Basics of polynomials

Application of discrete models

1 Introduction

Definition 1.1. A mathematical expression is called a **polynomial** if it contains only the following:

- constants (e.g. numbers),
- a variable (e.g. x),
- addition, subtraction, multiplication,
- exponentiation to an exponent in $\mathbb{N}$,
- parentheses.

Example. $3x^5 - 5x^3 + x + 1/2$, $x^2(x+3)^5$ and 15 are polynomials, but the following ones are not: $1/x$ (it contains division), x^{-5} (the exponent is not in $\mathbb{N}$) and $\sqrt{x}$ (square root is not allowed).

Note: this definition can be generalized to allow more than one variables, e.g. then $x^3 + xy^2 + y$ is a *bivariate polynomial*, i.e. a polynomial with two variables. Then, the above definition defines *univariate polynomials*, i.e. polynomials with one variable. But in this subject, we only deal with the latters, so we omit the qualifier "univariate".

Definition 1.2. $K[x]$ is the set of polynomials in the variable x and with constants from the set K.

Example. $\mathbb{R}[x]$ is the set of polynomials with real numbers, and $\mathbb{Z}[x]$ is its subset where the constants can only be integers. For example, $2x^2 - 5 \in \mathbb{Z}[x]$ and $\sqrt{2}x + 3/2 \in \mathbb{R}[x]$, but $\sqrt{2}x + 3/2 \notin \mathbb{Z}[x]$.

In this document, K is a number set, i.e. one of $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ or $\mathbb{C}$. (Later we will generalize it.)

The same polynomial can be written in multiple forms, for example:

$$(2x+1)^2 = 4x^2 + 4x + 1 = 4x + 1 + 4x^2 = x^5 + 1 + x - x^5 + 4x^2 + 3x.$$

If we need a unique representation, then we can use the following form:

Definition 1.3. The **canonical form** of a polynomial $p \in K[x]$ is the following:

$$p = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0,$$

where all $a_k \in K$ and $a_n \neq 0$.

Example. The canonical form of $p = (2x+1)^2$ is $p = 4x^2 + 4x + 1$.

A polynomial can be converted to canonical form by expanding the brackets, performing the multiplications in each term (e.g. $x \cdot 2x \cdot 3 = 6x^2$), combining terms with equal powers, and sorting the terms in descreasing powers of x.

Definition 1.4. In the above canonical form:

- a_k's are the **coefficients** of the polynomial (specifically, a_k is the **coefficient of degree** k);
- n is the **degree** of the polynomial – notation: $\deg p$ –;
- a_n is the **leading coefficient** of the polynomial;
- a_0 is the **constant term** of the polynomial.

Example. The coefficients of $p = 2x^3 - 6x + 4$ are 2, 0, -6 and 4 (note the 0 for the missing x^2 term), its degree is 3, its leading coefficient is 2 and its constant term is 4.

But there is a polynomial which, according to the above definition, cannot be written in canonical form: the 0 (because we required that $a_n \neq 0$). So we need to handle it separately:

Definition 1.5. $0 \in K[x]$ is the **zero polynomial**, its canonical form is 0, and its degree is $-\infty$.

(Note: it is $-\infty$ to make it smaller than of any other polynomial. It cannot be defined 0, because that would break many statements about the degree of polynomials, such as Theorem 1.7 below. Some sources leave the degree of the zero polynomial undefined, and Sage defines it to be -1.)

Definition 1.6.

- If $\deg p \leq 0$, then p is called a **constant polynomial**;
- if $\deg p = 1$, then p is called a **linear polynomial**;
- if $\deg p = 2$, then p is called a **quadratic polynomial**;
- if $\deg p = 3$, then p is called a **cubic polynomial**.

Theorem 1.7 (change of degree by arithmetic operations)**.** *If $p, q \in \mathbb{R}[x]$, then:*

1. $\deg(p+q) \leq \max(\deg p,\ \deg q)$ *(with equality if $\deg p \neq \deg q$);*
2. $\deg(p-q) \leq \max(\deg p,\ \deg q)$ *(with equality if $\deg p \neq \deg q$);*
3. $\deg(p \cdot q) = \deg p + \deg q$.

Example. If $p = 2x - 1$ and $q = x^2 + 2$, then $\deg p = 1$ and $\deg q = 2$, and indeed:

- $p + q = x^2 + 2x + 1$, so $\deg(p+q) = 2 = \max(1, 2)$,
- $p \cdot q = 2x^3 - x^2 + 4x - 2$, so $\deg(p \cdot q) = 3 = 1 + 2$.

(Note: these rules also work for the zero polynomial, e.g. $0 \cdot x^2 = 0$, and indeed, for the degrees, $-\infty + 2 = -\infty$.)

2 Polynomials in Sage

One way to calculate with polynomials in Sage is to use general symbolic expressions:

```
sage: (2*x - 1)*(x^2 + 2)
(x^2 + 2)*(2*x - 1)
sage: expand(_)
2*x^3 - x^2 + 4*x - 2
sage: type(_)
<class 'sage.symbolic.expression.Expression'>
```

(Reminder: `x` is a built-in symbolic variable, see the first document: SageMath introduction.)

But this is slightly cumbersome (e.g. it requires `expand()` after each multiplication), and not very efficient for polynomials. Instead, we can use Sage's structures that are optimized specifically for polynomials: `ZZ["x"]`, `QQ["x"]` etc., corresponding to the mathematical sets $\mathbb{Z}[x]$, $\mathbb{Q}[x]$ etc.:

```
sage: P = ZZ["x"]
sage: p = P(2*x - 1)    # convert a symbolic expression to a polynomial
sage: p
2*x - 1
sage: q = P([2, 0, 1])  # create from coefficients (starting from the constant)
sage: q
x^2 + 2
```

```
sage: p * q             # no need for expand() (and very fast)
2*x^3 - x^2 + 4*x - 2
sage: type(_)
<class 'sage.rings.polynomial.[...]'>
```

Warning: in the entered commands, `x` is still a symbolic variable, don't mix it with polynomials:

```
sage: p*(x^2 + 2)     # DON'T do this! (polynomial * symbolic expression)
(x^2 + 2)*(2*x - 1)
sage: type(_)         # the optimized polynomial structure is lost
<class 'sage.symbolic.expression.Expression'>
```

Note: as a slightly simplified explanation, Sage represents a symbolic expression as a sequence of operations, e.g. $x^2 + 2$ is represented as "square x, then add 2". This is very general, but not very efficient to calculate with. Polynomials on the other hand are represented using a sequence of coefficients, e.g. $x^2 + 2$ is represented as $2, 0, 1$. This makes much more efficient calculations possible.

To avoid accidentally using symbolic expressions, we can redefine `x` to be a polynomial:

```
sage: P = ZZ["x"]    # let P be the set of integer polynomials
sage: x = P.gen()    # let x be the variable of the polynomials in P
```

There is also a simplified syntax that defines both `P` and `x`:

```
sage: P.<x> = ZZ[]   # same as: P = ZZ["x"]; x = P.gen()
sage: P
Univariate Polynomial Ring in x over Integer Ring
sage: type(x)              # x is also redefined
<class 'sage.rings.polynomial.[...]'>
sage: (2*x - 1)*(x^2 + 2)  # now these are polynomials too
2*x^3 - x^2 + 4*x - 2
sage: var("x")             # reset x
sage: type(x)
<class 'sage.symbolic.expression.Expression'>
```

We can query several properties of polynomials:

```
sage: P.<x> = ZZ[]
sage: p = 2*x^3 - 6*x + 4
sage: p.degree()
3
sage: p.list()     # list of coefficients (starting from the constant term)
[4, -6, 0, 2]
sage: p[1]         # coefficient of degree 1
-6
sage: p.leading_coefficient()   # or: p.lc()
2
sage: p.constant_coefficient()  # or: p[0]
4
```

3 Evaluation

Definition 3.1. The **value** of the polynomial $p \in K[x]$ at c is obtained by replacing all occurences of x in p by the value c. Notation: $p(c)$. This computation is called the **evaluation** of p at c.

Example. If $p = x^2 - 3x + 5$ and $c = 2$, then $p(2) = 2^2 - 3 \cdot 2 + 5 = 3$.

Example. It is not required that $c \in K$: we can evaluate the same $p = x^2 - 3x + 5 \in \mathbb{Z}[x]$ at $c = 1/2 \notin \mathbb{Z}$ too: $p(1/2) = (1/2)^2 - 3 \cdot (1/2) + 5 = 15/4$.

We can evaluate a polynomial directly from its canonical form, i.e. $p(c) = a_n c^n + \ldots + a_1 c + a_0$ – but this is not efficient: it needs n multiplications for $a_n c^n$, $n-1$ multiplications for $a_{n-1}c^{n-1}$ etc., which is $n + (n-1) + \ldots + 2 + 1 = n(n+1)/2$ multiplications in total, plus n additions. (For example, if $\deg p = 10$, then it needs 55 multiplications and 10 additions.)

This can be improved by going from right to left, and using the previous value of c^k to calculate the next c^{k+1} with only one extra multiplication: $c^{k+1} = c^k \cdot c$. This methods needs only $2n-1$ multiplications and n additions in total (which for $\deg p = 10$ is 19 multiplications and 10 additions).

But even more efficient is **Horner's method** (or *Horner's scheme*), which turns the polynomial into a form that allows a very efficient evaluation. For example:

$$\begin{aligned} p &= 2x^4 + 3x^3 - 2x^2 - 4x + 3 = \\ &= (2x^3 + 3x^2 - 2x - 4)x + 3 = \\ &= ((2x^2 + 3x - 2)x - 4)x + 3 = \\ &= (((2x + 3)x - 2)x - 4)x + 3. \end{aligned}$$

That is, we extract x from all the terms we can, and do this again and again until all the exponents disappear. In general, this means the following transformation:

$$\begin{aligned} p &= a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_2 x^2 + a_1 x + a_0 = \\ &= ((\ldots((a_n x + a_{n-1})x + a_{n-2})x + \ldots + a_2)x + a_1)x + a_0. \end{aligned}$$

This form makes evaluation very efficient: in each step, we multiply a parenthesized subexpression (first a_n) by x (or rather, by c), then add the next coefficient to it to get the next parenthesized subexpression, and continue this until we finally get the whole value $p(c)$. This process requires only n multiplications and n additions.

Horner's method:

Input: the coefficients $a_0, a_1, \ldots, a_n$ and the value c
$b := a_n$
for $k = n-1$ *to* 0 **do**
  $b := b \cdot c + a_k$
Output: b

In each step, the value of b equals to one of the parenthesized subexpressions, e.g. after the first step, $b = a_n c + a_{n-1}$, then $b = (a_n c + a_{n-1})c + a_{n-2}$ etc.

(Note: this algorithm doesn't work directly for the zero polynomial, because it doesn't have any coefficients. But this can be fixed easily by giving it a single zero coefficient: $n = 0$ and $a_0 = 0$. In general, the algorithm doesn't require that $a_n \neq 0$, i.e. we can add redundant 0's at the end of the coefficient list without changing the result.)

In Sage, polynomials can be evaluated using parentheses, similar to the mathematical notation, e.g. `p(2)`, where `p` is a Sage polynomial. (And Sage also uses Horner's method for this.)

4 Roots

Definition 4.1. The value c is a **root** of the polynomial $p \in K[x]$ if $p(c) = 0$.

Example. $c = 1$ is a root of $p = 2x^2 - 3x + 1$, because $p(1) = 2 \cdot 1^2 - 3 \cdot 1 + 1 = 0$.

Example. Just like for evaluation, it is not required that $c \in K$: the above $p \in \mathbb{Z}[x]$ has another root, $c = 1/2 \notin \mathbb{Z}$: $p(1/2) = 2 \cdot (1/2)^2 - 3 \cdot (1/2) + 1 = 0$.

Example. $q = x^2 + 1 \in \mathbb{R}[x]$ has no real roots, only complex: $x_1 = i$ and $x_2 = -i$.

Theorem 4.2. *$c \in K$ is a root of $p \in K[x]$ if and only if $\exists q \in K[x] : p = (x - c)\, q$.*

Example. $c = 1$ is a root of the above $p = 2x^2 - 3x + 1$, and indeed: $p = (x - 1)(2x - 1)$.

Definition 4.3. The polynomial $x - c \in K[x]$ is called the **root factor** corresponding to c.

So the above theorem says that c is a root of p if and only if the root factor corresponding to c can be extracted from p.

It can happen that c is also root of the polynomial q left after the extraction $p = (x - c)\, q$. Then the same root factor can be extracted from q too: $q = (x - c)\, r$, i.e. $p = (x - c)^2\, r$, which means that $x - c$ can be extracted from p multiple times. Then we say that c is a **multiple root** of p, or more precisely:

Definition 4.4. $c \in K$ is a **root of multiplicity k** of the polynomial $p \in K[x]$ if $\exists q \in K[x]$ such that $p = (x - c)^k\, q$, but c is not a root of q. Terminology: if $k = 1$, then c is a **simple root**, if $k = 2$, it is a **double root** etc., and if $k \geq 2$, then c is a **multiple root** of p.

Example. $c = 1$ is a double root of $p = 2x^3 - 6x + 4$, because $p = (x - 1)^2\, (2x + 4)$, but 1 is not a root of $2x + 4$.

So c is a root of multiplicity k if $x - c$ can be extracted exactly k times.

A very important theorem follows from the root factor extraction:

Theorem 4.5. *A polynomial $p \in K[x] \setminus \{0\}$ of degree n can have at most n roots in K. This is true even if they are counted with multiplicities, i.e. the sum of the multiplicities of all roots is at most n.*

Example. A cubic polynomial can have at most 3 roots. Also, if it has a double root (e.g. $c = 1$ for the above p), then it can only have one more root, because the double root "counts twice".

(Proof: extract as many root factors from p as possible: $p = (x - c_1)(x - c_2) \cdots (x - c_m)\, q$, where q has no more roots. Then p can have no roots other than $c_1, c_2, \ldots, c_m$, because if $p(c) = 0$, then one of the factors must be 0, i.e. either some $c_i = c$, or q still has a root. But every extraction decreases the degree by one, so there can be at most $n = \deg p$ root factors, and so at most n roots.)

Some special cases:

- A polynomial of degree one, i.e. $p = ax + b$ can have at most one root (a simple root): $x = -b/a$ ("at most" one, because in $\mathbb{Z}$, this division may not be valid).
- A polynomial of degree zero (i.e. a nonzero constant polynomial, e.g. $p = 5$) has no roots.
- But the zero polynomial ($p = 0$) has all $c \in K$ as roots (this is why it had to be excluded from the above theorem).

Earlier, we defined the canonical form of polynomials. Now we define another important form using root factors:

Definition 4.6. The **root factor form** of a polynomial $p \in K[x]$ is the following:

$$p = a(x - c_1)(x - c_2) \cdots (x - c_m),$$

where $a, c_1, c_2 \ldots, c_m \in K$ and $a \neq 0$.

Example. The polynomial $p = 2x^3 - 6x + 4$ has the root factor form $p = 2(x - 1)^2(x + 2)$.

The point of this form is that it shows all the roots of the polynomial, with multiplicities. (And note that the constant factor $a = 2$ is the leading coefficient of the polynomial.)

But the root factor form doesn't always exist, only if the polynomial has exactly n roots in the base set K (counted with multiplicities). Note that the above theorem only guarantees that it has *at most* n roots. Also, the number of roots depends on the base set K:

Example. The polynomial $p = 16x^4 - 1$ has

- no roots in $\mathbb{Z}$;
- two simple roots in $\mathbb{Q}$ and $\mathbb{R}$: $x_1 = 1/2$ and $x_2 = -1/2$;
- and four roots in $\mathbb{C}$: $x_1 = 1/2$, $x_2 = -1/2$, $x_3 = i/2$, $x_4 = -i/2$;

therefore, the root factor form only exists in $\mathbb{C}$:

$$p = 16(x - 1/2)(x + 1/2)(x - i/2)(x + i/2).$$

In the above example, the set of complex numbers ($\mathbb{C}$) plays a special role. The following theorem says that not only in this example, but $\mathbb{C}$ is *always* that special:

Theorem 4.7 (Fundamental theorem of algebra)**.** *Every polynomial $p \in \mathbb{C}[x] \setminus \{0\}$ of degree n has exactly n roots in $\mathbb{C}$, when counted with multiplicities.*

This theorem guarantees that *any* (nonzero) polynomial with complex coefficients has a root factor form. (But calculating this form is an extremely difficult task.)

(Note: don't confuse the *fundamental theorem of algebra* with the *fundamental theorem of arithmetic* – the latter is about the unique prime factorization of integers, see: Divisors, prime numbers.)

In Sage, we can find the roots of a polynomial using the member function `roots()`:

```
sage: P.<x> = ZZ[]
sage: p = 2*x^3 - 6*x + 4
sage: p.roots()
[(-2, 1), (1, 2)]    # -2 is a simple, 1 is a double root
sage: p.roots(multiplicities=False)
[-2, 1]              # roots without multiplicities
sage: p = 16*x^4 - 1
sage: p.roots()
[]                   # no integer roots
sage: p.roots(QQ)    # find all rational roots
[(1/2, 1), (-1/2, 1)]
sage: p.roots(RR)    # in RR, Sage can only calculate numerically
[(-0.500000000000000, 1), (0.500000000000000, 1)]
sage: p.roots(CC)    # so in CC
[(-0.5000000000, 1), (0.5000000000, 1), (-0.5000000000*I, 1), (0.5000000000*I, 1)]
```

5 Lagrange interpolation

We know that a polynomial of degree n can have most n roots. This also means that if a polynomial of degree *at most* n has $n+1$ roots, then it can be nothing but the zero polynomial (0) (which has arbitrarily many roots). This statement can be generalized from roots to other values:

Theorem 5.1. *If two polynomials of degree $\leq n$ are equal in $n+1$ places, then they are equal.*

Example. If p and q have degree ≤ 2, and they are equal in three places: $p(1) = q(1)$, $p(2) = q(2)$ and $p(5) = q(5)$, then the theorem says that $p = q$. For degree ≤ 1, two values are sufficient (visually: two points define a line), and for constant polynomials, one is enough.

(The proof of the theorem is simple: the difference of the two polynomials has zeros on these $n+1$ places, i.e. roots, but then it can only be the zero polynomial.)

So a polynomial of degree n is uniquely determined by $n+1$ values.

But how can we calculate this unique polynomial from the $n+1$ values? First, consider a special case where all values are zero except one. For example, find the quadratic polynomial p for which $p(1) = 0$, $p(2) = 0$ and $p(5) = 8$. The first two conditions can be handled easily: $p := (x-1)(x-2)$. But the third one is not yet satisfied: $p(5) = (5-1)(5-2) = 12 \neq 8$. So we multiply this polynomial by a constant c to make it true, i.e. $c \cdot p(5) = 8$, and the solution of this is $c = 8/p(5) = 8/12 = 2/3$. Note that this transformation not only fixes the third value, but it also preserves the zeros. So let p be actually $p := 2/3 \cdot (x-1)(x-2) = 2x^2/3 - 2x + 4/3$.

But what if the values are not zeros? For example, which quadratic polynomial p has $p(1) = 4$, $p(2) = 2$ and $p(5) = 8$? For this, we use the above method three times, and combine the results:

$$p_1(1) = 4,\ p_1(2) = 0,\ p_1(5) = 0 \implies p_1 = 4 \cdot \frac{(x-2)(x-5)}{(1-2)(1-5)} = x^2 - 7x + 10$$

$$p_2(1) = 0,\ p_2(2) = 2,\ p_2(5) = 0 \implies p_2 = 2 \cdot \frac{(x-1)(x-5)}{(2-1)(2-5)} = -\frac{2}{3}x^2 + 4x - \frac{10}{3}$$

$$p_3(1) = 0,\ p_3(2) = 0,\ p_3(5) = 8 \implies p_3 = 8 \cdot \frac{(x-1)(x-2)}{(5-1)(5-2)} = \frac{2}{3}x^2 - 2x + \frac{4}{3}$$

$$p(1) = 4,\quad p(2) = 2,\quad p(5) = 8 \implies p = p_1 + p_2 + p_3 \qquad = x^2 - 5x + 8$$

This method is called **Lagrange interpolation**. In general:

Theorem 5.2 (Lagrange interpolation)**.** *If a polynomial p of degree $\leq n$ takes the following values on given $n+1$ different places: $p(x_1) = y_1$, $p(x_2) = y_2$, ..., $p(x_{n+1}) = y_{n+1}$, then:*

$$p = \sum_{i=1}^{n+1} y_i \prod_{\substack{j=1 \\ j \neq i}}^{n+1} \frac{x - x_j}{x_i - x_j}.$$

The interpolation doesn't work in $\mathbb{Z}[x]$, only in $\mathbb{Q}[x]$, $\mathbb{R}[x]$ etc., because it involves divisions (see the fractions in the above example). But then it works for *any* $x_1, \ldots, x_{n+1}$ and $y_1, \ldots, y_{n+1}$ values (provided that all x_i's are different).

Sage has a built-in function for Lagrange interpolation:

```
sage: P.<x> = QQ[]    # not ZZ[]!
sage: P.lagrange_polynomial([(1,4), (2,2), (5,8)])
x^2 - 5*x + 8
```

6 Shamir's secret sharing (SSS)

Assume that we have a secret piece of information (a number, e.g. an encryption key) that we need to share among multiple people (e.g. among the employees of a company).

The simplest way to do this is to give the secret to all participants. But then the obvious problem is that if any of them misuses it, or it is stolen from them, then the whole secret is compromised. We need a solution where no single person can access the secret.

The opposite approach is to split the secret into equal parts, and give each participant a single part. Then none of them can individually misuse their own part – not even a small group of them can do that –, so the secret can only be restored when all participants are involved. But the problem with this is that if any of them loses their part (or they are temporarily unavailable), then the common secret cannot be restored.

But there is also another problem with this approach: altough none of the participants have the *whole* secret, they still have a part of it, which brings them one step closer to finding out the secret using brute force. For example, if a number lock has a four-digit code, then someone who doesn't know anything about the code needs to try $10^4 = 10000$ possibilities to open it (in the worst case). But if someone knows one digit of the code, then they only need to try $10^3 = 1000$ possibilities, which is ten times faster. With two digits, it is hundred times faster.

The ideal solution would need k participants out of the n to restore the secret (e.g. three out of five), so that any k participants would be able to calculate the secret, but $k-1$ or less people not only couldn't recover the whole secret, but they couldn't even get one step closer to finding it out than an outsider with no information at all.

A solution for this is **Shamir's secret sharing** (SSS).

Note: the method as described below is not yet completely secure, but this way it is easier to understand for the first time; the improved version will be presented in the next document.

Let n be the number of participants, and k the threshold to restore the secret ($1 \leq k \leq n$). Let $S \in \mathbb{Z}$ be the secret to be shared, and let's "hide" it inside a polynomial of degree $k-1$ as follows:

$$p = a_{k-1}x^{k-1} + \ldots + a_2x^2 + a_1x + S,$$

where the constant term is the secret S, and the other coefficients $(a_1, \ldots, a_{k-1})$ are randomly chosen integers. This polynomial is then evaluated in n different places, e.g. $p(1), p(2), \ldots, p(n)$, and each participant is given one of these values (a pair (x_i, y_i)), called a *secret share*. Then any k participants can use their k secret shares to reconstruct the original polynomial using Lagrange interpolation (see above), and thus get its constant term, S. But no group of less than k people can restore the polynomial, because no matter how they fill the missing values, they get different valid-looking polynomials each time, and they have no way of knowing which one is the real one.

Let's see an example for $n = 5$ participants and threshold $k = 3$. Let the secret be the value $S = 67$, and generate two more random coefficients, e.g. $a_1 = 38$ and $a_2 = 13$, to construct the quadratic polynomial $p = 13x^2 + 38x + 67$. Then evaluate p in five places:

$$p(1) = 118, \quad p(2) = 195, \quad p(3) = 298, \quad p(4) = 427, \quad p(5) = 582,$$

and give each participant one of these values. Then any three of them can restore S, e.g. the 1st, 2nd and 5th can do this as follows (using Lagrange interpolation, see above):

$$p = 118 \cdot \frac{(x-2)(x-5)}{(1-2)(1-5)} + 195 \cdot \frac{(x-1)(x-5)}{(2-1)(2-5)} + 582 \cdot \frac{(x-1)(x-2)}{(5-1)(5-2)} = 13x^2 + 38x + 67.$$

But we don't actually need to calculate the whole polynomial, only its constant term, S, which is the same as $p(0)$ (i.e. the evaluation of p at $x = 0$), which can be calculated much faster:

$$S = p(0) = 118 \cdot \frac{(-2)(-5)}{(1-2)(1-5)} + 195 \cdot \frac{(-1)(-5)}{(2-1)(2-5)} + 582 \cdot \frac{(-1)(-2)}{(5-1)(5-2)} = 67.$$

6.1 Problem with SSS

But the method as described above still has a flaw. If two participants cooperate, then while they can't recover the whole secret, they can still figure out some properties of it. For example, if they know $p(1)$ and $p(5)$, then they can use the fact that the polynomial is quadratic with integer coefficients, i.e. it can be written as $p = ax^2 + bx + S$ with $a, b \in \mathbb{Z}$, to calculate the following:

$$\begin{aligned} p(5) &= 25a + 5b + S = 582 \\ p(1) &= a + b + S = 118 \\ p(5) - p(1) &= 24a + 4b = 464 \quad \Longrightarrow \quad b = 116 - 6a \\ \Longrightarrow \quad p(1) &= a + (116 - 6a) + S = 118 \quad \Longrightarrow \quad S = 5a + 2. \end{aligned}$$

So they learn that S gives the remainder 2 when divided by 5, i.e. $S \equiv 2 \pmod 5$, which means that if they want to find S by brute force, they only need to check fives times less possibilities.

This problem comes from the fact that the coefficients are integers, where division is not always possible. If they were arbitrary rational numbers, then the above equation $S = 5a + 2$ would give no information about S at all, because for any S, there would be an $a \in \mathbb{Q}$ that satisfies this equation, namely $a = (S - 2)/5$. But in $\mathbb{Z}$, this division is not always possible, which helps an attacker to reduce the set of possible values.

Then why don't we use rational numbers for the coefficients? For many reasons: it is inconvenient to calculate with fractions, the secret value is already an integer, and random number generation would also be problematic (e.g. what distribution do we use at all?).

What we would need is a "structure" where division is always possible like in $\mathbb{Q}$, but where calculation and random number generation is easier, like in $\mathbb{Z}$. A solution for this will be presented in the next document.

7 Author

This document was written by Uray M. János, partly using the following sources:

[1] https://compalg.elte.gitlab-pages.hu/dimoa-web/tematika/dimoa (Hungarian)
[2] https://en.wikipedia.org/wiki/Polynomial
[3] https://en.wikipedia.org/wiki/Lagrange_polynomial
[4] https://en.wikipedia.org/wiki/Shamir%27s_secret_sharing
[5] A. Shamir (1979): How to share a secret. *Communications of the ACM*, 22 (11), pp. 612–613. https://dl.acm.org/doi/10.1145/359168.359176

Errors and suggestions can be reported here: uray.janos@inf.elte.hu.