c_{\alpha_1}\mathbb{X}^{\alpha_1} + c_{\alpha_2}\mathbb{X}^{\alpha_2} + \cdots + c_{\alpha_m}\mathbb{X}^{\alpha_m}.$$

Define the *support* of f to be the set of all monomials in f:

$$\operatorname{supp} f := \left\{\mathbb{X}^{\alpha_1}, \ldots, \mathbb{X}^{\alpha_m}\right\}.$$

The maximal element of $\operatorname{supp} f$, with respect to the fixed order $\preceq$, is called the *leading monomial* of f and denoted $\operatorname{lm}_{\preceq}(f)$. If $\operatorname{lm}_{\preceq}(f) = \mathbb{X}^{\beta}$, then the *leading coefficient* of f is $\operatorname{lc}_{\preceq}(f) := c_{\beta}$. Finally, the term $c_{\beta} \cdot \mathbb{X}^{\beta} = \operatorname{lc}_{\preceq}(f) \cdot \operatorname{lm}_{\preceq}(f)$ is unsurprisingly called the *leading term* of f and denoted $\operatorname{lt}_{\preceq}(f)$. If the monomial order is fixed and there is no risk of confusion, we shall drop the subscript and write just $\operatorname{lm}(f)$, $\operatorname{lc}(f)$ and $\operatorname{lt}(f)$.

Example 108 (continued). Consider the same polynomial f as in the previous example. Then, with respect to the lexicographic order we have

$$\operatorname{lm}_{lex}(f) = x^2z^2, \qquad \operatorname{lc}_{lex}(f) = 5, \qquad \operatorname{lt}_{lex}(f) = 5x^2z^2.$$

Proposition 9.2.2. *A ring $K[x]$ of univariate polynomials over a field admits only one monomial order, namely*

$$x^m \prec x^n \iff m < n.$$

Proof. Let $\preceq$ be a monomial order on $\mathbb{M}^1$. Fix two distinct integers $m, n \in \mathbb{N}_0$. Suppose that $m < n$. We have $1 \prec x^{n-m}$ by $\mathbf{MO}_1$ and so $\mathbf{MO}_3$ implies that $x^m \prec x^n$. □

9.3 Gauss elimination revisited

In this chapter we deal with systems of multivariate polynomial equations of arbitrary degrees. Nevertheless, in order to provide certain motivation and build some intuition for what follows, let us first consider an example system of linear equation (say over $\mathbb{Q}$):

$$\begin{cases} x - 2y + 3z - 2 = 0 \\ 2x + y - z + 1 = 0 \\ x + y - 2z + 4 = 0. \end{cases}$$

The left-hand-sides of the above equations are linear polynomials, that we shall denote respectively by f_1, f_2 and f_3. Think about the lexicographic order in $\mathbb{Q}[x, y, z]$ with $x \succ y \succ z$. Hence, the polynomials above are written in a descending order of terms, with respect to lex.

An elementary method for solving the above system is to perform Gaussian elimination of rows (see Section ?). In order to eliminate variable x from the second and third equations, we replace f_2 (respectively f_3) by a new polynomial of the form $f_2 - 2 \cdot f_1 = 5y - 7z + 5$ (respectively $f_3 - f_1 = 3y - 5z + 6$). In a fancy way, we write it as

missing

$$f_j \leftarrow f_j - \frac{c_{\alpha}\mathbb{X}^{\alpha}}{\operatorname{lt}(f_1)} \cdot f_1, \qquad \text{for } j = 2, 3 \tag{9.1}$$

where $\mathbb{X}^\alpha$ is a nonzero monomial of f_j divisible by $\mathrm{lm}(f_1)$. This lead to a new system

$$\begin{cases} x-2y+3z-2=0 \\ 5y-7z+5=0 \\ 3y-5z+6=0. \end{cases}$$

Observe that the leading term of the second polynomial is now $5y$ and so we may eliminate y from the two other equations using a rule of substitution similar to Eq. (9.1), but with f_1 replaced by f_2 and the condition $j=2,3$, replaced by $j=1,3$:

$$f_j \leftarrow f_j - \frac{c_\alpha \mathbb{X}^\alpha}{\mathrm{lt}(f_2)} \cdot f_2, \qquad \text{for } j=1,3,$$

This gives:

$$\begin{cases} x+\frac{1}{5}z=0 \\ 5y-7z+5=0 \\ -\frac{4}{5}z+3=0. \end{cases}$$

Finally, eliminate z, one more time applying a substitution of the form as in Eq. (9.1), but now take f_3 in place of f_1. Therefore

$$\begin{cases} x+\frac{3}{4}=0 \\ 5y-\frac{85}{4}=0 \\ -\frac{4}{5}z+3=0 \end{cases}$$

and the unique solution of our system is $x=-\frac{3}{4}$, $y=\frac{17}{4}$, $z=\frac{15}{4}$. The polynomials $g_1 := x+\frac{3}{4}$, $g_2 := y-\frac{17}{4}$ and $g_3 := z-\frac{15}{4}$ generate exactly the same ideal in $\mathbb{Q}[x,y,z]$ as the original polynomials f_2, f_2, f_3. They are just "nicer generators", that allow us to read the solutions. As we will learn in Section 9.7, the polynomials g_1, g_2, g_3 form a *reduced Gröbner basis* of this ideal.

9.4 Multivariate polynomial reduction

Observe that a way we presented Eq. (9.1), it can be directly applied to any multivariate polynomial, not necessary a linear one. Let K be a field of constants and assume that we have fixed some monomial order $\preceq$ on $K[\mathbb{X}]$.

Definition 9.4.1. We say that a polynomial f *reduces in one step* to a polynomial h modulo a polynomial g (with respect to the fixed monomial order $\preceq$), if there is a nonzero term $c_\alpha \mathbb{X}^\alpha$ in f such that

$$\mathrm{lm}(g) \text{ divides } \mathbb{X}^\alpha \qquad \text{and} \qquad h = f - \frac{c_\alpha \mathbb{X}^\alpha}{\mathrm{lt}(g)} \cdot g.$$

When this is the case, we write shortly $f \xrightarrow{g} h$.

Definition 9.4.2. We say that a polynomial f *reduces* to a polynomial h modulo a subset $G \subset K[x_1, \ldots, x_n]$, denoted $f \xrightarrow{G} h$, if there is a finite sequence of polynomials $g_1, \ldots, g_m \in G$ (not necessary distinct) such that

$$f = f_0 \xrightarrow{g_1} f_1 \xrightarrow{g_2} \cdots \xrightarrow{g_m} f_m = h.$$

Example 109. Take a ring $\mathbb{Q}[x, y, z]$ with lexicographic order, $x \succ y \succ z$ and consider polynomials

$$f = x^2 - xy, \qquad g_1 = x^2 - z, \quad g_2 = xy + z.$$

Then

$$f \xrightarrow{g_1} -xy + z \xrightarrow{g_2} 2z.$$

The polynomial $2z$ cannot be further reduces.

Of course, any theory developed for multivariate polynomials should be applicable to univariate polynomials, as well. Thus, take for example two univariate polynomials $f = 4x^5 + 4x^4 - 10x^3 - 12x^2 + 7x + 13$ and $g = 2x^2 - 3$. Reducing f modulo g as many times as possible, we have

$$\begin{aligned} f &\xrightarrow{g} 4x^4 - 4x^3 - 12x^2 + 7x + 13 \\ &\xrightarrow{g} -4x^3 - 6x^2 + 7x + 13 \\ &\xrightarrow{g} -6x^2 + x + 13 \\ &\xrightarrow{g} x + 4. \end{aligned}$$

The resulting polynomial $h = x + 4$ is nothing else but the remainder of f modulo g (in a sense of the standard polynomial division). This example suggests how to define remainders in the multivariate case.

Definition 9.4.3. A polynomial $f \in K[\mathbb{X}]$ is said to be *reduced* with respect to a subset $G \subset K[\mathbb{X}]$ if f does not contain nonzero terms divisible by the leading term of any $g \in G$.

In a nutshell, f is reduced if and only if one cannot reduce it any further.

Definition 9.4.4. If f reduces to a polynomial h modulo $G \subset K[\mathbb{X}]$ and h is reduced with respect to G, then we say that h is a *remainder* of f modulo G.

Exercise 24. Let $f, g \in K[x]$ be two univariate polynomials. Prove that the remainder of f modulo $\{g\}$ in a sense of the above definition is the same as the remainder of f modulo g in a standard sense.

Exercise 25. Let $f \in K[x]$ be a univariate polynomial and $I \lhd K[x]$ be an ideal generated by some polynomial g. Prove that the remainder of f modulo I is unique and equals $f \bmod g$.

In contrast to the univariate case, in the realm of multivariate polynomials, a remainder has no longer to be unique, as the following example shows.

Example 110. Take a ring $P = \mathbb{Q}[x, y]$ and use the lexicographic order with $x \succ y$. Consider the following three polynomials:

$$f = x^2y + 3x^2 - y^2, \qquad g_1 = xy + 2x, \quad g_2 = x^2 - y^2.$$

Reducing f, first modulo g_1, then modulo g_2 yields:

$$\begin{aligned} f &\xrightarrow{g_1} x^2 - y^2 \\ &\xrightarrow{g_2} 0. \end{aligned}$$

On the other hand, if we first reduce f modulo g_2, we cannot reduce it by g_1 any longer, however we may reduce it again modulo g_2. This way we have

$$\begin{aligned} f &\xrightarrow{g_2} 3x^2 + y^3 - y^2 \\ &\xrightarrow{g_2} y^3 + 2y^2. \end{aligned}$$

As we see, the remainders obtained in both cases differ.

In the next section we characterize those sets for which remainders are unique. First, however, let us present an algorithm that computes the remainder of a polynomial modulo a set of polynomials. Basically it is a generalization of univariate monomial division (see Algorithm 1.1.12).

Algorithm 9.4.5. *Let $R = K[\mathbb{X}]$ be a multivariate polynomial ring and $\preceq$ a fixed monomial order. Given a polynomial f and a finite set $G = \{g_1, \ldots, g_n\} \subset R$, this algorithm returns a remainder of f modulo G (possibly one of the remainders if the remainder is not unique).*

1. *Initialize $r := f$;*
2. *repeat the following steps:*
   a) *for every term $c_\alpha \mathbb{X}^\alpha$ of r and every $g_i \in G$, check whether $\operatorname{lm}(g_i)$ divides $c_\alpha \mathbb{X}^\alpha$;*
   b) *if no such $c_\alpha \mathbb{X}^\alpha$ and g_i are found, then return r and quit;*
   c) *otherwise reduce r setting*

$$r := r - \frac{c_\alpha \mathbb{X}^\alpha}{\operatorname{lt}(g_i)} \cdot g_i.$$

Proof of termination

Let f, G be as above and let r be the remainder returned by the algorithm, then we may express f as

$$f = h_1 g_1 + \cdots + h_n g_n + r,$$

where each h_i is a sum of all the terms $\frac{c_\alpha \mathbb{X}^\alpha}{\operatorname{lt}(g_i)}$ computed in successive reductions modulo g_i.

Observation 9.4.6. *If the remainder of f modulo $G = \{g_1, \ldots, g_n\}$ is zero, then f belongs to the ideal $\langle G \rangle$.*

9.5 Gröbner bases

At the end of previous section, we observed that for any set of generators G of an ideal $I = \langle G \rangle$, a sufficient condition for a polynomial f to belong to I is $f \xrightarrow{G} 0$. Now, we shall characterize these sets of generators for which this is also a necessary condition.

Theorem 9.5.1. *Let G be a finite set of generators of some ideal $I \lhd K[\mathbb{X}]$ and let $\preceq$ be a fixed monomial order. The following conditions are equivalent:*

1. *the set of leading terms (with respect to $\preceq$) of the elements of G generate the ideal $\mathrm{Lt}(I) := \langle \mathrm{lt}(f) \mid f \in I \rangle$;*
2. *the leading term $\mathrm{lt}(f)$ of every nonzero polynomial $f \in I$ is divisible by the leading term $\mathrm{lt}(g)$ of some $g \in G$;*
3. *a remainder of every polynomial $f \in K[\mathbb{X}]$ modulo G is unique;*
4. *if $f \in K[\mathbb{X}]$ is a polynomial, then f belongs to I if and only if $f \xrightarrow{G} 0$.*

Proof

Definition 9.5.2. The set G satisfying any (hence all) of the conditions in Theorem 9.5.1 is called a *Gröbner basis* (with respect to the fixed monomial order) of the ideal $I = \langle G \rangle$.

In the next section we will learn how to check whether a given set forms a Gröbner basis and how to construct Gröbner bases. Here, we shall only emphasize the fact, that the condition of being a Gröbner basis is relative to the selected monomial order. A set G can be a Gröbner basis with respect to one monomial order but not with respect to another one. We will construct such an example (see Example 111), once we have all the necessary tools at our disposal.

An immediate consequence of Theorem 9.5.1 is the correctness of the following membership and inclusion tests.

Algorithm 9.5.3. *Let $R = K[\mathbb{X}]$ be a multivariate polynomial ring and $\preceq$ be a monomial order. Given a polynomial $f \in R$ and a Gröbner basis G of an ideal $I \lhd R$, this algorithm tests whether $f \in I$.*

1. *Compute the remainder r of f modulo G;*
2. *return true if $r = 0$ and false otherwise.*

Algorithm 9.5.4. *Let $R = K[\mathbb{X}]$ be a multivariate polynomial ring and $\preceq$ be a monomial order. Given a finite set $F = \{f_1, \ldots, f_m\}$ of generators of an ideal $I \lhd R$ and a Gröbner basis G of another ideal $J \lhd R$, this algorithm tests if $I \subseteq J$.*

1. *Compute the remainders $r_1, \ldots, r_m$ of all the polynomials $f_1, \ldots, f_m$;*
2. *if all r_i's are zero, then return true, otherwise return false.*

Proposition 9.5.5. *Every ideal $I \lhd K[\mathbb{X}]$ has a Gröbner basis.*

Proof

9.6 Buchberger's criterion

In this section we develop an algorithm, due to B. Buchberger, that constructs a Gröbner basis of an ideal $I \lhd K[\mathbb{X}]$ given any (finite) set of generators of I. To this end, we need first to introduce a notion of S-polynomials. Observe that the least common multiple of any two monomials $\mathbb{X}^\alpha$, $\mathbb{X}^\beta$ is effectively computable and equals $\mathbb{X}^\gamma$, where γ is the component-wise maximum of α and β:

$$\gamma = (\gamma_1, \ldots, \gamma_n), \qquad \gamma_i = \max(\alpha_i, \beta_i) \quad \text{for } 1 \leq i \leq n.$$

Definition 9.6.1. Let $f, g \in K[\mathbb{X}]$ be two nonzero polynomials and assume that $\preceq$ is a monomial order in $K[\mathbb{X}]$. The *S-polynomial* of f, g is a polynomial given by the formula

$$S_\preceq(f,g) := \frac{\operatorname{lcm}\big(\operatorname{lm}_\preceq(f), \operatorname{lm}_\preceq(g)\big)}{\operatorname{lt}_\preceq(f)} \cdot f - \frac{\operatorname{lcm}\big(\operatorname{lm}_\preceq(f), \operatorname{lm}_\preceq(g)\big)}{\operatorname{lt}_\preceq(g)} \cdot g.$$

We again tend to omit the subscript, when it is clear what the monomial order is used.

Theorem 9.6.2 (Buchberger)**.** *Let $G = \{g_1, \ldots, g_n\} \subset K[\mathbb{X}]$ and assume that a monomial order in $K[\mathbb{X}]$ is fixed. The following conditions are equivalent:*

1. *G is a Gröbner basis;*
2. *$S(g_i, g_j) \xrightarrow{G} 0$ for every $1 \leq i, j \leq n$.*

Proof

Let us rewrite Buchberger's theorem in form of an algorithm.

Algorithm 9.6.3. *Given a finite subset $G = \{g_1, \ldots, g_n\}$ of a multivariate polynomial ring $K[\mathbb{X}]$ equipped with some fixed monomial order $\preceq$, this algorithm returns either a logical constant 'true', if G is a Gröbner basis of $\langle G \rangle$, or 'false' together with a nonzero remainder of the S-polynomial of two elements in G.*

1. *For every two polynomials $f, g \in G$, $f \neq g$:*
   a) *construct their S-polynomial;*
   b) *find the remainder r of $S(f,g)$ modulo G, using Algorithm 9.4.5;*
   c) *if $r \neq 0$, then return $(false, r)$.*
2. *if all the remainders are zero, then return true.*

Example 111. Take a ring $R = \mathbb{Q}[x, y, z]$ and polynomials

$$g_1 = 3y^2 - x, \qquad g_2 = 3x^2 - z.$$

Then, with respect to graded lexicographic order with $x \succ y \succ z$, we have

$$S_{\text{grevlex}}(g_1, g_2) = -\frac{1}{3}x^3 + \frac{1}{3}y^2 z \xrightarrow{\{g_1, g_2\}} 0.$$

On the other hand, lexicographic order (still $x \succ y \succ z$) for the same polynomials yields

$$S_{\text{lex}}(g_1, g_2) = -3xy^2 + \frac{1}{3}z \xrightarrow{\{g_1,g_2\}} -9y^4 + \frac{1}{3}z \neq 0.$$

Thus, $\{g_1, g_2\}$ is a Gröbner basis with respect to grevlex but not w.r.t. lex.

Buchberger's theorem not only provides a mean of verifying whether a set is a Gröbner basis but also gives a method of constructing such a basis. The general strategy is to start from some (finite) set of generators and keep appending nonzero remainders of S-polynomials to it.

Algorithm 9.6.4 (Buchberger)**.** *Let $R = K[\mathbb{X}]$ be a ring of multivariate polynomials equipped with some fixed monomial order. Given a finite set F of generators of some ideal $I \lhd R$, this algorithm returns a Gröbner basis G of I, which contains F.*

1. *Initialize $G := F$;*
2. *execute Algorithm 9.6.3 with an input G, if it returns true, then output G and quit;*
3. *otherwise append the remainder returned by Algorithm 9.6.3 to G and repeat the previous step.*

Proof of correctness. The correctness of the result is immediate by Theorem 9.5.1, hence we only need to show that the algorithm terminates in finite time. Denote by $G_0, G_1, \ldots$ the sets constructed in the successive iterations of the algorithm. Thus, $G_0 = F$ and $G_{k+1} = G_k \cup \{r_k\}$, where $r_k \neq 0$ is a remainder of $S(g_i, g_j)$ for some $g_i, g_j \in G_k$. Consider the ideals generated by leading terms of these sets:

$$I_0 := \operatorname{Lt}(G_0) = \langle \operatorname{lt}(g) \mid g \in G_0 \rangle, \ldots I_k := \operatorname{Lt}(G_k) = \langle \operatorname{lt}(g) \mid g \in G_k \rangle, \ldots .$$

Since r_k is reduced with respect to G_k, thus it cannot belong to $\operatorname{Lt}(G_k)$. It follows that $I_0 \subsetneq I_1 \subsetneq \cdots \subsetneq I_k \subsetneq \cdots$ is a strictly ascending chain of ideals. Now, Hilbert basis theorem (see Corollary 9.1.4) asserts that this chain is finite. Hence, the algorithm terminates. □

Example 112. Consider the lexicographic order on $R = \mathbb{Q}[x, y]$ with $x \succ y$. Let

$$g_1 = x^2 + xy + y^2, \qquad g_2 = x + y, \qquad g_3 = x.$$

The S-polynomial $S(g_1, g_2)$ equals

$$S\big(x^2 + xy + y^2, x + y\big) = \frac{\operatorname{lcm}\big(x^2, x\big)}{x^2} \cdot (x^2 + xy + y^2) - \frac{\operatorname{lcm}\big(x^2, x\big)}{x} \cdot (x + y) = y^2.$$

It is clearly reduced with respect to the set $\{g_1, g_2, g_3\}$. Hence this set does not form a Gröbner basis. Append y^2 to it and compute $S(g_2, g_3) = y$. This polynomial is again reduced. Therefore we append it to our set of generators. The resulting set

$$\big\{x^2 + xy + y^2, x + y, x, y^2, y\big\}$$

is now a Gröbner basis, because now S-polynomials of every two elements reduce to zero.

9.7 Minimal and reduced Gröbner bases

In general, a Gröbner basis produced by Algorithm 9.6.4 can turn out to be inconveniently large including superfluous elements. For instance it contains the whole original set of generators. Take for example an ideal $I = \langle x^4 - x^3 - 2x + 2, x^3 - x^2 + 3x - 3\rangle$ in a univariate polynomial ring, say $\mathbb{Q}[x]$. Algorithm 9.6.4 computes a Gröbner basis of I

$$\{x - 1, x^4 - x^3 - 2x + 2, x^3 - x^2 + 3x - 3\}.$$

It is clear that only the first polynomial really matters. The two others may simply be omitted. This example suggests how to get rid of superfluous elements appearing in a Gröbner basis.

Theorem 9.7.1. *Let G be a Gröbner basis of an ideal $I \lhd K[\mathbb{X}]$ and let f, g be two distinct elements of G. If $\mathrm{lm}(g)$ divides $\mathrm{lm}(f)$, then $G \setminus \{f\}$ is also a Gröbner basis of the same ideal I.*

Proof

Definition 9.7.2. A Gröbner basis G is called *minimal*, if there are no distinct elements $f, g \in G$ with $\mathrm{lm}(f)$ divisible by $\mathrm{lm}(g)$.

The name "minimal" is justified by the following result.

Theorem 9.7.3. *Every two minimal Gröbner bases (with respect to the same monomial order) of a given ideal are equinumerous.*

Proof

Observe that in the above proof we actually showed more than just the thesis of Theorem 9.7.3. In fact we proved:

Corollary 9.7.4. *If $G = \{g_1, \ldots, g_n\}$ and $H = \{h_1, \ldots, h_n\}$ are two minimal Gröbner bases of the same ideal I, then there is such a permutation $\sigma \in S_n$, that*

$$\mathrm{lm}(g_i) = \mathrm{lm}(h_{\sigma(i)}), \qquad \text{for} \quad 1 \leq i \leq n.$$

Theorem 9.7.1 may be directly converted into an algorithm producing a minimal Gröbner basis.

Algorithm 9.7.5. *Given a Gröbner basis G of an ideal I, this algorithm computes a minimal Gröbner basis of I, contained in G.*

1. *For every $g \in G$ and every $f \in G \setminus \{g\}$:*
   a) *check if $\mathrm{lm}(g)$ divides $\mathrm{lm}(f)$;*
   b) *if it does, then remove f from G;*
2. *return G.*

In Theorem 9.7.1 (an the above Algorithm) we were checking whether the leading monomial of one polynomial (say f) is divisible by the leading monomial of another polynomial (say g). If instead we test every monomial from $\operatorname{supp}(f)$ for divisibility by $\operatorname{lm}(g)$, then we will end up in something even better than a minimal Gröbner basis, namely a *reduced* Gröbner basis.

Theorem 9.7.6. *Let G be a Gröbner basis of an ideal $I \lhd K[\mathbb{X}]$ and let f, g be two distinct elements of G. If f reduces to h modulo g, then $G \setminus \{f\} \cup \{h\}$ is again a Gröbner basis of the same ideal I.*

Proof

Definition 9.7.7. A Gröbner basis G of an ideal $I \lhd K[\mathbb{X}]$ is called *reduced* if:

- it consists of monic polynomials (i.e. $\operatorname{lc}(g) = 1$ for every $g \in G$);
- for every $g \in G$ no nonzero term $c_\alpha \mathbb{X}^\alpha \in \operatorname{supp}(g)$ is divisible by a leading monomial $\operatorname{lm}(h)$ of some $h \neq g$, $h \in G$.

Also the above theorem is directly translatable to an algorithm.

Algorithm 9.7.8. *Given a minimal Gröbner basis of an ideal $I \lhd K[\mathbb{X}]$, this algorithm constructs a reduced Gröbner basis of I.*

1. *Divide every element of G by its leading coefficients.*
2. *For every element $g \in G$ do the following:*
   a) *Compute the remainder r of g modulo $G \setminus \{g\}$.*
   b) *Replace g by r in G.*
3. *Eeturn G.*

Example 113 (continuation of Example 112). Take an ideal $I \lhd \mathbb{Q}[x, y]$ generated by the polynomial from Example 112:

$$g_1 = x^2 + xy + y^2, \qquad g_2 = x + y, \qquad g_3 = x.$$

As we know, Algorithm 9.6.4 produces a Gröbner basis of I:

$$G_1 = \left\{x^2 + xy + y^2, x + y, x, y^2, y\right\}.$$

Applying Algorithm 9.7.5 to it, we obtain a minimal basis:

$$G_2 = \left\{x + y, y\right\}.$$

The polynomial $x + y$ reduces further to x modulo y. Therefore, the reduced Gröbner basis of this ideal is

$$G_3 = \left\{x, y\right\}.$$

Example 114. Take the three linear polynomials from Section 9.3:

$$f_1 = x - 2y + 3z - 2, \qquad f_2 = 2x + y - z + 1, \qquad f_3 = x + y - 2z + 4$$

in the ring $\mathbb{Q}[x, y, z]$ equipped with lexicographic order, $x \succ y \succ z$. The Buchberger's algorithm (Algorithm 9.6.4) computes a Gröbner basis:

$$G_1 = \left\{x - 2y + 3z - 2, 2x + y - z + 1, x + y - 2z + 4, -\frac{5}{2}y + \frac{7}{2}z - \frac{5}{2}, \frac{4}{5}z - 3\right\}.$$

A minimal Gröbner basis contained in G_1 and computed by the means of Algorithm 9.7.5 is

$$G_2 = \left\{x - 2y + 3z - 2, -\frac{5}{2}y + \frac{7}{2}z - \frac{5}{2}, \frac{4}{5}z - 3\right\}.$$

Finally, we reduce it using Algorithm 9.7.8 to get

$$G_3 = \left\{x + \frac{3}{4}, y - \frac{17}{4}, z - \frac{15}{4}\right\}.$$

Theorem 9.7.9 (Buchberger)**.** *Every nonzero ideal in a multivariate polynomial ring $K[\mathbb{X}]$ over a field has a unique reduced Gröbner basis (with respect to a fixed monomial order).*

Proof. Since every ideal has a finite set of generators by the Hilbert's basis theorem (see Corollary 9.1.3), thus the existence follows by successive applications of Algorithms 9.6.4, 9.7.5 and 9.7.8. Therefore we only need to prove the uniqueness.

TODO

□

An immediate application of the above theorem is a criterion for testing ideal equality:

Corollary 9.7.10. *Two nonzero ideals of a multivariate polynomial ring are equal if and only if their reduced Gröbner bases (with respect to the same monomial order) are equal (as sets).*

9.8 Elimination of variables

Now it is time to pick fruits of the theory developed so far and see how Gröbner bases can be used when solving systems of polynomial equations. The main tool is the elimination theorem presented below, which requires us first to introduce a notion of elimination orders.

Definition 9.8.1. Consider a ring $K[x_1, \ldots, x_m; y_1, \ldots, y_n]$ of multivariate polynomials in variables $x_1, \ldots, x_m$ and $y_1, \ldots, y_n$. A monomial order $\preceq$ is called an *elimination*

order with respect to variables $x_1, \dots, x_m$, if all monomials depending only on $y_1, \dots, y_n$ are smaller than monomials depending on $x_1, \dots, x_m$. In other words:

$$x_1^{\alpha_1} \cdots x_m^{\alpha_m} \succ y_1^{\beta_1} \cdots y_n^{\beta_n}. \tag{9.2}$$

for every $\alpha = (\alpha_1, \dots, \alpha_m)$ and every $\beta = (\beta_1, \dots, \beta_n)$.

Keeping to the convention used in this entire chapter, we shall shortly write $K[\mathbb{X}; \mathbb{Y}]$ for $K[x_1, \dots, x_m; y_1, \dots, y_n]$. The condition Ineq. (9.2) reads then:

$$\mathbb{X}^\alpha \succ \mathbb{Y}^\beta, \qquad \text{for all} \quad \alpha \in \mathbb{N}_0^m, \beta \in \mathbb{N}_0^n.$$

A model example of an elimination order is just the lexicographic order on $K[\mathbb{X}]$, which is an elimination order with respect to each of the sets $\{x_1\}, \{x_1, x_2\}, \dots, \{x_1, \dots, x_{n-1}\}$.

better formulation needed

Exercise 26. Let $\preceq_x$ be a monomial order on $K[\mathbb{X}]$ and $\preceq_y$ be a monomial order on $K[\mathbb{Y}]$. Show that a relation $\preceq$, defined by the formula

$$\mathbb{X}^\alpha \mathbb{Y}^\beta \preceq \mathbb{X}^\gamma \mathbb{Y}^\delta \iff \Big(\text{either} \quad \mathbb{X}^\alpha \prec_x \mathbb{X}^\gamma \quad \text{or} \quad \mathbb{X}^\alpha = \mathbb{X}^\gamma \text{ and } \mathbb{Y}^\beta \preceq_y \mathbb{Y}^\delta\Big),$$

is an elimination order on $K[\mathbb{X}; \mathbb{Y}]$ with respect to $\mathbb{X}$.

Observation 9.8.2. *If $\preceq$ is an elimination order on $K[\mathbb{X}; \mathbb{Y}]$ with $\mathbb{X} \succ \mathbb{Y}$, then $\preceq$ induces a monomial order on the subring $K[\mathbb{Y}]$.*

Theorem 9.8.3. *Let $\preceq$ be an elimination order on $K[\mathbb{X}; \mathbb{Y}]$ with $\mathbb{X} \succ \mathbb{Y}$. If G is a Gröbner basis (with respect to $\preceq$) of an ideal $I \lhd K[\mathbb{X}; \mathbb{Y}]$, then*

$$G \cap K[\mathbb{Y}]$$

is a Gröbner basis of $I \cap K[\mathbb{Y}]$ with respect to the monomial order induced by $\preceq$ on $K[\mathbb{Y}]$.

Proof

More explicit version

If our elimination order is simply lex, then it follows from the above theorem, that successive elements of a Gröbner basis depends on fewer and fewer variables. In particular, if the algebraic set $\mathscr{Z}(I)$ associated to our ideal I is finite (i.e. the system of polynomial equations, we begun with, has only finitely many zeros with coordinates in K^{alg}), then the last element of the Gröbner basis must be a univariate polynomial. Once we know its roots, we can plug them into the other polynomials and solve the system backward. Informally speaking, the theorem says that using an elimination order, we may eliminate certain (largest with respect to selected ordering) variables from the system of polynomial equations underlying the ideal. In case of lexicographic monomial order, this "triangulates" the system.

Remark 68. One may wonder why we define elimination orders and state Theorem 9.8.3 in such a general form instead of just using lex order all the time. The reason is efficiency—Gröbner basis computations with respect to lex order tends to be extremely slow. Therefore, if it is not absolutely needed, one should rather avoid using the lexicographic order. In the next chapter (see Section) we will see how to cope with polynomial systems, having only finitely many solutions, without lex. If we do not need a fully triangulated system, but only wish to eliminate some variables, then a custom-made order (see Exercise 26) will almost certainly be a better choice.

missing

Example 115. Take a system of polynomial equation (say over $\mathbb{R}$):

$$\begin{cases} x^2+y^2+z^2-2x=0 \\ x^3-x-2y=0 \\ x-y+2z=0 \\ 5x^2-2xy+5y^2-8x=0. \end{cases}$$

Compute a reduced Gröbner basis of the ideal generated by the above polynomials with respect to the lexicographic monomial order with $z \succ y \succ x$. Thus, we are led to a triangular system

$$\left\{ \begin{array}{l} z-\frac{1}{4}x^3+\frac{3}{4}x=0 \\ y-\frac{1}{2}x^3+\frac{1}{2}x=0 \\ x^6-\frac{14}{5}x^4+\frac{29}{5}x^2-\frac{32}{5}x=0 \end{array} \right] \in \mathbb{R}[x] \left. \right] \in \mathbb{R}[x,y] \left. \right] \in \mathbb{R}[x,y,z]$$

Observe that the last polynomial depends only on x. The roots of this univariate polynomial are (approximately)

$$\{0.000, 1.46, -1.56-0.852i, -1.56+0.852i, 0.834-0.828i, 0.834+0.828i\}.$$

Back-substituting them into the other equations we find all the solutions of our system (approximately):

$$\begin{cases} x=0.000 \\ y=0.000 \\ z=0.000 \end{cases} \quad \begin{cases} x=1.46 \\ y=0.829 \\ z=-0.316 \end{cases} \quad \begin{cases} x=-1.56-0.852i \\ y=0.570-2.39i \\ z=1.07-0.770i \end{cases}$$

$$\begin{cases} x=-1.56+0.852i \\ y=0.570+2.39i \\ z=1.07+0.770i \end{cases} \quad \begin{cases} x=0.834-0.828i \\ y=-0.984-0.166i \\ z=-0.909+0.331i \end{cases} \quad \begin{cases} x=0.834+0.828i \\ y=-0.984+0.166i \\ z=-0.909-0.331i \end{cases}$$

As it was already mentioned in the introduction to Chapter 8, "back-solving" can suffer from serious accuracy loss, hence one may use the above solutions as just initial guesses for a multivariate numerical root-finder (see Section 10) that will eventually produce an answer of a desired accuracy.

Example 116. Consider again our standard example of a multivariate polynomial system Eq. (8.1):

$$\begin{cases} 3x^2 - 3xy - 10x + 3y^2 + 5 = 0 \\ x^2 - 5xy + y^2 + 6y - 3 = 0, \end{cases}$$

that we have already encountered in the previous chapter twice. This time, we will solve it using Gröbner bases. Denote the polynomials from the system, respectively by f_1 and f_2. Take an ideal $I = \langle f_1, f_2 \rangle$ associated to this system and compute its Gröbner basis with respect to the lexicographic monomial order with $x \succ y$:

$$G = \left\{ x + 12y^3 - 28y^2 + y + 9, y^4 - \frac{19}{6}y^3 + \frac{73}{36}y^2 + \frac{29}{36}y - \frac{13}{18} \right\}.$$

The last polynomial generates the intersection ideal $I \cap \mathbb{R}[y]$. Hence, it depends only on y—in fact in this case it simply equals $\operatorname{res}_x(f_1, f_2)$. The roots of this polynomial are the ordinates of the solutions to the original system. We computed them before in Chapter 8. Analogously, compute a Gröbner basis of I again with respect to the lexicographic order but this time let $y \succ x$. The basis consists of the polynomials:

$$y + 12x^3 - 68x^2 + \frac{307}{3}x - \frac{125}{3}, \qquad \text{and} \qquad x^4 - \frac{43}{6}x^3 + \frac{613}{36}x^2 - \frac{583}{36}x + \frac{46}{9}.$$

Now, the latter polynomial depends only on x, as it generates $I \cap \mathbb{R}[x]$ (it is just a scalar multiplicity of $\operatorname{res}_y(f_1, f_2)$) and its roots are the abscissas of the sought solutions. As we did before, substituting every possible pair of x and y into our original system we identify the solutions.

10

Zero-dimensional polynomial systems

Multivariate Newton–Raphson method

A

Fundamental concepts from algebra

There's al-gebra. That's like sums with letters. For... for people whose brains aren't clever enough for numbers

T. Pratchett, „Jingo"

Group, subgroup, morphisms, kernel, image, rings, subrings, ideals, isomorphism theorem, linear space

Bibliography

[1] John Abbott. Quadratic interval refinement for real roots. *ACM Commun. Comput. Algebra*, 48(1-2):3–12, 2014.

[2] A. Akritas, A. Strzeboński, and P. Vigklas. Implementations of a new theorem for computing bounds for positive roots of polynomials. *Computing*, 78(4):355–367, 2006.

[3] A. G. Akritas and A. W. Strzeboński. A comparative study of two real root isolation methods. *Nonlinear Anal. Model. Control*, 10(4):297–304, 2005.

[4] Alberto Alesina and Massimo Galuzzi. Vincent's theorem from a modern point of view. *Rend. Circ. Mat. Palermo (2) Suppl.*, (64):179–191, 2000. Categorical studies in Italy (Perugia, 1997).

[5] M. M. Artjuhov. Certain criteria for primality of numbers connected with the little Fermat theorem. *Acta Arith.*, 12:355–364, 1966/1967.

[6] A. Oliver L. Atkin. Probabilistic primality testing. Research Report 1779, INRIA, 1992.

[7] Eric Bach. *Analytic methods in the analysis and design of number-theoretic algorithms*. ACM Distinguished Dissertations. MIT Press, Cambridge, MA, 1985.

[8] Erwin H. Bareiss. Sylvester's identity and multistep integer-preserving Gaussian elimination. *Math. Comp.*, 22:565–578, 1968.

[9] Evelia García Barroso and Arkadiusz Płoski. Euclidean algorithm and polynomial equations after labatie. In Tadeusz Krasiński and Stanisław Spodzieja, editors, *Analytic and Algebraic Geometry*, pages 41–49. Łódź University Press, 2013.

[10] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in real algebraic geometry*, volume 10 of *Algorithms and Computation in Mathematics*. Springer-Verlag, Berlin, second edition, 2006.

[11] Stuart J. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Inform. Process. Lett.*, 18(3):147–150, 1984.

[12] Richard S. Bird. A simple division-free algorithm for computing determinants. *Inform. Process. Lett.*, 111(21-22):1072–1074, 2011.

[13] W. S. Brown. On Euclid's algorithm and the computation of polynomial greatest common divisors. *J. Assoc. Comput. Mach.*, 18:478–504, 1971.

[14] Yann Bugeaud and Maurice Mignotte. On the distance between roots of integer polynomials. *Proc. Edinb. Math. Soc. (2)*, 47(3):553–556, 2004.

[15] Yann Bugeaud and Maurice Mignotte. Polynomial root separation. *Int. J. Number Theory*, 6(3):587–602, 2010.

[16] Augustin-Louis Cauchy. Note III. Sur la résolution numérique des équations. In *Oeuvres complètes d'Augustin Cauchy*, volume 3 of *Série 2*. Gauthier-Villars et fils (Paris), 1847.

[17] Eng-Wee Chionh, Ming Zhang, and Ronald N. Goldman. Transformations and transitions from the sylvester to the bezout resultant. Technical report, 1998.

[18] Eng-Wee Chionh, Ming Zhang, and Ronald N. Goldman. Fast computation of the Bezout and Dixon resultant matrices. *J. Symbolic Comput.*, 33(1):13–29, 2002.

[19] George E. Collins. The calculation of multivariate polynomial resultants. *J. Assoc. Comput. Mach.*, 18:515–532, 1971.

[20] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symbolic Comput.*, 9(3):251–280, 1990.

[21] Nicolas Courtois, Gregory V. Bard, and Daniel Hulme. A new general-purpose method to multiply 3x3 matrices using only 23 multiplications. *CoRR*, abs/1108.2830, 2011.

[22] David Cox, John Little, and Donal O'Shea. *Ideals, varieties, and algorithms*. Undergraduate Texts in Mathematics. Springer, New York, third edition, 2007. An introduction to computational algebraic geometry and commutative algebra.

[23] Richard Crandall and Carl Pomerance. *Prime numbers*. Springer-Verlag, New York, 2001. A computational perspective.

[24] J. H. Davenport, Y. Siret, and E. Tournier. *Computer algebra*. Academic Press, Ltd., London, second edition, 1993. Systems and algorithms for algebraic computation, With a preface by Daniel Lazard, Translated from the French by A. Davenport and J. H. Davenport, With a foreword by Anthony C. Hearn.

[25] The Sage Developers. *Sage Mathematics Software (Version 5.6)*, YYYY. `http://www.sagemath.org`.

[26] Benjamin Fine and Gerhard Rosenberger. *The fundamental theorem of algebra*. Undergraduate Texts in Mathematics. Springer-Verlag, New York, 1997.

[27] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. Mpfr: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Softw.*, 33(2), June 2007.

[28] F. R. Gantmacher. *Applications of the theory of matrices*. Translated by J. L. Brenner, with the assistance of D. W. Bushaw and S. Evanusa. Interscience Publishers, Inc., New York; Interscience Publishers Ltd., London, 1959.

[29] C. G. Gibson. *Elementary geometry of algebraic curves: an undergraduate introduction*. Cambridge University Press, Cambridge, 1998.

[30] Klaus Glashoff. A note on elimination polynomials. Hambuger Beiträge zur Angewandten Mathematik, Reihe A, Preprint 137, 1998.

[31] Natalio H. Guersenzvaig and Fernando Szechtman. Roots multiplicity and squarefree factorization of polynomials using companion matrices. *Linear Algebra Appl.*, 436(9):3160–3164, 2012.

[32] Aaron Herman, Hoon Hong, and Elias Tsigaridas. Improving root separation bounds. *J. Symbolic Comput.*, 84:25–56, 2018.

[33] Hoon Hong. Bounds for absolute positiveness of multivariate polynomials. *J. Symbolic Comput.*, 25(5):571–585, 1998.

[34] Ellis Horowitz. *Algorithms for symbolic integration of rational functions*. PhD thesis, University of Wisconsin, 1969.

[35] Shane T. Jensen and George P. H. Styan. Some comments and a bibliography on the Laguerre-Samuelson inequality with extensions and applications in statistics and matrix theory. In *Analytic and geometric inequalities and applications*, volume 478 of *Math. Appl.*, pages 151–181. Kluwer Acad. Publ., Dordrecht, 1999.

[36] Rodney W. Johnson and Aileen M. McLoughlin. Noncommutative bilinear algorithms for 3×3 matrix multiplication. *SIAM J. Comput.*, 15(2):595–603, 1986.

[37] Michael Kalkbrener. *Three Contributions to Elimination Theory*. PhD thesis, RISC, Johannes Kepler University Linz, 1991.

[38] Victor J. Katz and Karen Hunger Parshall. *Taming the unknown*. Princeton University Press, Princeton, NJ, 2014. A history of algebra from antiquity to the early twentieth century.

[39] Michael Kerber and Michael Sagraloff. Efficient real root approximation. In *ISSAC 2011—Proceedings of the 36th International Symposium on Symbolic and Algebraic Computation*, pages 209–216. ACM, New York, 2011.

[40] Kinji Kimura. Computing the general discriminant formula of degree 17. Conference talk at Gröbner Bases, Resultants and Linear Algebra: GBRELA 2013, RISC, 2013.

[41] Israel Kleiner. *A history of abstract algebra*. Birkhäuser Boston, Inc., Boston, MA, 2007.

[42] Donald E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[43] Neal Koblitz. *A course in number theory and cryptography*, volume 114 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, second edition, 1994.

[44] Fanyu Kong, Zhun Cai, Jia Yu, and Daxing Li. Improved generalized Atkin algorithm for computing square roots in finite fields. *Inform. Process. Lett.*, 98(1):1–5, 2006.

[45] Namhun Koo, Gook Hwa Cho, and Soonhak Kwon. Square root algorithm in $\mathbb{F}_q$ for $q \equiv 2^s + 1 \pmod{2^{s+1}}$. preprint available at `http://eprint.iacr.org/2013/087.pdf`, 2013.

[46] Przemysł aw Koprowski, Kurt Mehlhorn, and Saurabh Ray. Corrigendum to "Faster algorithms for computing Hong's bound on absolute positiveness" [J. Symb. Comput. 45 (2010) 677–683] [MR2639310]. *J. Symbolic Comput.*, 87:238–241, 2018.

[47] Przemysław Koprowski. Roots multiplicity without companion matrices. *Fundamenta Informaticae*, 153(3):265–270, 2017.

[48] E. E. Kummer. Über die Ergänzungssätze zu den allgemeinen Reciprocitätsgesetzen. *J. Reine Angew. Math.*, 44:93–146, 1852.

[49] Julian D. Laderman. A noncommutative algorithm for multiplying 3×3 matrices using 23 multiplications. *Bull. Amer. Math. Soc.*, 82(1):126–128, 1976.

[50] E. Landau. Sur quelques théorèmes de M. Petrovitch relatifs aux zéros des fonctions analytiques. *Bull. Soc. Math. France*, 33:251–261, 1905.

[51] Serge Lang. *Algebra*, volume 211 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, third edition, 2002.

[52] Kenneth Lange. Hadamard's determinant inequality. *Amer. Math. Monthly*, 121(3):258–259, 2014.

[53] François Le Gall. Powers of tensors and fast matrix multiplication. In *ISSAC 2014—Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, pages 296–303. ACM, New York, 2014.

[54] Meena Mahajan and V. Vinay. A combinatorial algorithm for the determinant. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (New Orleans, LA, 1997)*, pages 730–738. ACM, New York, 1997.

[55] K. Mahler. An application of Jensen's formula to polynomials. *Mathematika*, 7:98–100, 1960.

[56] K. Mahler. An inequality for the discriminant of a polynomial. *Michigan Math. J.*, 11:257–262, 1964.

[57] Mitch Main, Micah Donor, and R. Corban Harwood. An Elementary Proof of Dodgson's Condensation Method for Calculating Determinants. *ArXiv e-prints*, July 2016.

[58] O. M. Makarov. An algorithm for multiplication of 3×3 matrices *Zh. Vychisl. Mat. i Mat. Fiz.*, 26(2):293–294, 320, 1986.

[59] O. M. Makarov. A noncommutative algorithm for multiplying 5×5 matrices using 102 multiplications. *Inform. Process. Lett.*, 23(3):115–117, 1986.

[60] Kurt Mehlhorn and Saurabh Ray. Faster algorithms for computing Hong's bound on absolute positiveness. *J. Symbolic Comput.*, 45(6):677–683, 2010.

[61] M. Mignotte. An inequality about factors of polynomials. *Math. Comp.*, 28:1153–1157, 1974.

[62] Maurice Mignotte. On the distance between the roots of a polynomial. *Appl. Algebra Engrg. Comm. Comput.*, 6(6):327–332, 1995.

[63] Gary L. Miller. Riemann's hypothesis and tests for primality. *J. Comput. System Sci.*, 13(3):300–317, 1976. Working papers presented at the ACM-SIGACT Symposium on the Theory of Computing (Albuquerque, N.M., 1975).

[64] Gary Lee Miller. *Riemann's hypothesis and tests for primality*. ProQuest LLC, Ann Arbor, MI, 1975. Thesis (Ph.D.)–University of California, Berkeley.

[65] Siguna Müller. On the computation of square roots in finite fields. *Des. Codes Cryptogr.*, 31(3):301–312, 2004.

[66] David Rea Musser. *Algorithms for polynomial factorization*. PhD thesis, University of Wisconsin, 1971.

[67] The on-line encyclopedia of integer sequences. published electronically at `https://oeis.org`.

[68] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical recipes*. Cambridge University Press, Cambridge, third edition, 2007. The art of scientific computing.

[69] Proceeding of the 2004 American Control Conference, Boston Massachusetts June 30–July 2, 2004. *Inequalities and Bounds for the Zeros of Polynomials Using Perron-Frobenius and Gerschgorin Theories*, 2004.

[70] Michael O. Rabin. Probabilistic algorithm for testing primality. *J. Number Theory*, 12(1):128–138, 1980.

[71] H. E. Rose. *A course in number theory*. Oxford Science Publications. The Clarendon Press, Oxford University Press, New York, second edition, 1994.

[72] Harvey E. Rose. *Linear algebra*. Birkhäuser Verlag, Basel, 2002. A pure mathematical approach.

[73] Günter Rote. Division-free algorithms for the determinant and the Pfaffian: algebraic and combinatorial approaches. In *Computational discrete mathematics*, volume 2122 of *Lecture Notes in Comput. Sci.*, pages 119–135. Springer, Berlin, 2001.

[74] G. Rousseau. On the quadratic reciprocity law. *J. Austral. Math. Soc. Ser. A*, 51(3):423–425, 1991.

[75] Siegfried M. Rump. Polynomial minimum root separation. *Math. Comp.*, 33(145):327–336, 1979.

[76] P. A. Samuelson. A method of determining explicitly the coefficients of the characteristic equation. *Ann. Math. Statistics*, 13:424–429, 1942.

[77] Mary Shaw and J. F. Traub. On the number of multiplications for the evaluation of a polynomial and some of its derivatives. *J. Assoc. Comput. Mach.*, 21:161–167, 1974.

[78] Jan Skowron and Andrew Gould. General complex polynomial root solver and its further optimization for binary microlenses. 2012.

[79] Michael Soltys. Berkowitz's algorithm and clow sequences. *Electron. J. Linear Algebra*, 9:42–54, 2002.

[80] Doru Ştefănescu. New bounds for positive roots of polynomials. *J.UCS*, 11(12):2125–2131 (electronic), 2005.

[81] Josef Stein. Computational problems associated with racah algebra. *Journal of Computational Physics*, 1(3):397–405, February 1967.

[82] Volker Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13:354–356, 1969.

[83] Sándor Szabó. Variants of an algorithm of J. Stein. *J. Int. Math. Virtual Inst.*, 1:1–16, 2011.

[84] Robert G. Tobey. *Algorithms for antidifferentiation of rational functions*. PhD thesis, Harvard, 1967.

[85] Elias P. Tsigaridas and Ioannis Z. Emiris. On the complexity of real root isolation using continued fractions. *Theoret. Comput. Sci.*, 392(1-3):158–173, 2008.

[86] Panagiotis S. Vigklas. *Upper bounds on the values of the positive roots of polynomials*. PhD thesis, University of Thessaly, Volos, Greece, 2010.

[87] Gilles Villard. On computing the resultant of generic bivariate polynomials. In *ISSAC'18—Proceedings of the 2018 ACM International Symposium on Symbolic and Algebraic Computation*, pages 391–398. ACM, New York, 2018.

[88] Joachim von zur Gathen and Jürgen Gerhard. Fast algorithms for Taylor shifts and certain difference equations. In *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation (Kihei, HI)*, pages 40–47. ACM, New York, 1997.

[89] Helmut Wielandt. Unzerlegbare, nicht negative Matrizen. *Math. Z.*, 52:642–648, 1950.

[90] Herbert S. Wilf. Perron-Frobenius theory and the zeros of polynomials. *Proc. Amer. Math. Soc.*, 12:247–250, 1961.

[91] F. Winkler. *Polynomial algorithms in computer algebra*. Texts and Monographs in Symbolic Computation. Springer-Verlag, Vienna, 1996.

[92] David Y.Y. Yun. On square-free decomposition algorithms. In *Proceedings of the Third ACM Symposium on Symbolic and Algebraic Computation*, SYMSAC '76, pages 26–35, New York, NY, USA, 1976. ACM.

[93] D. V. Zhdanovich. Exponent of complexity of matrix multiplication. *Fundam. Prikl. Mat.*, 17(2):107–166, 2011/12.

[94] Richard Zippel. *Effective Polynomial Computation*. Kluwer Academic Publishers, Boston, 1993.