Computational mathematics focused on number theory

Aurel Page

5/06/2022 - 14/06/2022
SuSAAN - Nesin math village - Izmir, Turkey

Inria Bordeaux / IMB

Introduction

What is this course about?

- Computational mathematics with a focus on algebra and number theory
- Practical point of view: toolbox.
- If you are more of a theory person: avoid common pitfalls when doing computations.
- If you are more of a computation person: improve and widen your algorithmic skills.

Lectures

1. Complexity and arithmetic operations
2. Algebraic number theory
3. Linear algebra and lattices
4. Other algorithmic techniques

General advice: pseudocode

What is pseudocode?

- On paper.
- **Enough information** so that someone who does not know the algorithm could implement it.
- **No low-level details** (loop indices, temporary variables etc).
- Sentences allowed.
- Should be independent of any programming language.

Before implementing an algorithm:

- Explain the idea to another person.
- Write pseudocode.

Pseudocode: example

Let M be a square $m \times m$ matrix.

Pseudocode:

- $G \leftarrow$ graph: vertices $\{1, \ldots, m\}$ and edge (i, j) if $M_{i,j} \neq 0$.
- find a triangle T in G.
- ...

Not pseudocode:

- for $i = 1$ to m:
- for $j = 1$ to m:
- for $k = 1$ to m:
- if $M[i, j]! = 0$ && $M[i, k]! = 0$ && $M[j, k]! = 0$:
- $C := [i, j, k]$
- break i
- ...

General advice: tests

Before, while and after you implement an algorithm, write **tests**.

- Cases that you solved by hand.
- Cases that you can compute independently.
- Limit cases (some value is 0 or maximal, some set is empty, etc).
- Tests that exercise every part of the code.
- Try to produce tests that make your algorithm fail.

General advice: team up

Team up!

- If you have theoretical skills: with someone who can do computations that you cannot.
- If you have computational skills: with someone who cannot do computations that you can.
- You can do both!

Benedict Gross:
”I used to use low level languages like Fortran, Algol, and C. Now I use high level languages like Buhler, Elkies, and Stein.”

Pari/GP

- Computer algebra software specialised in number theory.
- Originally developed by Henri Cohen and his co-workers in Bordeaux in the 80's.
- Currently maintained by Karim Belabas and Bill Allombert.
- Standalone, lightweight, free software.
- C librari (Pari) and calculator with a script language (GP).
- Easy to use and fast.

Exercise sessions

Participants:

- Group by pairs.
- Both should write code, alternatively.
- The person who does not write code should prepare tests on paper.

Exercises:

- Basic exercises
- Advanced exercises
- Exploration

Plan

Today:

1. Complexity
2. Arithmetic operations

Complexity

What is complexity?

The **complexity** of an algorithm is the asymptotic dependence of the running time on the input size.

It depends on the algorithm (and not just on the desired output) but not on the implementation.

Big O notation

Let $f, g\colon \mathbb{N} \to \mathbb{R}_{>0}$.

Definition: We write $f = O(g)$ if there exists $N, C > 0$ such that

$$f(n) \leq C \cdot g(n) \text{ for all } n \geq N.$$

Example: $7n + \sqrt{n} = O(n)$.

Definition: We write $f = \tilde{O}(g)$ if for every $\varepsilon > 0$ we have

$$f = O(g^{1+\varepsilon}).$$

Example: $n \log(n) = \tilde{O}(n)$.

Size of input

The size of input always means its **bit size**, that is, the number of binary symbols necessary to represent it.

Examples:

- The size of an integer $N \geq 0$ is

$$\lceil \log(N+1)/\log(2) \rceil = O(\log N)$$

  (number of base 2 digits).
- The size of an array is the sum of the sizes of its entries.

Elementary operations

By default, complexity measures the number of **bitwise operations**. However, sometimes we express the complexity in terms of other **elementary operations** that should be specified.

Example: number of ring operations $(+, -, \times)$ in a ring R. The bit complexity will then depend on the cost of the individual ring operations.

Complexity example

Selection sort:

- Input: an array T of length n.
- Output: the array T, sorted by increasing entries.

1. for $i = 1$ to n:
2. $\quad j \leftarrow$ position of the minimum entry among $T[i], \ldots, T[n]$.
3. $\quad$ Swap $T[i]$ and $T[j]$.

Complexity: $O(n^2)$ elementary operations: $O(n^2)$ comparisons and $O(n)$ swaps.

Note: there are sorting alorithms with complexity $O(n \log n)$.

Other relevant parameters

Sometimes (especially for mathematical algorithms), **other parameters** are more relevant than the **size of input**, and we express the complexity in terms of them.

Example: in numerical algorithms, the desired precision of the output is usually a relevant parameter.

Input representation

Warning: sometimes there are several natural ways to represent the input, of very different input sizes.

Examples:

- Integer: unary vs binary representation.
- Finite group: multiplication table vs generators as permutations.
- Polynomial: dense vs sparse representation.

Practical complexity: classes

Let n denote the size of the input.

- **Fast**: $\tilde{O}(n)$ (quasi-linear)
- **Medium**: $O(n^C)$ for some $C > 1$ (polynomial)
- **Slow**: $O(2^{n^\alpha})$ for some $0 < \alpha < 1$ (subexponential)
- **Very slow**: $O(2^{Cn})$ for some $C > 0$ (exponential)

Practical complexity: prediction

We can use complexity to **predict the running time** of an implementation.

- Measure the running time on small examples.
- Replace $O(f(n))$ by $C \cdot f(n)$ where C matches the measured running time.
- Use $C \cdot f(n)$ to predict running times.

The measured running time should not be too small to get meaningful predictions.

Predicting running times: example

Assume we have an algorithm of complexity $O(n^2)$ where n is the input size.

- With $n = 100$, we measure 1s. We estimate $C = 10^{-4}$.
- With $n = 200$, we predict 4s, but we measure 3s.
  We are not in the asymptotic regime yet.
  We re-estimate $C = 7.5 \cdot 10^{-5}$.
- With $n = 400$, we predict 12s. We measure 11.5s. Great!

Probabilistic algorithms

Some algorithms use a random number generator (RNG): they are called **probabilistic algorithms**.

Warnings:

- For a fixed input, the running time could depend on the random values.
- Complexity will be expressed ”in expected time” or ”with a certain probability”.
- For some algorithms, there are several valid outputs and the returned one depends on the random values.

One can usually fix the value of the random seed used by the RNG to make the results reproducible.

Theory and practice

Sometimes the algorithms used in practice do not have the best theoretical complexity:

- the **hidden constants** could be too large for the asymptotically faster algorithm to be faster in real instances, or
- the algorithm is complicated and is **not implemented** yet.

In such cases, I will mention the two complexities.

Memory: space complexity

Another important ressource is **memory**.

The dependence of the maximum size of the data used by the algorithm during its execution on the size of the input is the **space complexity**.

As before, it can be measured in bits (by default) or in other ”elementary units” to be specified.

Space complexity: example

Let R be a ring. Define a sequence by $u_1 = a \in R$, $u_2 = b \in R$, and for $k > 2$, $u_k = u_{k-1}^2 + 5u_{k-2}^3$.

Problem: Given (a, b, n), compute u_n.

Algorithm A:

1. Initialise an array T of length n, with first two entries a, b.
2. Iteratively compute $T[i] = u_i$ from the previous values.

Time complexity: $O(n)$. Space complexity: $O(n)$ ring elements.

Algorithm B:

1. Initialise a pair $P = (a, b)$.
2. Iteratively update the pair so that $P = (u_{i-1}, u_i)$ at step i.

Time complexity: $O(n)$. Space complexity: $O(1)$ ring elements.

Arithmetic operations

Representation of arithmetic objects

Every object is built from arrays of machine integers (usually 64 bits).

- Multiprecision (MP) integers: array representing the coefficients in base $B = 2^{64}$.
- Multiprecision floating-point real: a MP integer $2^{n-1} \geq N < 2^n$ (mantissa) and an exponent e representing $N \cdot 2^{e-n}$
- Polynomials: array of numbers representing the coefficients.
- Vectors: arrays of numbers; matrices: arrays of vectors.
- etc.

Addition

Addition of n-bits numbers, polynomials, matrices: **Fast** $O(n)$.

Warning: this is not true for $\mathbb{Q}$, as they require multiplications and GCDs of integers.

Addition: complexity

Addition of two n-digits integers $\sum_{i=0}^{n-1} N_i B^i$ and $\sum_{i=0}^{n-1} M_i B^i$.

1. Compute the $n+1$ digits of $A+B$, starting from the least significant ones.
2. At step i, compute $(A+B)_i = A_i + B_i + C$ where C is the carry from the previous addition.

Each step is $O(1)$ elementary operations, and there are at most $n+1$ steps: total time $O(n)$.
Only need to store C in addition to the output: memory $O(1)$ excluding the output.

Multiplication

- Integer and polynomial multiplication: **Fast** $\tilde{O}(n)$.
- $k \times m$-matrix - k-vector multiplication ($n = k(m+1)$): **Fast** $O(km) = O(n)$.
- matrix-matrix multiplication: **Medium** (more details later).

Integer multiplication: schoolbook

Multiplication of two n-digits integers $\sum_{i=0}^{n-1} N_i B^i$ and $\sum_{i=0}^{n-1} M_i B^i$.

1. Initialise all $2n+1$ digits of the product AB to 0.
2. For every pair (i, j): add $A_i \cdot B_j$ to $(AB)_{i+j}$.
3. Propagate the carry.

We perform $O(1)$ operations for each pair of coefficients: total $O(n^2)$ (**Medium**).

Fast multiplication

For simplicity, multiplication of two degree $n = 2^k$ polynomials $P, Q \in \mathbb{C}[X]$.

Idea: **evaluate** both at $2n$ points, multiply the evaluations, and **interpolate**.

Need good evaluation points: 2^k-th roots of unity $\zeta \rightsquigarrow$ we will be able to use a **divide-and-conquer** method.

- Write $P(X) = P_0(X^2) + XP_1(X^2)$.
- We have $P(\zeta) = P_0(\zeta^2) + \zeta P_1(\zeta^2)$.
- P_0 and P_1 have degree 2^{k-1} and ζ^2 is a 2^{k-1}-th root of unity.

Fast multiplication

Fast Fourier transform (FFT): $P \mapsto$ all $P(\zeta)$.

1. Compute P_0 and P_1.
2. Recursively compute all $P_0(\zeta')$ and $P_1(\zeta')$.
3. Recover all the $P(\zeta)$: each $P_i(\zeta')$ contributes to the two $P(\zeta)$ such that $\zeta^2 = \zeta'$.

The number $T(n) = T(2^k) = u_k$ of complex number operations satisfies

$$T(n) = 2T(n/2) + O(n), \text{ i.e. } u_k = 2u_{k-1} + O(2^k).$$

Solution $u_k = O(k2^k)$, i.e. $T(n) = O(n \log n)$.

Fast multiplication

Solution $T(n) = O(n \log n)$ for evaluation.

Fourier transform is almost an involution: same time for interpolation.

But we need to take into account the cost of **complex numbers operations at suitable precision**. Alternative, work in a suitable finite field. End result (Schönage–Strassen '71): $O(n \log n \log \log n)$ (Practice).

Harvey–van der Hoeven 2021: $O(n \log n)$ (Theory).

Matrix multiplication: schoolbook

Multiplication of two $m \times m$ matrices A and B (size $2m^2$).

1. Initialise the coefficients of AB to 0.
2. For every triple (i, j, k): add $A_{i,k}B_{k,j}$ to $(AB)_{i,j}$.

We perform $O(1)$ operations for every triple of coefficients: total $O(m^3) = O(n^{3/2})$ (**Medium**).

Currenly unknown how to obtain $\tilde{O}(n) = \tilde{O}(m^2)$ (**Fast**).

Faster matrix multiplication

Strassen '69: formula for the product of two 2×2 matrices with 7 **multiplications** and many additions, valid even for **noncommutative coefficients**.

Strassen's algorithm for $m = 2^k$:

1. Subdivide A and B into $m/2 \times m/2$-blocks.
2. Compute the 7 Strassen products recursively.
3. Recover the product AB blockwise.

The number $T(m) = T(2^k) = u_k$ of operations satisfies

$$T(m) = 7T(m/2) + O(m^2), \text{ i.e. } u_k = 7u_{k-1} + O(4^k).$$

Solution $u_k = O(7^k)$.

Faster matrix multiplication

Strassen (Practice):
Recall $m = 2^k$.
$T(m) = 7^k = O(m^{\log 7/\log 2}) = O(m^{2.81\ldots}) = O(n^{1.40\ldots})$.

Best known (Theory): $\tilde{O}(m^{2.37\ldots}) = \tilde{O}(n^{1.19\ldots})$.

Division

- Integer or polynomial division with remainder: **Fast** $\tilde{O}(n)$.
- Matrix division: **Medium** $\tilde{O}(m^{\omega})$.

Greatest common divisor

Greatest common divisor of integers or polynomials: **Fast** $\tilde{O}(n)$.

Variant: resultant.

Resultant

Let R be a ring and $P, Q \in R[X]$. The **resultant** $\mathrm{Res}(P, Q) = \mathrm{Res}_X(P, Q) \in R$ satisfies:

- $\mathrm{Res}(P, Q)$ has a polynomial expression in the coefficients of P and Q.
- If R is a field containing all roots of P, then

$$\mathrm{Res}(P, Q) = c^{\deg Q} \prod_{P(\alpha)=0} Q(\alpha),$$

  where c is the leading coefficient of P.
- $\mathrm{Res}(P, Q) \in R[X]P + R[X]Q$.
- $\mathrm{Res}(Q, P) = (-1)^{\deg P \cdot \deg Q} \mathrm{Res}(P, Q)$.

Resultant

Resultant computation: **Fast** $\tilde{O}(n)$.

- Computations with numbers α represented by P such that $P(\alpha) = 0$.
  Example: $\mathrm{Res}_X(P(X), Q(Y - X))$ has roots $\alpha + \beta$ where $P(\alpha) = Q(\beta) = 0$.
- Elimination.

Exponentials

- Over $\mathbb{C}$: **Fast** quasi-linear in size of input and precision.
- Over a ring R: $(a, N) \mapsto a^N$.
  Write $N = \sum_{i=0}^{n} N_i 2^i$ with $N_i \in \{0, 1\}$ and use:

$$a^N = (((a^{N_n})^2 \cdot a^{N_{n-1}})^2 \cdots)^2 \cdot a^{N_0}.$$

  Uses $O(n)$ operations in R (**Fast**).

Logarithms

- Over $\mathbb{C}$: **Fast** quasi-linear in size of input and precision.
- Over a ring R: **discrete logarithm problem** $(a, a^N) \mapsto N$?
- Discrete logarithm in an n-bits finite field of size $q = p^e$ ($q \approx 2^n$): **Slow** $\exp(\tilde{O}(n^{1/3}))$.
  But quasi-polynomial (close to **Medium**) for very small p.

Factorisation

Problem: write integer or polynomial as product of primes or irreducibles.

In short:

- Integers: **Slow** (subexponential time).
- Polynomials (over $\mathbb{R}$, $\mathbb{C}$ and finitely generated fields): **Medium** (polynomial time).

Factorisation of integers

n-bits integer N: recall $n = O(\log N)$.

Easy special cases:

- Detection of powers: **Fast** $\tilde{O}(n)$.
- Detection of primes: **Medium** $\tilde{O}(n^6)$.

Hard cases:

- Trial division algorithm: **Very slow** $\tilde{O}(\sqrt{N}) = \tilde{O}(2^{n/2})$.
- Find all b-bits prime factors $p \mid N$: **Slow** $\exp(\tilde{O}(\sqrt{b} + \log n))$.
- Completely factor N: **Slow** $\exp(\tilde{O}(n^{1/3}))$.

Open problem: find the largest square factor faster than complete factorisation?

Squarefree factorisation of polynomials

Squarefree factorisation: write

$$P = P_1 P_2^2 \dots P_k^k,$$

where $P_1, \dots, P_k$ are squarefree.

Use $\gcd(P, P') = P_2 P_3^2 \dots P_k^{k-1}$: **Fast** $\tilde{O}(n)$ (Yun '76).

Factorisation of polynomials over finite fields: distinct degree factorisation

Let $P \in \mathbb{F}_q[X]$ be squarefree.
Write

$$P = P_1 P_2 \dots P_k,$$

where all irreducible factors of P_i have degree i.

$$X^{q^i} - X = \prod_{\alpha \in \mathbb{F}_{q^i}} (X - \alpha) \Longrightarrow \prod_{k|i} P_k = \gcd(X^{q^i} - X, P).$$

Compute this gcd by first computing $(X \bmod P)^{q^i}$: **Medium** (polynomial time).

Factorisation of polynomials over finite fields: equal degree factorisation

Let $P \in \mathbb{F}_q[X]$ be squarefree

$$P = P_1 \dots P_k$$

with all P_i irreducible of degree d.
Note

$$R = \mathbb{F}_q[X]/(P) \cong \prod_{i=1}^{k} \mathbb{F}_q[X]/(P_i) \cong \mathbb{F}_{q^d}^k.$$

If $r \in R$, then $r^{(q^d-1)/2} \mapsto (\pm 1, \pm 1, \dots, \pm 1)$.
Take random $r \in R$ and compute $\gcd(r^{(q^d-1)/2} - 1, P)$: **Medium** (expected polynomial time probabilistic algorithm).

Computational mathematics focused on number theory

Aurel Page

5/06/2022 - 14/06/2022
SuSAAN - Nesin math village - Izmir, Turkey

Inria Bordeaux / IMB

Last lecture: complexity

Complexity: running time as a function of size n of input.

- **Fast**: $\tilde{O}(n)$ (quasi-linear)
- **Medium**: $O(n^C)$ for some $C > 1$ (polynomial)
- **Slow**: $O(2^{n^\alpha})$ for some $0 < \alpha < 1$ (subexponential)
- **Very slow**: $O(2^{Cn})$ for some $C > 0$ (exponential)

Last lecture: arithmetic operations

- Basic operations on integers and polynomials: **Fast**.
- Basic operations on matrices: **Medium**.
- Factorisation of polynomials: **Medium**.
- Factorisation of integers: **Slow**.

Plan

Today:

1. Reconstruction
2. Algebraic number theory

Reconstruction

What is reconstruction?

Recover an object from **specialisations** or **approximations**.

Examples: $N \in \mathbb{Z}$

- $\tilde{N} \in \mathbb{R}$ with $|N - \tilde{N}| < \frac{1}{2} \Longrightarrow N = \lfloor \tilde{N} \rceil$.
- $M = N \bmod D$ and $|N| < \frac{D}{2} \Longrightarrow N$ is the lift of M in $(-\frac{D}{2}, \frac{D}{2}]$.

Why?

- Control the **size** of objects during a computation.
- **Better tools** in the specialised or approximate version.

Example: cyclotomic polynomials

For $m \in \mathbb{Z}_{\geq 1}$

$$\Phi_m(X) = \prod_{\zeta}(X - \zeta) \in \mathbb{Z}[X]$$

where ζ ranges over primitive m-th roots of unity.

Use $\zeta = \exp(\frac{2\pi ik}{m}) \in \mathbb{C}$ for k coprime to m.

Chinese remainder theorem

If $N = \prod_{i=1}^{k} N_i$ with N_i pairwise coprime, then

$$\mathbb{Z}/N\mathbb{Z} \cong \prod_{i=1}^{k} \mathbb{Z}/N_i\mathbb{Z}$$

as rings, where $x \mapsto (x \bmod N_1, \ldots, x \bmod N_2)$.

Inverse map (reconstruction): **Fast** $\tilde{\mathrm{O}}(\log N)$.

Interpolation

If $x_0, \ldots, x_n \in K$ are pairwise distinct, then

$$K[X]_{\leq n} \cong K^n,$$

as K-vector spaces, where $P \mapsto (P(x_0), \ldots, P(x_n))$.

Inverse map (reconstruction): **Fast** $\tilde{\mathrm{O}}(n)$.

Rational number reconstruction

If $x = a/b \in \mathbb{Q}$ and $|x - \tilde{x}| < \frac{1}{|b|}$, then a/b appears in the **continued fraction** expansion of $\tilde{x}$.

Expansion computation (reconstruction): **Fast**.

Rational fraction reconstruction

If $F = A/B \in K(X)$ and $F - \tilde{F} = O(X^n)$ with $\tilde{F} \in K[[X]]$ and $\deg A + \deg B \leq n$, then F can be recovered from **Padé approximants**.

Approximant computation (reconstruction): **Fast** $\tilde{O}(n)$.

Kronecker substitution

- If $P = \sum_i P_i X^i \in \mathbb{Z}[X]$ with $|P_i| < \frac{B}{2}$, then P can be recovered from $P(B)$.
- If $P = \sum_i P_i(X) Y^i \in R[X][Y]$ with $\deg P_i < d$, then P can be recovered from $P(X^d) \in R[X]$.

Reconstruction: **Fast**.

Algebraic number theory

Number fields and function fields

Number field F	Function field $\mathbb{F}(\mathcal{C})$
$\mathbb{Q} \subset F$	$\pi : \mathcal{C} \to \mathbb{P}^1$
$\mathbb{Z}_F$	$\mathcal{O}(\mathcal{C} \setminus \pi^{-1}(\infty))$
Ideal	Divisor
Class group $\mathsf{Cl}(F)$	Jacobian $\mathsf{Jac}(\mathcal{C})$

Complexity parameters

Most significant parameters:

- Degree $d = [F : \mathbb{Q}]$.
- Discriminant Δ_F.

We have $d = O(\log \Delta_F)$, and this is optimal.

Main problems

Given a number field F, compute

1. the **ring of integers** $\mathbb{Z}_F$,
2. the **Galois group** $\mathrm{Gal}(\tilde{F}/F)$ of the Galois closure $\tilde{F}$ of F,
3. the **class group** $\mathrm{Cl}(F)$, and
4. the **unit group** $\mathbb{Z}_F^\times$.

Algebraic number theory: Number fields

Defining polynomial

We can always write

$$F = \mathbb{Q}[X]/(P) = \mathbb{Q}(\alpha)$$

where $P(\alpha) = 0$, and $P \in \mathbb{Z}[X]$ is monic and **irreducible** of degree $d = [F : \mathbb{Q}]$.

This is how we will specify a number field.

There always exists a defining polynomial of size $O(d \log \Delta_F)$.

Embeddings between number fields

A ring homomorphism between number fields is always an **embedding**, and is an **isomorphism** if and only if they have the same degree.

The homomorphisms

$$\mathbb{Q}[X]/(P) = \mathbb{Q}(\alpha) \longrightarrow K$$

are exactly given by $\alpha \mapsto \beta$, where $\beta \in K$ is a root of P.

We are reduced to finding roots of polynomials over number fields: **Medium**.

Extension

Let $Q \in \mathbb{Z}_F[Y]$ be monic and irreducible.
Let $K = F[Y]/(Q) = F(\beta)$ where β is a root of Q.

There exists $k \in \mathbb{Z}$ such that

$$K = \mathbb{Q}(\beta + k\alpha).$$

Defining polynomial for K: $\mathrm{Res}_X(P, Q(Y - kX))$ **Fast**.

Elements of number fields

We can represent an element of F uniquely by a polynomial in $\mathbb{Q}[X]$ of degree less than d.

All arithmetic operations reduce to polynomials: **Medium**. **Division** is the worst because of coefficient growth.

Trace and norm

Let $\lambda = R(\alpha) \in F$, and let $m_\lambda : F \to F$ be defined by $x \mapsto \lambda x$.

Define:

- The trace $\mathrm{Tr}_{F/\mathbb{Q}}(\lambda) = \mathrm{Tr}(m_\lambda)$: **Fast.**
- The norm $N_{F/\mathbb{Q}}(\lambda) = \det(m_\lambda) = \mathrm{Res}_X(P, R)$: **Fast.**

Algebraic number theory: Ring of integers

Orders and discriminants

An **order** $\mathcal{O}$ is a subring of F that is generated over $\mathbb{Z}$ by a basis $(w_i)_i$ of F.

Example: $\mathcal{O} = \mathbb{Z}[\alpha]$.

Every order $\mathcal{O}$ satisfies $\mathcal{O} \subset \mathbb{Z}_F$: the ring of integers is the maximal order.

Discriminant: $\mathrm{disc}(\mathcal{O}) = \det(\mathrm{Tr}(w_i w_j)) \in \mathbb{Z} \setminus \{0\}$.

- If $\mathcal{O} \subset \mathcal{O}'$ then $\mathrm{disc}(\mathcal{O}) = \mathrm{disc}(\mathcal{O}')[\mathcal{O}' : \mathcal{O}]^2$.
- $\Delta_F = \mathrm{disc}\, \mathbb{Z}_F$.
- $\mathrm{disc}(\mathbb{Z}[\alpha]) = \mathrm{Res}(P, P')$: **Fast**.

To compute $\mathbb{Z}_F$, start with $\mathcal{O} = \mathbb{Z}[\alpha]$ and increase it.

Saturation

Let p be a prime.
The **p-saturation** of a subgroup H of an abelian group $(G, \times)$ is

$$\{g \in G \mid g^{p^k} \in H \text{ for some } k \geq 0\}.$$

This is also the smallest subgroup $H \subset S \subset G$ such that $[G : S]$ is not divisible by p.

In additive notation $B \subset (A, +)$:

$$\{a \in A \mid p^k a \in B \text{ for some } k \geq 0\}.$$

Slow algorithm: for every $b \in B/pB$, test if $\frac{1}{p}b \in A$? Repeat.

Ring of integers and factorisation

Start with $\mathcal{O} = \mathbb{Z}[\alpha]$.

We reduce to two steps:

1. Find the p such that p divides $[\mathbb{Z}_F : \mathcal{O}]$:
   Factor $\mathrm{disc}(\mathcal{O})$ **Slow**.
2. For each p, compute the p-saturation of $\mathcal{O}$ in $\mathbb{Z}_F$
   (a ***p*-maximal order**): **Medium**.

Chistov ('89): problem is equivalent to finding the largest squarefree factor of a given integer.

Algorithms computing a *p*-maximal order

There are two main algorithms to compute a *p*-maximal order:

- "Round 2": based on linear algebra. **Medium** (polynomial time).
- "Round 4": based on polynomial operations. Faster in practice, not proved to terminate in polynomial time.

Round 2

Define

$$J_p(\mathcal{O}) = \{x \in \mathcal{O} \mid x \bmod p \text{ is nilpotent in } \mathcal{O}/p\mathcal{O}\}, \text{ and}$$

$$\mathcal{O}' = \{y \in F \mid yJ_p(\mathcal{O}) \subset J_p(\mathcal{O})\}.$$

Theorem: $\mathcal{O}'$ is an order containing $\mathcal{O}$, and $\mathcal{O} = \mathcal{O}'$ if and only if $\mathcal{O}$ is p-maximal.

Let $a \in A = \mathcal{O}/p\mathcal{O}$, and assume for simplicity that $p > d$. Then a is nilpotent if and only if $\mathrm{Tr}(ax) = 0$ for all $x \in A$.

Algebraic number theory: Ideals

Ideals of rings of integers

- **Fractional ideal**: $\frac{1}{b}\mathfrak{a}$ with $\mathfrak{a} \subset \mathbb{Z}_F$ nonzero with $b \in \mathbb{Z}_{>0}$.
- Every fractional ideal is invertible.
- $N(\mathfrak{a}) = |\mathbb{Z}_F/\mathfrak{a}|$ extends multiplicatively.
- Unique factorisation into products of prime ideals.
- Every prime ideal $\mathfrak{p}$ appears in the factorisation of $\mathfrak{a} = p\mathbb{Z}_F$ where $p\mathbb{Z} = \mathfrak{p} \cap \mathbb{Z}$.

Representation of ideals

- $\mathfrak{a} \subset \mathbb{Z}_F$ represented by the matrix of a basis.
- $\mathfrak{a} = a\mathbb{Z}_F + b\mathbb{Z}_F$ where $a, b \in \mathbb{Z}_F$.

Factorisation

Problem: Consider $\mathfrak{a} \subset \mathbb{Z}_F$, compute its factorisation.

Factoring ideals is at least as hard as factoring integers.

1. Factor $N(\mathfrak{a})$: **Slow**.
2. Given a prime divisor p, find the factorisation of $p\mathbb{Z}_F$: **Medium**.
3. Compute the exponent $v_{\mathfrak{p}}(\mathfrak{a})$ (valuation) of $\mathfrak{p}$ in $\mathfrak{a}$: **Medium**.

Decomposition of primes

Let p be a prime number, and write $p\mathbb{Z}_F = \prod \mathfrak{p}^{e_\mathfrak{p}}$.

$$\mathbb{Z}_F/p\mathbb{Z}_F \cong \prod \mathbb{Z}_F/\mathfrak{p}^{e_\mathfrak{p}}.$$

For simplicity, assume p does not divide $\mathrm{disc}(\mathbb{Z}[\alpha])$.
Write $P = \prod_i P_i \bmod p$ the factorisation of P in $\mathbb{F}_p[X]$.

$$\mathbb{Z}[\alpha]/(p) \cong \mathbb{Z}[X]/(p,P) \cong \mathbb{F}_p[X]/(P) \cong \prod_i \mathbb{F}_p[X]/(P_i).$$

We obtain the prime ideals

$$\mathfrak{p}_i = (p, P_i(\alpha)) \text{ with residue field } \mathbb{Z}_F/\mathfrak{p}_i \cong \mathbb{F}_p[X]/(P_i).$$

Valuation

Basic algorithm:

1. Reduce the basis of $\mathfrak{a}$ modulo $\mathfrak{p}$.
2. If $\mathfrak{a} \bmod \mathfrak{p} = 0$, divide by $\mathfrak{p}$ and repeat.

Can do better by precomputing $z \in \frac{1}{p}\mathbb{Z}_F$ such that

- $v_{\mathfrak{p}}(z) = -1$,
- $v_{\mathfrak{q}}(z) = 0$ for all $\mathfrak{q} \neq \mathfrak{p}$ above p.

Algebraic number theory: Galois group

Galois theory

A number field F is **Galois** if of the following equivalent properties hold:

- F is generated by the set of **all roots** of a polynomial in $\mathbb{Q}[X]$;
- F contains all roots its defining polynomial P;
- $|\mathrm{Aut}(F)| = d$.

In this case, we call $\mathrm{Gal}(F/\mathbb{Q}) = \mathrm{Aut}(F)$ its **Galois group**, and there is a one-to-one, inclusion-reversing correspondence between subfields of F and subgroups of $\mathrm{Gal}(F/\mathbb{Q})$.

In general, $F \subset \tilde{F}$ **Galois closure** (smallest Galois extension containing F), generated by the set of all roots $\alpha_1, \ldots, \alpha_d$ of P.

Frobenius elements

Assume $F/\mathbb{Q}$ is Galois, and let $\mathfrak{p} \subset \mathbb{Z}_F$ be an unramified ($e_\mathfrak{p} = 1$) prime ideal.

There exists a **Frobenius element** $\mathrm{Frob}_\mathfrak{p} \in \mathrm{Gal}(F/\mathbb{Q})$ such that

- $\mathrm{Frob}_\mathfrak{p}(\mathfrak{p}) = \mathfrak{p}$,
- $\mathrm{Frob}_\mathfrak{p}$ induces the Frobenius automorphism $x \mapsto x^p$ on the residue field of $\mathfrak{p}$, and
- $\mathrm{Frob}_\mathfrak{p}$ has order equal to the degree of the residue field.

The $\mathrm{Frob}_\mathfrak{p}$ for $\mathfrak{p} \mid p$ are conjugate and we call Frob_p this conjugacy class.

In general, the cycle type of $\mathrm{Frob}_p \in \mathrm{Gal}(\tilde{F}/\mathbb{Q})$ as a permutation of the roots of P is the factorisation type of p in F.

Two problems

We have two different problems:

1. Given F Galois, compute $\mathrm{Gal}(F/\mathbb{Q})$.
2. Given F arbitrary, compute $\mathrm{Gal}(\tilde{F}/\mathbb{Q}) \subset S_d$.

$[\tilde{F} : \mathbb{Q}]$ can be as large as $d!$.

Galois closure

Algorithm:

1. Start with $F_1 = F$.
2. At Step i, factor P over F_i. If there is a nonlinear factor, let F_{i+1} be the corresponding extension, and repeat.

Every step is polynomial time (**Medium**), but the sizes increase! The algorithm is polynomial in the **size of the output**.

Automorphisms: direct computation

In general, computing the **automorphism group** amounts to finding the roots of the defining polynomial P over F.

We reduce to the polynomial factorisation problem: **Medium**.

This works regardless of $F/\mathbb{Q}$ being Galois or not.

Combinatorial algorithm

Allombert's algorithm.

- Tries to guess a Frobenius element generating a normal subgroup, compute the corresponding subfield K, recursively compute the Galois group of K.
- Works for supersolvable Galois groups.
- **Fast** in practice but can be exponential in the worst case (**Very slow**).

Special cases

Other special cases:

- Abelian Galois group (Acciaro – Klüners): **Medium** but faster.
- Nilpotent Galois groups (Allombert, unpublished): **Medium** with a similar speedup.

Subfields

Computation of subfield F^H corresponding to a subgroup H:

- Generated by the trace $\mathrm{Tr}_{F/F^H}(\lambda)$ for some $\lambda \in F$.
- Computation by reconstruction.

Complexity: **Medium**.

Frobenius elements

Computation of the Frobenius element $\mathrm{Frob}_{\mathfrak{p}}$:

1. For every $\sigma \in \mathrm{Gal}(F/\mathbb{Q})$:
2. Test if σ has the correct order.
3. Test if $\sigma(\mathfrak{p}) = \mathfrak{p}$.
4. Test if $\sigma(\alpha) = \alpha^p \bmod \mathfrak{p}$.

Complexity: **Medium**.

Galois group of the Galois closure

Idea:

1. Update a subgroup $G \subset S_d$ such that $\mathrm{Gal}(\tilde{F}/\mathbb{Q}) \subset G$.
2. Start with $G = S_d$.
3. Repeat:
4. Compute the maximal proper subgroups $H \subset G$.
5. For each H, test if $\mathrm{Gal}(\tilde{F}/\mathbb{Q}) \subset H$.

Step (4) requires nontrivial computational group theory, no known polynomial time algorithm.

Resolvents

Assume $\mathrm{Gal}(\tilde{F}/\mathbb{Q}) \subset G \subset S_d$.
Let $F \in \mathbb{Z}[X_1, \ldots, X_d]$ and let $H \subset G$ be its stabiliser. Define the **resolvent polynomial**

$$R(F) = \prod_{\sigma \in G/H} (X - F(\alpha_{\sigma(1)}, \ldots, \alpha_{\sigma(d)})) \in \mathbb{Z}[X].$$

Theorem: Assume $R(F)$ is squarefree. The resolvent $R(F)$ has a root in $\mathbb{Z}$ if and only if $\mathrm{Gal}(\tilde{F}/\mathbb{Q})$ is conjugate to a subgroup of H.

Computation by reconstruction and factorisation. Bottleneck: find nice F so that $R(F)$ is not too large.

Subfields

Resolvents also provide defining polynomials for the subfield fixed by a subgroup $H \subset \mathrm{Gal}(\tilde{F}/\mathbb{Q})$.

Frobenius elements

Identifying Frobenius elements (Dokchitser–Dokchitser) by a variant of the resolvent method.

Let C be a conjugacy class in $\mathrm{Gal}(\tilde{F}/\mathbb{Q})$ and $h(X) \in \mathbb{Z}[X]$. Define

$$\Gamma_C = \prod_{\sigma \in C} \left(X - \sum_{i=1}^{d} h(\alpha_i)\alpha_{\sigma(i)} \right) \in \mathbb{Z}[X].$$

Theorem: there exists $h(X)$ such that

$$\mathrm{Frob}_p \in C \iff \Gamma_C(\mathrm{Tr}_{F/\mathbb{Q}}(h(\alpha)\alpha^p)) = 0 \bmod p$$

for almost all p.

Remark: often one can take $h(X) = X^2$.

Subfields without the Galois group

If we only want the subfields of F (van Hoeij – Klueners – Novocin):

- Number of subfields can be superpolynomial.
- Notion of **generating subfields**: all subfields are intersections of the generating ones.
- Uses subfields of the form

$$\{x \in F \mid \sigma_1(x) = \sigma_2(x)\},$$

  where $\sigma_1, \sigma_2 \colon F \to K$.
- Get (K, σ_1, σ_2) from factoring P over F: **Medium**.

Algebraic number theory: Class group and units

Class group and unit group

A fractional ideal is **principal** if it is generated by one element. The **class group**

$$\mathrm{Cl}(F) = \{\text{fractional ideals}\}/\{\text{principal ideals}\}.$$

is a finite abelian group.

The **unit group** $\mathbb{Z}_F^\times$ is a finitely generated abelian group of rank $r_1 + r_2 - 1$, where (r_1, r_2) is the signature of F.

Problems

- Compute the structure of $\mathrm{Cl}(F)$ and $\mathbb{Z}_F^\times$;
- Compute generators for $\mathrm{Cl}(F)$ and $\mathbb{Z}_F^\times$;
- Given an element of $\mathrm{Cl}(F)$ or $\mathbb{Z}_F^\times$, write it as a product of the generators;
- Given a principal ideal $\mathfrak{a}$, compute a generator of $\mathfrak{a}$.

S-units

Let S be a set of prime ideals of F.
The group of S**-units** is

$$\mathbb{Z}_{F,S}^{\times} = \{u \in F^{\times} \mid v_{\mathfrak{p}}(u) = 0 \text{ for all } \mathfrak{p} \notin S\}.$$

It is a finitely generated group of rank $r_1 + r_2 + |S| - 1$.

Reductions to S-unit group

If S **generates** $\mathrm{Cl}(F)$, then considering

$$v_S \colon \mathbb{Z}_{F,S}^{\times} \to \mathbb{Z}^S,$$

where $v_S(u) = (v_{\mathfrak{p}}(u))_{\mathfrak{p} \in S}$, we have

$$\mathrm{Cl}(F) = \operatorname{coker} v_S \text{ and } \mathbb{Z}_F^{\times} = \ker v_S.$$

Moreover, if $\mathfrak{a}$ factors completely over the primes in S and $\mathfrak{a}$ is principal, then every generator of $\mathfrak{a}$ is an S-unit.

Roots of unity

The torsion subgroup of $\mathbb{Z}_F^\times$ is easier:

- $\zeta_m \in F \Longrightarrow \mathbb{Q}(\zeta_m) \subset F \Longrightarrow \varphi(m) \mid d \Longrightarrow$ finitely many possible m.
- For fixed m, factor the cyclotomic polynomial Φ_m over F.

Total complexity: **Medium**.

The Riemann Hypothesis

The function

$$\zeta(s) = \prod_p (1 - p^{-s})^{-1} \text{ for } \Re(s) > 1$$

can be continued to a meromorphic function on $\mathbb{C}$.

Conjecture (Riemann Hypothesis):
All zeroes ρ of ζ satisfy $\Re(\rho) \leq \frac{1}{2}$.

The generalisation of this conjecture to other *L*-functions is called the **Generalised Riemann Hypothesis (GRH)**.

Algorithms

- Under GRH, all problems: **Slow** $\exp(\tilde{\mathrm{O}}(d^{2/3})) + \exp(\tilde{\mathrm{O}}((\log \Delta_F)^{1/2}))$.
- Unconditionally: **Very slow** $\tilde{\mathrm{O}}(\Delta_F^{1/2})$.
- Unconditionally, only $\mathbb{Z}_F^\times$: **Very slow** $\tilde{\mathrm{O}}(\Delta_F^{1/4})$.

In practice, complicated behaviour. Can reach:

- $d \sim 100$, $\Delta_F \sim 10^{200}$.
- But for $d = 2$, $\Delta_F \sim 10^{40}$.

Algorithms for special cases

There are special case where we can do better.

- **Cyclotomic fields** and their subfields: partial information from Iwasawa theory and other special techniques (Schoof, Fukuda).
- Fields with a **special** $\mathrm{Aut}(F)$: for instance, can reach $d \sim 1700$ and $\Delta_F \sim 10^{5000}$ (Bauch–Bernstein–de Valence–Lange–van Vredendaal, Biasse–Fieker–Hofmann–P.).

Computational mathematics focused on number theory

Aurel Page

5/06/2022 - 14/06/2022
SuSAAN - Nesin math village - Izmir, Turkey

Inria Bordeaux / IMB

First lecture: complexity

Complexity: running time as a function of size n of input.

- **Fast**: $\tilde{O}(n)$ (quasi-linear)
- **Medium**: $O(n^C)$ for some $C > 1$ (polynomial)
- **Slow**: $O(2^{n^\alpha})$ for some $0 < \alpha < 1$ (subexponential)
- **Very slow**: $O(2^{Cn})$ for some $C > 0$ (exponential)

First lecture: arithmetic operations

- Basic operations on integers and polynomials: **Fast**.
- Basic operations on matrices: **Medium**.
- Factorisation of polynomials: **Medium**.
- Factorisation of integers: **Slow**.

Second lecture: reconstruction

Reconstruction algorithms recover an object from specialisations or approximations. There are **fast** algorithms for reconstruction.

- Integers: modular or multimodular, real approximations.
- Polynomials: evaluation-interpolation.
- Rationals: continued fractions.
- Fractions: Padé approximants.

Second lecture: algebraic number theory

- **Ring of integers**: factorisation of discriminant (**Slow**), then Round 2 (**Medium**).
- **Ideal factorisation**: integer factorisation (**Slow**), then prime decomposition and valuations (**Medium**).
- **Galois group**, two settings: Galois input field (**Medium**), or non-Galois input field (no known polynomial-time algorithm).
- **Class group and units**: **Slow** under GRH, **Very slow** unconditionally.

Plan

Today:

1. Linear algebra
2. Lattices

Linear algebra

Linear algebra problems

Let R be a ring. An R-module is an abelian group with a linear action of R, i.e. $r \cdot (v_1 + v_2) = r \cdot v_1 + r \cdot v_2$.

Linear algebra problems:

- Image, kernel, preimages, determinant of a matrix $M \in M_{m,n}(R)$.
- Sum, intersection, equality of submodules of R^m.

Representation theory problems:

- Isomorphism of finitely generated R-modules.
- Basis, canonical form, invariants of R-modules.

Example

Let $R = \mathbb{Z}[\zeta_p]$.

- **Equality** of submodules of R^m reduces to equality of submodules of $\mathbb{Z}^{m(p-1)}$.
- **Isomorphism** of R-modules: for $\mathfrak{a}, \mathfrak{b} \subset \mathbb{Z}[\zeta_p]$, we have $\mathfrak{a} \cong \mathfrak{b}$ as R-modules if and only if $\mathfrak{a}\mathfrak{b}^{-1}$ is principal.

"Representation theory": R-module $= \mathbb{Z}$-module with action of the cyclic group C_p + conditions.

Linear algebra: Algorithms

Normal forms of matrices: one-sided

$M \in M_{m,n}(R)$.

$N = MU$ where $U \in \mathrm{GL}_n(R)$.

- Basic property: **echelon form**. In each column, **pivot** (last nonzero coefficient), whose row index is strictly increasing.
- Over many rings, normal form for **submodules** of R^m.
- Over a field: **reduced echelon form**, coefficients to the right of a pivot are 0.
- Over a PID: **Hermite Normal Form (HNF)**, coefficients to the right of a pivot are reduced modulo the pivot.
- Over $\mathbb{Z}/N\mathbb{Z}$: **Howell form**:

$$R = \mathbb{Z}/4\mathbb{Z},\ M = \begin{pmatrix} 0 & 3 \\ 0 & 2 \end{pmatrix},\ N = \begin{pmatrix} 2 & 1 \\ 0 & 2 \end{pmatrix}.$$

Normal forms of matrices: two-sided

$M \in M_{m,n}(R)$.

$N = UMV$ where $U \in \mathrm{GL}_m(R)$, $V \in \mathrm{GL}_n(R)$.

- Over a field: diagonal with coefficients 1 (**rank**) then 0.
- Over a PID: **Smith Normal Form (SNF)** diagonal with each coefficient dividing the next one (**elementary divisors**).

Reveals isomorphism class of R^m/MR^n.

Example: SNF over $\mathbb{Z}$ gives the form $\mathbb{Z}/d_1\mathbb{Z} \times \cdots \times \mathbb{Z}/d_k\mathbb{Z} \times \mathbb{Z}^r$.

Normal forms of matrices: conjugacy

$M \in M_n(R)$.

$N = U^{-1}MU$ where $U \in \mathrm{GL}_n(R)$, i.e. $UN = MU$.

- Over algebraically closed fields: Jordan form (**eigenvalues**).
- Over fields: Frobenius form.
- Related to $R[X]$**-module** structure on R^n given by $X \cdot v = Mv$ (U is an $R[X]$-module isomorphism).

Echelon form algorithm

Let R be a ring that admits extended GCDs: for every $a, b \in R$, there exists $d \in R$ and $U \in \mathrm{GL}_2(R)$ such that

$$\begin{pmatrix} a & b \end{pmatrix} = \begin{pmatrix} 0 & d \end{pmatrix} U.$$

Algorithm: Input $M \in M_{m,n}(R)$.

1. For each row from the bottom to the top:
2. Use the first nonzero coefficient as pivot.
3. Set to 0 all coefficients to the left of the pivot.
4. Cleanup.

Complexity: $O(m^2 n)$ (**Medium**).

Faster echelon form

Use **faster multiplication** of matrices to get all linear algebra operation in time $O(n^\omega)$.
Idea: use block operations.

Simple case: inverse with $m = n = 2^k$ blocks of size 2^{k-1}.

$$M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

Generically uses 2 **inverses** and $O(1)$ multiplications.
The number $T(n) = T(2^k) = u_k$ of ring operations satisfies

$$T(n) = 2T(n/2) + O(n^\omega), \text{ i.e. } u_k = 2u_{k-1} + O(2^{\omega k}).$$

Solution $u_k = O(2^{\omega k})$, i.e. $T(n) = O(n^\omega)$.

Complexity depending on the ring

The complexity depends on the ring because of **coefficient growth**.

- Echelon form over a finite field: $\tilde{O}(n^\omega \log q)$.
- Howell form over $\mathbb{Z}/N\mathbb{Z}$: $\tilde{O}(n^\omega \log N)$.
- HNF over $\mathbb{Z}$: $\tilde{O}(n^{\omega+1} \log B)$ if coefficients are bounded by B.

Reconstruction to prevent coefficient blowup

In rings where coefficients can blowup, use reconstruction.

Example: determinant over $\mathbb{Z}$.

- Compute B such that $|\det M| \leq B$.
- Compute $\det M \bmod p_i$ for several p_i such that $\prod_i p_i > 2B$.
- Reconstruct $\det M$.

Linear algebra: Applications

Techniques to turn a problem into linear algebra

1. Logarithms
2. Newton's method
3. Index calculus

Logarithms

Multiplicative problems can be turned into linear ones using **logarithms**.

Example: norm equation in number fields. Given $a \in \mathbb{Z}_F$, solve

$$N_{K/F}(x) = a \text{ with } x \in \mathbb{Z}_K.$$

1. Factor the ideal $a\mathbb{Z}_F$.
2. Compute corresponding S-units in F and K.
3. Write $a \in \mathbb{Z}_F$ as a product of S-units.
4. Look for x as an S-unit: solve the linear system.

Example: saturation in number fields

Let p be a prime number and F a number field.

Problem: Given $H \subset F^\times$, compute the p-saturation of H.

Necessary conditions for $h = \prod_i h_i^{x_i}$ to be a p-th power:

- all $v_\mathfrak{p}(h) = 0 \bmod p$, and
- all discrete logarithms of h mod $\mathfrak{q}$ are $= 0 \bmod p$.

Algorithm (Pohst–Zassenhaus):

1. Use a finite number of conditions to form a **linear system mod** p satisfied by the x_i.
2. Solve the system to identify candidate h.
3. Try to compute a p-th root of h (polynomial factorisation over F).
4. If this fails, add more conditions. Repeat.

Newton or Hensel method

To solve $f(X) = 0$ from approximate solution X_0, look for $X_1 = X_0 + E$ with E small and write

$$f(X_1) = f(X_0) + Df_{X_0}(E) + O(E^2).$$

Solve $Df_{X_0}(E) = -f(X_0)$ to get

$$f(X_1) = O(E^2).$$

Example: $f(X) = X^{-1} - A$, $Df_X(E) = X^{-1}EX^{-1}$.

$$X_1 = 2X_0 - X_0AX_0.$$

Example: Dunford decomposition

Problem: Write $M = D + N$ with D diagonalisable, N nilpotent and N, D polynomials in M.

Algorithm:

1. Let P be the characteristic polynomial of M.
2. Let f be the squarefree part of P (we have $f(M)^n = 0$).
3. Do a Newton iteration to solve $f(X) = 0$, starting from $X_0 = M$.
4. The solution is $X = D$.

Factor bases and index calculus

Idea:

- We want to solve a multiplicative problem.
- We choose a **factor base** $\mathcal{B}$: a set of small primes over which we will try to factor elements.
- We generate small random elements and hope that they are **smooth**, i.e. that they factor over the factor base (**relations**).
- We use **linear algebra** to solve the initial problem from the relations.

Factorisation

Let N be the n-bits integer to factor. We want to find a, b such that

$$a^2 = b^2 \bmod N$$

so that $(a+b)(a-b) = 0 \bmod N$, and $\gcd(a+b, N)$ can be a nontrivial factor.

Algorithm:

1. Let $\mathcal{B} = \{p \mid p < y\} = \{p_1, \ldots, p_k\}$.
2. Repeat:
3. Generate a random $c \in \mathbb{Z}/N\mathbb{Z}$, let $c' = c^2 \bmod N$.
4. If c' is smooth, record relation $c^2 = p_1^{x_1} \ldots p_k^{x_k} \bmod N$.
5. By computing a kernel mod 2, find a combination of the relations that is a square, and deduce a and b.

Factorisation: analysis

Analysis: we are constructing random elements mod N, and hope that they are y-smooth.

Theorem: Random integers $\leq x$ are y-smooth with probability $\approx u^{-u}$ where $u = \log x / \log y$.

- Choose $y = \exp(n^{1/2})$, so that $u = O(n^{1/2})$.
- We find a smooth element after $u^u = \exp(\tilde{O}(n^{1/2}))$ trials.
- The size of linear algebra is $O(y) = O(\exp(n^{1/2}))$.

Total: $\exp(\tilde{O}(n^{1/2}))$ (**Slow**).

Discrete logarithms

Given g a generator of $\mathbb{F}_p^\times$ and $h = g^N \in \mathbb{F}_p^\times$, we want to find N.

Algorithm:

1. Let $\mathcal{B} = \{p \mid p < y\} = \{p_1, \ldots, p_k\}$.
2. Repeat:
3. Generate a random $z \in \mathbb{Z}/(p-1)\mathbb{Z}$, let $r = g^z \bmod p$.
4. If r is smooth, record relation $g^z = p_1^{x_1} \ldots p_k^{x_k} \bmod p$.
5. Invert the system mod $p-1$ to deduce the discrete logarithm of all p_i.
6. Draw random z until $g^{-z}h \bmod p$ is smooth.
7. From $h = g^z p_1^{x_1} \ldots p_k^{x_k}$, deduce the discrete logarithm of h.

S-unit group

Problem: find the *S*-unit group of a number field F.

Algorithm:

1. Choose y such that $S = \mathcal{B} = \{\mathfrak{p} \mid N(\mathfrak{p}) < y\} = \{\mathfrak{p}_1, \ldots, \mathfrak{p}_k\}$ generates the class group of F.
2. Repeat:
3. Generate a random $u \in \mathbb{Z}_F$.
4. If the ideal $u\mathbb{Z}_F$ is smooth (u is an S-unit):
5. Record relation: $u\mathbb{Z}_F = \mathfrak{p}_1^{x_1} \cdot \mathfrak{p}_k^{x_k}$.
6. Compute $\mathrm{Cl}(F)$ and $\mathbb{Z}_F^\times$ as a cokernel and kernel.

Termination criterion: analytic class number formula.

Another application: guessing relations

Problem: given the first few terms of a sequence (u_n), guess a recurrence relation of the form:

$$P_0(n)u_n + \cdots + P_k(n)u_{n+k} = 0.$$

Method: Choose $d \geq 0, k \geq 1$. Find a linear combination of

$$u_n, nu_n, \ldots, n^d u_n, u_{n+1}, nu_{n+1}, \ldots, n^d u_{n+1}, \ldots, n^d u_{n+k}$$

that vanishes for many values of n (more than $(k+1)(d+1)$).

Variant: Guess an algebraic or differential equation for the generating series $\sum_{n\geq 0} u_n X^n$. Solve the differential equation symbolically.

Lattices

Lattices

Let $V = \mathbb{R}^n$.
A subgroup $L \subset V$ is a **lattice** if the following equivalent conditions hold:

- L is generated by an $\mathbb{R}$-basis of V,
- L is discrete and V/L is compact.

A morphism between lattices must be linear and preserve the L^2 norm.

Equivalently, we can use $\mathbb{Z}^n$ equipped with a Euclidean norm.

Lattice problems

Input: a basis of a lattice $L \subset V$.

Problems:

- **Shortest vector problem (SVP)**: Compute a shortest nonzero vector of L.
- **Closest vector problem (CVP)**: Given $v \in V$, compute an element of L that is closest to v.
- **Automorphisms**: Compute $\mathrm{Aut}(L)$.
- **Isomorphism**: Given another lattice L', test if it is isomorphic to L.

Variants: γ-approximate versions, L^p norms.

Lattices: Algorithms

Approximation algorithm for SVP: LLL

LLL algorithm (Lenstra–Lenstra–Lovácz):

- Solves γ-SVP with $\gamma = C^n$.
- Polynomial time (**Medium**).
- In practice, $C \approx 1.02$.
- Gives a short and almost orthogonal basis of L: a **reduced basis**.

Exact algorithms for SVP

Enumeration algorithm (Kannan, Fincke–Pohst):

1. Compute a reduced basis of L.
2. Enumerate a set of linear combinations of the basis guaranteed to contain a shortest vector.

Complexity: $2^{O(n \log n)}$ (**Very very slow**). Polynomial space.

Sieve algorithm:

1. Compute a reduced basis of L.
2. Compute many vectors in L.
3. Find differences between pairs of vectors, that give shorter vectors. Repeat.

Complexity: $2^{O(n)}$ (**Very slow**). Exponential space. Also for L^p norms.

In practice: currently, n up to 180.

Approximation algorithm for SVP: BKZ

BKZ algorithm:

- Parameter: **block size** $2 \leq \beta \leq n$.
- Polynomial number of calls to an exact SVP algorithm in dimension β: complexity $2^{O(\beta)} \cdot (\text{input size})^c$.
- Solves γ-SVP with $\gamma = C^{n/\beta}$
- In practice: $C^{1/\beta} \approx 1.01$ for $\beta = 30$.
- Produces a reduced basis.

CVP algorithms

Babai's algorithm:

- Approximation of CVP depending on basis quality.
- Polynomial time (**Medium**).

Kannan's embedding:
B basis matrix of L, target vector v. Search short vector in

$$B' = \begin{pmatrix} B & -v \\ 0 & \alpha \end{pmatrix}$$

for some parameter α: $\|B'\binom{x}{\lambda}\|^2 = \|Bx - \lambda v\|^2 + \lambda^2\alpha^2$ (we want $\lambda = 1$).

Best known algorithms: $2^{O(n)}$ (**Very slow**). Also for L^p norms.

Automorphism and isomorphism algorithms

Plesken–Souvignier algorithm:

- Enumerate possible images of a short basis.
- Exploit some group theory.
- Many extra tricks to make it faster.
- Can add conditions on the isomorphisms (preserving extra structure).
- Complexity $2^{O(n^2 \log n)}$ (**Very very very slow**).

Haviv–Regev algorithm:

- Searches for a special vector in the dual lattice.
- Complexity $2^{O(n \log n)}$ (Theory, **Very very slow**)

Lattices over function fields

- $V = \mathbb{F}(X)^n$, $L \cong \mathbb{F}[X]^n$.
- size: maximum degree of the coefficients.
- Polynomial time algorithm (**Medium**) for SVP (A. Lenstra).

Lattices: Applications

Identifying a lattice problem

Two main families of applications:

1. Make something **smaller** under linear transformation (reduction theory).
2. Linear algebra problem with **size constraints**.

Example: short basis of a ring of integers

Let F be a number field with complex embeddings $\sigma_i \colon F \hookrightarrow \mathbb{C}$.
For $v \in F$ denote

$$T_2(v) = \sum_i |\sigma_i(v)|^2.$$

- Use a **reduced basis** of $\mathbb{Z}_F$ with respect to T_2.
- Reduces the **cost of arithmetic operations** in $\mathbb{Z}_F$.

Example: Knapsack problem

Knapsack problem: Given $x_1, \ldots, x_n \in \mathbb{Z}$ and $S \in \mathbb{Z}$, find a subset of $\{x_1, \ldots, x_n\}$ whose sum is S.

Rewrite as

$$\sum_i \varepsilon_i x_i = S \text{ with } \varepsilon_i \in \{0, 1\}.$$

Lagarias-Odlyzko lattice:

$$\begin{pmatrix} 1 & 0 & \ldots & 0 & -x_1 \\ 0 & 1 & \ldots & 0 & -x_2 \\ 0 & 0 & \ddots & 0 & \vdots \\ 0 & 0 & \ldots & 1 & -x_n \\ 0 & 0 & \ldots & 0 & S \end{pmatrix}$$

Linear relations between real numbers

Problem: Given approximations of $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$, find **small** $a_1, \ldots, a_n \in \mathbb{Z}$ such that

$$a_1\alpha_1 + \cdots + a_n\alpha_n \approx 0.$$

Example: in a paper, formula

$$\frac{R^2 L}{\pi\Omega\sqrt{D}} \in \mathbb{Q}^\times.$$

Not verified numerically. Look for linear relation between $\log R$, $\log L$, $\log \pi$, $\log \Omega$, $\frac{1}{2}\log D$, $\log 2$, $\log 3$, and $\log 5$.

$$\frac{R^2 LD}{\pi\Omega} \approx \frac{2^6}{3^2 \cdot 5^2}.$$

Algebraic relation satisfied by a real number

Special case: Given an approximation of $\alpha \in \mathbb{C}$, find **small** $a_0, \ldots, a_n \in \mathbb{Z}$ such that

$$a_0 + a_1\alpha + \cdots + a_n\alpha^n \approx 0 \text{ i.e. } P(\alpha) = 0$$

with $P \in \mathbb{Z}[X]$.

If we can guess that $\alpha \in F$ for a specific number field F with basis $w_1, \ldots, w_n$, it is better to look for $c_0, c_1, \ldots, c_n \in \mathbb{Z}$ such that

$$c_0\alpha \approx c_1 w_1 + \cdots + c_n w_n.$$

Small defining polynomial

Let $F = \mathbb{Q}(\alpha)$ with defining polynomial P.
Problem: find a small defining polynomial for F.

Algorithm:

1. Compute $\mathbb{Z}_F$.
2. Compute a reduced basis of $\mathbb{Z}_F$ with respect to T_2.
3. Find a small element β of $\mathbb{Z}_F$ such that $F = \mathbb{Q}(\beta)$.
4. Output the minimal polynomial Q of β.

Reconstruction in number fields

Problem: Given $v \bmod \mathfrak{a}$ where $v \in \mathbb{Z}_F$ is sufficiently small, **reconstruct** v.

If $\mathfrak{a} = A\mathbb{Z}_F$ with $A \in \mathbb{Z}$, simply reconstruct the coefficients **independently**:

$$v = \sum_i x_i w_i$$

with $x_i \in \mathbb{Z}$ known $\bmod A$.

In general, **solve a CVP**: given $v' \in \mathbb{Z}_F$ such that $v' = v \bmod \mathfrak{a}$, look for $a \in \mathfrak{a}$ such that $v' - a$ is small: then $v = v' - a$.

Factorisation of polynomials

Problem: Find a nontrivial factor of $P \in \mathbb{Z}[X]$.

Algorithm (Lenstra–Lenstra–Lovácz):

- Let p be a prime number such that P is squarefree mod p.
- Let $Q \in \mathbb{F}_p[X]$ be a nontrivial factor of $P \bmod p$.
- Lift Q to a factor of $P \bmod p^k$ for some large enough k.
- There exists a unique factor $R \in \mathbb{Z}[X]$ of P such that Q divides $R \bmod p^k$.
- R is a **short vector** in the lattice

$$L = \{T \in \mathbb{Z}[X]_{<\deg P} \mid T \bmod p^k \text{ is a multiple of } Q\}.$$

- By taking k large enough, the LLL approximation of SVP is sufficient.

Generators of arithmetic groups

Let F be a number field.
Problem: Compute generators (and other kinds of information) of the group $\mathrm{GL}_n(\mathbb{Z}_F)$.

Algorithm (Voronoï):

1. Use a special kind of lattices called ”perfect”.
2. Repeat until no new lattice is found:
3. For each perfect lattice, compute its ”neighbours”.
4. Test **isomorphism** with previous perfect lattices.
5. From the geometry of the set of perfect forms (”Voronoï tesselation”), deduce information about $\mathrm{GL}_n(\mathbb{Z}_F)$.

Computational mathematics focused on number theory

Aurel Page

5/06/2022 - 14/06/2022
SuSAAN - Nesin math village - Izmir, Turkey

Inria Bordeaux / IMB

First lecture: complexity

Complexity: running time as a function of size n of input.

- **Fast**: $\tilde{O}(n)$ (quasi-linear)
- **Medium**: $O(n^C)$ for some $C > 1$ (polynomial)
- **Slow**: $O(2^{n^\alpha})$ for some $0 < \alpha < 1$ (subexponential)
- **Very slow**: $O(2^{Cn})$ for some $C > 0$ (exponential)

First lecture: arithmetic operations

- Basic operations on integers and polynomials: **Fast**.
- Basic operations on matrices: **Medium**.
- Factorisation of polynomials: **Medium**.
- Factorisation of integers: **Slow**.

Second lecture: reconstruction

Reconstruction algorithms recover an object from specialisations or approximations. There are **fast** algorithms for reconstruction.

- Integers: modular or multimodular, real approximations.
- Polynomials: evaluation-interpolation.
- Rationals: continued fractions.
- Fractions: Padé approximants.

Second lecture: algebraic number theory

- **Ring of integers**: factorisation of discriminant (**Slow**), then Round 2 (**Medium**).
- **Ideal factorisation**: integer factorisation (**Slow**), then prime decomposition and valuations (**Medium**).
- **Galois group**, two settings: Galois input field (**Medium**), or non-Galois input field (no known polynomial-time algorithm).
- **Class group and units**: **Slow** under GRH, **Very slow** unconditionally.

Third lecture: linear algebra

Assume $n \times n$ matrix multiplication in time $O(n^\omega)$

- **Linear algebra operations**: $O(n^\omega)$ ring operations (**Medium**).
- Over infinite rings, where coefficients can grow: **Medium** reconstruction-based algorithms.
- Methods to reduce a problem to linear algebra: **logarithms**, **Newton**'s method, index calculus (**factor base**).

Third lecture: lattices

Lattice: $L \subset V$ generated by an $\mathbb{R}$-basis.

- Lattice problems: **SVP**, **CVP**, **isomorphism**.
- Time-approximation trade-off for SVP: approximation factor $C^{n/\beta}$ in time $2^{O(\beta)}$ (**reduced basis**).
- **Very very slow** algorithms for lattice isomorphism.
- Identifying a lattice problem: **reduce** an object under linear transformations; linear problem with **size constraints**.

Plan

Today:

1. General algorithmic techniques
2. *L*-functions

General algorithmic techniques

Sorting

Sorting an array of n objects: $O(n \log n) = \tilde{O}(n)$ (**Fast**).

Applications:

- Removing **duplicate** objects;
- Finding **collisions**.

Sorting: duplicate objects

Array of n objects.

- Using pairwise **isomorphism tests**: $O(n^2)$ (**Medium**).
- If a **canonical form** (or complete invariants) is available: $O(n)$ computations of canonical form, then sort in $\tilde{O}(n)$ (**Fast**).
- If no canonical form is available:
  - Pseudo-canonical form (small number of possible forms for each object): partially remove duplicates.
  - Invariants: sort to group objects having the same invariants $\tilde{O}(n)$, then pairwise test $O(m^2)$ in group of size m.

Example: enumeration of number fields

Fix $d \geq 2$.

Problem: find all number fields F of degree d with $|\Delta_F| \leq X$.

Theorem (Hunter): Every number field F of degree d with $|\Delta_F| \leq X$ admits a defining polynomial $\sum_{i=0}^{d} a_i X^i$ with $|a_i| \leq f(i, X)$.

Algorithm:

1. Enumerate all polynomials P satisfying the bounds.
2. Keep only the ones with $|\Delta_F| \leq X$.
3. Remove duplicates:
   - Pseudo-canonical form with reduced defining polynomial.
   - Invariants: discriminant, signature, decomposition of small primes.

Total complexity $O(X^{(d+2)/4})$ (smallest discriminant F known for $d \leq 9$).

Example: Baby-Step-Giant-Step

Problem: discrete logarithm in a cyclic group G: given generator g and $h = g^k$, find k.

Let $N = |G|$.

1. Let $M = \lceil \sqrt{N} \rceil$ (note $k = a + bM$ with $0 \leq a, b < M$).
2. Compute $\{1, g, g^2, \ldots, g^{M-1}\}$ (**baby steps**).
3. Compute $\{h, hg^{-M}, hg^{-2M}, \ldots, hg^{-(M-1)M}\}$ (**giant steps**).
4. Find a collision $g^a = hg^{-bM}$, and deduce $h = g^{a+bM}$.

Complexity: $\tilde{\mathrm{O}}(M) = \tilde{\mathrm{O}}(\sqrt{N})$ (**Slow**).

Data structures

A data structure is an object containing n elements, and supporting certain operations (insertion, deletion, query, etc).

- **Array**: creation with fixed length, modify/read arbitrary element in time $O(1)$ (**Fast**).
- Chained **list**: append/read next element in time $O(1)$ (**Fast**).
- Balanced tree "**set**": insert/find/delete element in time $O(\log n)$ (**Fast**).
- Heap "**priority queue**": insert/delete element in time $O(\log n)$, find maximum in time $O(1)$ (**Fast**).
- **Interval tree**: insert/delete numbers / count elements in an interval $[a, b]$ in time $O(\log n)$ (**Fast**).
- etc.

Graphs

A **graph** is a pair (V, E) where:

- V is a set (**vertices**), and
- E is a subset of $V \times V$ (**edges** connecting pairs of vertices).
- Variants: directed or undirected, multiple edges, etc.
- Vertices and edges can have a label or weight.

A morphism of graphs $(V, E) \to (V', E')$ is a map of sets $V \to V'$ that preserves edges.

A graph encodes information in an abstract and visual way.

Example: sparse linear algebra

Operations on **sparse matrices** (i.e. with many zeroes) often have a nice interpretation in terms of graphs.

Example: large primes in index calculus

- Use a larger factor base, but limit number of **large primes** to 1 or 2.
- One large prime: eliminate by pairs (find **collisions**).
- Two large primes mod 2: find **cycles** in the graph with vertices = large primes and edges = relations.

Example: sparse linear algebra

Sparse matrix $M \in M_{m,n}(R)$: list pairs (i, j) of indices with nonzero corresponding coefficient $M_{i,j} \in R$.

Graph version: **bipartite graph** with left vertices $\{1, \ldots, n\}$ (column indices), right vertices $\{1, \ldots, m\}$ (row indices) and weighted edge (i, j) if $M_{i,j} \neq 0$.

Interpretation of elimination: a set of pivots corresponds to a **matching** (set of edges with no common vertex) + condition ("restricted matching").

Finding optimal matchings is difficult, but the graph interpretation leads to good heuristics.

Example: sparse linear algebra

A **complex** is a sequence

$$C_0 \xrightarrow{d_0} C_1 \xrightarrow{d_1} C_2 \xrightarrow{d_1} \cdots \xrightarrow{d_{k-1}} C_k$$

of vector spaces (or R-modules) such that $d_{i+1} d_i = 0$ for all i.
Goal: compute **homology groups** $H_i = \ker d_{i+1} / \operatorname{im} d_i$.

Example: a single map $f\colon V_1 \to V_2$ gives the complex

$$0 \longrightarrow V_1 \xrightarrow{f} V_2 \longrightarrow 0,$$

whose homology groups are the kernel and cokernel of f.

Example: sparse linear algebra

Complex:

$$C_0 \xrightarrow{d_0} C_1 \xrightarrow{d_1} C_2 \xrightarrow{d_1} \cdots \xrightarrow{d_{k-1}} C_k$$

such that $d_{i+1} d_i = 0$ for all i.

Graph interpretation: $(k+1)$-partite graph.

Algebraic Morse theory: from a ”zig-zag path” with invertible weights, compute an equivalent, simplified

Example: equivalence relations

Goal: encode an **equivalence relation** on a set of n elements.

- Graph interpretation: connected components.
- **Representative** of each equivalence class.
- **Union**: add a relation between two elements: $O(\alpha(n))$ (**Fast**),
- **Find** the representative of the class of an element: $O(\alpha(n))$ (**Fast**),

where $\alpha(n)$ is the inverse Ackermann function (extremely slowly growing).

Graph isomorphism

Problem: given two graphs G and G', find an isomorphism between them.

In practice:

- Combinatorics + backtrack + group theory.
- **Fast** on most instances.
- Also computes a **canonical form**.

In theory:

- Babai: quasi-polynomial time algorithm (almost **Medium**).
- Mostly group theory.

Example: lattice isomorphism via graphs

Problem: given two lattices L and L', find an isomorphism between them.

- Let B such that $V = \{v \in L \mid \|v\| \leq B\}$ generates L.
- Graph with **vertex set** V.
- **Edge** (v_1, v_2) with weight $\langle v_1, v_2 \rangle$.
- Lattice isomorphism reduces to graph isomorphism on those graphs.
- Dutour–Haensch–Voight–van der Woerden: **canonical form** for lattices, from graphs.

Graph spectrum and random walks

Let G be an undirected graph with vertex set $\{1, \ldots, n\}$.

- The **adjacency matrix** is $M \in M_n(\mathbb{R})$ with $M_{i,j} = 1$ if $i = j$ or (i, j) is an edge, and $M_{i,j} = 0$ otherwise.
- The (i, j)-th coefficient of M^k count the number of paths of length k between i and j.
- The **spectrum** of the symmetric matrix M determines how fast random walks on G become equidistributed.

L-functions

Example: Dedekind zeta function

Let $\Gamma(s) = \int_0^\infty \exp(-t)t^{s-1}dt$, $\Gamma_{\mathbb{R}}(s) = \pi^{-s/2}\Gamma(s/2)$ and $\Gamma_{\mathbb{C}}(s) = \Gamma_{\mathbb{R}}(s)\gamma_{\mathbb{R}}(s+1)$.

Let F be a number field. The **Dedekind zeta function**

$$\zeta_F(s) = \prod_{\mathfrak{p}}(1 - N(\mathfrak{p})^{-s})^{-1}$$

extends to a meromorphic function such that

$$\Lambda(s) = |\Delta_F|^{s/2}\Gamma_{\mathbb{R}}(s)^{r_1}\Gamma_{\mathbb{C}}(s)^{r_2}\zeta_F(s) \text{ satisfies } \Lambda(s) = \Lambda(1-s).$$

Example: Dirichlet *L*-function

Let $\chi : (\mathbb{Z}/N\mathbb{Z})^\times \to \mathbb{C}^\times$ be a character.
The **Dirichlet *L*-function**

$$L(\chi, s) = \prod_{p \nmid N} (1 - \chi(p)p^{-s})^{-s}$$

extends to a holomorphic function such that, if χ is **primitive** (does not come from a smaller N),

$$\Lambda(s) = N^{s/2}\Gamma_{\mathbb{R}}(s + \varepsilon)L(\chi, s) \text{ satisfies } \Lambda(s) = \pm\Lambda(1 - s).$$

($\varepsilon \in \{0, 1\}$).

Example: elliptic curve

Let $E/\mathbb{Q} : Y^2 = X^3 + aX + b$ be an elliptic curve of conductor N. The ***L*-function of** E

$$L(E, s) = \prod_{p|N}(\cdots)\prod_{p\not|N}(1 - a_p p^{-s} + p^{1-2s})^{-1}$$

extends to a holomorphic function such that

$$\Lambda(s) = N^{s/2}\Gamma_{\mathbb{C}}(s)L(E, s) \text{ satisfies } \Lambda(s) = \pm\Lambda(2 - s).$$

Example: Hecke character

Let F be a real quadratic field with narrow class number 1 and fundamental unit u. The **Hecke L-function**

$$L(\psi, s) = \prod_{\mathfrak{p}} (1 - \exp(\frac{2\pi i \log g}{\log u}) N(\mathfrak{p})^{-s})^{-1}$$

where g is a totally positive generator of $\mathfrak{p}$, extends to a holomorphic function such that

$\Lambda(s) = |\Delta_F|^{s/2} \Gamma_{\mathbb{R}}(s + \frac{\pi}{2 \log u}) L(\psi, s)$ satisfies $\Lambda(s) = w\overline{\Lambda(1 - \bar{s})}$.

with $|w| = 1$.

General definition

An ***L*-function** of **degree** d and **conductor** N is a meromorphic function $L(s)$ on $\mathbb{C}$ with finitely many poles defined for $\Re(s)$ sufficiently large by an **Euler product**

$$L(s) = \prod_p F_p(p^{-s})$$

where F_p is a polynomial of degree $\leq d$ with $F_p(0) = 1$, and exactly d and all roots of modulus 1 for $p \not| N$, such that

$$\Lambda(s) = N^{s/2} \prod_{i=1}^{d} \Gamma_{\mathbb{R}}(s + \alpha_i) \text{ satisfies } \Lambda(s) = w\overline{\Lambda(1 - \bar{s})}.$$

with $|w| = 1$ (root number).

Modular and automorphic forms

Other sources of *L*-functions:

- **Modular forms**: functions on Poincaré 's upper half-plane satisfying a functional equation under $\mathrm{SL}_2(\mathbb{Z})$ and eigenfunctions of "Hecke operators".
- More generally, **automorphic forms**.

Curves and varieties

Other sources of *L*-functions: curves and more generally varieties X.

- Euler product from **point counts** of X modulo p.
- Gamma factors from **analytic properties (Hodge theory)** of X over $\mathbb{C}$.
- N from the **bad reduction** of X.

Galois representations

Other sources of L-functions: Galois representation

$$\rho\colon \mathrm{Gal}(K/F) \to \mathrm{GL}_d(E).$$

- Euler product from characteristic polynomial of **Frobenius elements**.
- Gamma factors from eigenvalues of **complex conjugations**.
- N from the **ramification properties** of K/F.

L-functions: Conjectures

Conjectures: GRH

Conjecture: Generalised Riemann Hypothesis (GRH):

$$L(\rho) = 0 \Longrightarrow \Re(\rho) \leq \frac{1}{2}.$$

Known: a very small zero-free region.

Conjectures: Lindelöf hypothesis

Conjecture: Lindelöf Hypothesis:

$$L(\frac{1}{2} + it) = O((N(1 + |t|))^{\varepsilon}) \textbf{ for all } \varepsilon > 0.$$

Known: $\varepsilon = 1/4$, sometimes slightly less (subconvexity).

Conjectures: special values

Conjecture: special values (BSD, Beilinson, Bloch-Kato, etc): Order of vanishing and value at certain special points have a different interpretation.

Example: Birch and Swinnerton–Dyer $L(E, s)$. Order of vanishing at $s = 1$ should be the rank of $E(\mathbb{Q})$.

Known: Some special cases. For BSD, rank 0 or 1.
For ζ_F, analytic class numer formula.

Conjectures: automorphy

Conjecture: automorphy: all *L*-functions come from automorphic forms.

Known: Many special cases, the most famous being opbtained from Wiles's method.

Algorithms

- Computation of good Euler factors.
- Evaluation.
- Computation of bad Euler factors.

Applications of GRH

- Strong prime number theorem:
  - Generation of class groups
  - Effective detection of powers.
  - Effetive Chebotarev theorem.
- Fast evaluation at low precision.

Questions?

Thank you!