# CSE291-14: The Number Field Sieve

https://cseweb.ucsd.edu/classes/wi22/cse291-14

Emmanuel Thomé

January 11, 2022

# Part 2

## Combinations of congruences

Combining congruences

CFRAC and QS

QS-era improvements

The golden age of QS and MPQS

Analysis of QS

# Plan

Combining congruences

CFRAC and QS

QS-era improvements

The golden age of QS and MPQS

Analysis of QS

# Combining congruences

An early idea due to Fermat. We try to factor $N$. Set $r = \left\lceil \sqrt{N} \right\rceil$.

- For $i = 0, \ldots,$ compute $f(i) = (r+i)^2 - N$.
- If $f(i)$ is a square, then we have:

$$\begin{aligned} (r+i)^2 - N &= x^2, \\ (r+i-x)(r+i+x) &= N. \end{aligned}$$

Let $N = pq$. This method factors $N$ in time $O(\frac{|p-q|}{2})$.
This suceeds if $p$, $q$ are too close to $\sqrt{N}$. Otherwise hopeless.

Note: tricks to make this work without long integer arithmetic:

- $f(i+1) - f(i) = 2(r+i) + 1$.
- To test whether $f(i) = \square$, look at $\left(\frac{f(i)}{p}\right)$ for many small $p$'s.

This saves some work, but does not change the outcome much.

Remaining idea: search for squares.

# Combining congruences

Given a composite $N$, what does $x^2 \equiv y^2 \mod N$ give ?

$$\begin{aligned} x^2 &\equiv y^2, \\ (x-y)(x+y) &\equiv 0, \\ (\frac{x}{y}-1)(\frac{x}{y}+1) &\equiv 0 \quad \text{(we may assume } \gcd(y,N)=1\text{)}. \end{aligned}$$

- If $N$ has $k$ distinct prime factors, there are $2^k$ different square roots
- A "random" congruence $x^2 \equiv y^2$ reveals a factor with prob $1 - \frac{1}{2^{k-1}}$.
- Note that this cannot work for prime powers.

# Kraitchik

From the 1930's:

- Looking at congruences is enough.
- If $r^2$ mod $N$ and $s^2$ mod $N$ are not squares, but their product is, then we succeed.

This is the principle of combination of congruences.

# Combination of congruences

$$46^2 \bmod 2041 = 75 = 3 \times 5^2,$$
$$47^2 \bmod 2041 = 168 = 2^3 \times 3 \times 7,$$
$$48^2 \bmod 2041 = 263 = \text{I am lazy, too hard...}$$
$$49^2 \bmod 2041 = 360 = 2^3 \times 3^2 \times 5,$$
$$50^2 \bmod 2041 = 459 = 3^3 \times 17,$$
$$51^2 \bmod 2041 = 560 = 2^4 \times 5 \times 7, \qquad \ldots$$

This leads to

$$(\underbrace{46 \times 47 \times 49 \times 51}_{x})^2 \equiv 2^{10}3^4 5^4 7^2 \equiv (\underbrace{2^5 3^2 5^2 7}_{y})^2 \mod N.$$

And $\gcd(x - y, N) = 13$.

# Dixon random squares algorithm

This was formalized by Dixon in the 1970s. Proven probabilistic.

**Smoothness** is the important thing!

We are interested in the factorization of $r^2 \bmod N$ only if it is **smooth**.

- We fix a smoothness bound $B$.
- The set of primes $\mathcal{P}_B$ is called the factor base.

**Algorithm:**

- Pick $r$ at random. Test divisibility by all primes below $B$. If $r^2 \bmod N$ is $B$-smooth, keep the relation:

$$r_i^2 \bmod N \equiv p_1^{e_{i,1}} \times \cdots \times p_k^{e_{i,k}}.$$

- Try to match these. This is a linear algebra problem over $\mathbb{F}_2$.

# Combination by linear algebra

We have a set $\mathcal{R}$ of relations $r_i^2 \equiv p_1^{e_{i,1}} \times \cdots \times p_k^{e_{i,k}}$.

- Consider the matrix $M \in \mathcal{M}_{\#\mathcal{R} \times \#\mathcal{P}}(\mathbb{Z})$, $M = (e_{i,j})$.

$$\begin{aligned} 46^2 \bmod 2041 &= 75 = 3 \times 5^2, \\ 47^2 \bmod 2041 &= 168 = 2^3 \times 3 \times 7, \\ 49^2 \bmod 2041 &= 360 = 2^3 \times 3^2 \times 5, \\ 51^2 \bmod 2041 &= 560 = 2^4 \times 5 \times 7, \end{aligned} \quad \Longrightarrow \quad \begin{pmatrix} 0 & 1 & 2 & 0 \\ 3 & 1 & 0 & 1 \\ 3 & 2 & 1 & 0 \\ 4 & 0 & 1 & 1 \end{pmatrix}$$

- A vector $V = (v_i)_{1 \leq i \leq \#\mathcal{R}}$ yields $VM = (\sum_i v_i e_{i,j})_j$, and:

$$(\prod r_i^{v_i})^2 \equiv \prod_j p_j^{\sum_i v_i e_{i,j}}.$$

- We want $V$ such that coordinates of $VM$ are even: it suffices to search for nullspace elements over the field $\mathbb{F}_2$.

# Pitfalls

**The random squares method has one main pitfall**

$r^2 \mod N$ is big: as large as $N$.

Further improvements focused on making the numbers to test for smoothness somewhat smaller.

# Plan

Combining congruences

CFRAC and QS

QS-era improvements

The golden age of QS and MPQS

Analysis of QS

# CFRAC

The CFRAC algorithm (Lehmer&Powers, 1931, and Morrison&Brillhart, 1975) was the first practical algorithm to factor large numbers, starting with $F_7 = 2^{128} + 1$.

Idea: use the continued fraction approximation of $\sqrt{N}$.

How continued fractions work is barely relevant, but a by-product is a host of identities of the form:

$$U_n^2 - NV_n^2 = Q_n,$$
$$U_n^2 \equiv Q_n \mod N,$$

with the very sweet property that $|Q_n| < 2\sqrt{N}$.

Test the $Q_n$'s for smoothness. They are only as large as $2\sqrt{N}$.

# The quadratic sieve (Pomerance)

In fact, continued fractions are not achieving anything magical.

A simpler supply of small square residues:

- Let $f(i) = \left(\left\lceil \sqrt{N} \right\rceil + i\right)^2 - N$.
- As long as $i$ remains small, $f(i)$ is of the order of $\sqrt{N}$.

Fermat was using the same $f$, hoping for $f(i)$ to be a square.

Here, we are combining this with the Dixon-like approach.

# QS: enter sieving

QS turn the Dixon approach on its head when it comes to factoring the residues.

Decide beforehand on a sieving space: interval $i \in [-A, A]$. We will try to factor all residues $f(i)$.

- Dixon: for each $i$, for each $p$, see if $p$ divides $f(i)$.
- QS: for each $p$, for each $i$, see if $p$ divides $f(i)$.
- QS: for each $p$, determine indices $i$ such that $p$ divides $f(i)$.

# Computing all valuations at once

Fix $p$. Let $r_0$, $r_1$ be the two roots of the quadratic equation $f(i) \equiv 0 \mod p$.

$$\{i \in [-A, A],\ f(i) \equiv 0 \mod p\} = \{r_0, r_0 \pm p, \ldots\} \cup \{r_1, r_1 \pm p, \ldots\}.$$

**Algorithm:** We maintain an array $T[i]$ indexed by $i \in [-A, A]$.

- For each $p \leq B$, do:
  - Compute $r_0, r_1$
  - $r := r_0$. While $r \leq A$ do:
    - $T[r] \leftarrow T[r] + \log p$,
    - $r \leftarrow r + p$.
  - idem for $r_1$ as well as $\{r_i - kp\}$.
- Do this also for prime powers
- For all $i$ such that $T[i] = \log |f(i)|$, we know that $f(i)$ is smooth.

# $T[i] = \log|f(i)| \Leftrightarrow f(i)$ smooth

Ignore powers for the moment. For each $p$:

- we have characterized the set $\mathcal{S}_{p,i} = \{i,\ \nu_p(f(i)) > 0\}$.
- we have added $\log_2 p$ to $T[i]$ for each $i$ in this set.

Once we have sieved for all $p \in \mathcal{P}_B$:

$$\begin{aligned} T[i] &= \sum_{p \in \mathcal{P}_B} \begin{cases} \log_2 p \text{ if } p \mid f(i), \\ 0 \text{ otherwise,} \end{cases} \\ &= \sum_{p \in \mathcal{P}_B,\ p \mid f(i)} \log p, \\ &= \log\left(B\text{-smooth part of } f(i) \text{ without powers}\right). \end{aligned}$$

If $T[i] = \log|f(i)|$ after sieving, then $f(i)$ is $B$-smooth and square-free.

# Sieving with powers

Assume that $f(i) \equiv 0$ has 2 distinct roots mod $p$ (so $p \nmid \mathrm{disc}(f)$).

- How many roots mod $p^2$ ?
- How many roots mod $p^k$ ?
- Which log contribution should we add ?

# Sieving with powers

Assume that $f(i) \equiv 0$ has 2 distinct roots mod $p$ (so $p \nmid \mathrm{disc}(f)$).

- How many roots mod $p^2$ ?
- How many roots mod $p^k$ ?
- Which log contribution should we add ?

We want to have an accumulated contribution of $k \log_2 p$ when $p^k \mid f(i)$, but since $f(i)$ is hit when sieving for $p, p^2, \ldots, p^k$, we need only add $\log_2 p$ each time.

# $T[i] = \log|f(i)| \Leftrightarrow f(i)$ smooth

For each $p^k$ (assuming we consider $k$ up to $\infty$. In fact we don't):

- we have characterized the set $\mathcal{S}_{p^k,i} = \{i,\ \nu_p(f(i)) \geq k\}$.
- we have added $\log_2 p$ to $T[i]$ for each $i$ in this set.

Thus eventually:

$$\begin{aligned} T[i] &= \sum_{p \in \mathcal{P}_B} \left( \sum_{k \text{ s.t. } i \in \mathcal{S}_{p^k,i}} \log p \right), \\ &= \sum_{p \in \mathcal{P}_B} \left( \sum_{k,\ \nu_p(f(i)) \geq k} \log p \right), \\ &= \sum_{p \in \mathcal{P}_B} \nu_p(f(i)) \log p, \\ &= \log\left(B\text{-smooth part of } f(i)\right). \end{aligned}$$

# QS: summary

We want to factor $N$. Let $f(x) = (\lceil\sqrt{N}\rceil + x)^2 - N$.

- Choose bounds $A$ and $B$. Analysis will tell how.
- Create array of low-precision values $T$, indices $i \in [-A, A]$.
- For all primes below $B$:
  - Find roots of $f$ mod $p$.
  - Add $\log_2 p$ to the appropriate locations.
  - Do this also for powers.
- For all $i$, compare $\log_2 |f(i)|$ and $T[i]$.
  Deduce the $i$'s such that $f(i)$ is $B$-smooth.
- Factor the smooth $f(i)$, possibly with resieving (lecture 1).
- Form a big linear system. Find a nullspace element.
- Determine a congruence of squares. Attempt to factor $N$.

# QS: the road ahead

In the previous summary of QS:

- The decor is basically here so that we can explain many of the following improvements.
- As it turns out, almost every line of the previous description has been a topic of study.
  These lines of study have all led to improvements that, most often are still part of relevant know-how for NFS.
  - Some improvements are very local.
  - Some have a broader consequence.

# Plan

Combining congruences

CFRAC and QS

QS-era improvements

The golden age of QS and MPQS

Analysis of QS

# QS: all steps deserve a look

We want to factor $N$. Let $f(x) = (\lceil \sqrt{N} \rceil + x)^2 - N$.

- Choose bounds $A$ and $B$. Analysis will tell how.
- Create array of low-precision values $T$, indices $i \in [-A, A]$.
- For all primes below $B$:
  - Find roots of $f$ mod $p$.
  - Add $\log_2 p$ to the appropriate locations.
  - Do this also for powers.
- For all $i$, compare $\log_2 |f(i)|$ and $T[i]$.
  Deduce the $i$'s such that $f(i)$ is $B$-smooth.
- Factor the smooth $f(i)$, possibly with resieving (lecture 1).
- Form a big linear system. Find a nullspace element.
- Determine a congruence of squares. Attempt to factor $N$.

# QS: all steps deserve a look

We want to factor $N$. Let $f(x) = (\lceil \sqrt{N} \rceil + x)^2 - N$.

- Choose bounds $A$ and $B$. Analysis will tell how.
- Create array of low-precision values $T$, indices $i \in [-A, A]$.
- For all primes below $B$:
  - Find roots of $f$ mod $p$.
  - Add $\log_2 p$ to the appropriate locations.
  - Do this also for powers.
- For all $i$, compare $\log_2 |f(i)|$ and $T[i]$.
  Deduce the $i$'s such that $f(i)$ is $B$-smooth.
- Factor the smooth $f(i)$, possibly with resieving (lecture 1).
- Form a big linear system. Find a nullspace element.
- Determine a congruence of squares. Attempt to factor $N$.

# Data type for array cells $T[i]$

$\log_2 |f(i)|$ and $\log_2 p$ are real numbers. What do we store in $T[i]$?

- Straightforward choice: store the integer $\lfloor \log_2 |f(i)| \rceil$, which certainly fits in 8 bits.
  This introduces some fuzziness, because of rounding. We can probably live with it.
- Better choice: adjust the log base so that the full 8-bit range is used. This is better for accuracy.
  - Determine the max value of $|f(i)|$ beforehand.
  - Provide for some tolerance for rounding.

# QS: all steps deserve a look

We want to factor $N$. Let $f(x) = (\lceil \sqrt{N} \rceil + x)^2 - N$.

- Choose bounds $A$ and $B$. Analysis will tell how.
- Create array of low-precision values $T$, indices $i \in [-A, A]$.
- For all primes below $B$:
  - Find roots of $f$ mod $p$.
  - Add $\log_\beta p$ to the appropriate locations.
  - Do this also for powers.
- For all $i$, compare $\log_\beta |f(i)|$ and $T[i]$.
  Deduce the $i$'s such that $f(i)$ is $B$-smooth.
- Factor the smooth $f(i)$, possibly with resieving (lecture 1).
- Form a big linear system. Find a nullspace element.
- Determine a congruence of squares. Attempt to factor $N$.

# log-norm computations and comparisons

When we compare $T[i]$ with $\log_\beta |f(i)|$, do we actually compute $f(i)$ and its log? Of course not.

- Keeping track of $f(i)$ alone would not be too hard.
- Once we take the log, the (rounded) value changes rarely.

In the QS context, this is rather easy to solve. This is an instance of the log-norm computations issue, that appears also in NFS.

Note that in practice, we prefer to do log-norm computations first:

- Fill $T[i]$ with rounded values of $\log_\beta |f(i)|$.
- Subtract the $\log_\beta p$ from array cell.

The benefit is that comparisons are done versus a constant bound.

# QS: all steps deserve a look

We want to factor $N$. Let $f(x) = (\lceil \sqrt{N} \rceil + x)^2 - N$.

- Choose bounds $A$ and $B$. Analysis will tell how.
- Create array of low-precision values $T$, indices $i \in [-A, A]$.
- For all primes below $B$:
  - Find roots of $f$ mod $p$.
  - Add $\log_\beta p$ to the appropriate locations.
  - Do this also for powers.
- For all $i$, compare $\log_\beta |f(i)|$ and $T[i]$.
  Deduce the $i$'s such that $f(i)$ is $B$-smooth.
- Factor the smooth $f(i)$, possibly with resieving (lecture 1).
- Form a big linear system. Find a nullspace element.
- Determine a congruence of squares. Attempt to factor $N$.

# Not all primes are the same

As presented, QS is one very large sieving effort.

The sieving work varies a lot with $p$.

- $p$ small: an immense number of narrowly-spaced table updates.
- $p$ large: rare, far apart table updates.

The proper way to deal with that has been a key topic for decades, and is still a very current topic with NFS today.

# QS: all steps deserve a look

We want to factor $N$. Let $f(x) = (\lceil \sqrt{N} \rceil + x)^2 - N$.

- Choose bounds $A$ and $B$. Analysis will tell how.
- Create array of low-precision values $T$, indices $i \in [-A, A]$.
- For all primes below $B$:
  - Find roots of $f$ mod $p$.
  - Add $\log_\beta p$ to the appropriate locations.
  - Do this also for powers.
- For all $i$, compare $\log_\beta |f(i)|$ and $T[i]$.
  Deduce the $i$'s such that $f(i)$ is $B$-smooth.
- Factor the smooth $f(i)$, possibly with resieving (lecture 1).
- Form a big linear system. Find a nullspace element.
- Determine a congruence of squares. Attempt to factor $N$.

# Linear algebra

How to deal with the linear systems that appear in factoring computations is a topic of its own.

Remember that as we only want to make all valuations even, linear algebra only needs to be done modulo 2.

Tentatively, we'll cover that mid-February.

# QS: all steps deserve a look

We want to factor $N$. Let $f(x) = (\lceil \sqrt{N} \rceil + x)^2 - N$.

- Choose bounds $A$ and $B$. Analysis will tell how.
- Create array of low-precision values $T$, indices $i \in [-A, A]$.
- For all primes below $B$:
  - Find roots of $f$ mod $p$.
  - Add $\log_\beta p$ to the appropriate locations.
  - Do this also for powers.
- For all $i$, compare $\log_\beta |f(i)|$ and $T[i]$.
  Deduce the $i$'s such that $f(i)$ is $B$-smooth.
- Factor the smooth $f(i)$, possibly with resieving (lecture 1).
- Form a big linear system. Find a nullspace element.
- Determine a congruence of squares. Attempt to factor $N$.

# Congruence of squares

To form a congruence of squares, the task is pretty much trivial in the QS context.

- Multiply the $f(i)$ that participate in the winning combination.
- Compute the corresponding factored form $\prod_{p\in\mathcal{F}} p^{\nu_p}$, which has all valuation $\nu_p$ even. (We need the $\nu_p$'s in $\mathbb{Z}$.)
- Then we have:

$$\left(\prod(\lceil\sqrt{N}\rceil + i)\right)^2 = \pm\left(\prod_{p\in\mathcal{F}} p^{\nu_p/2}\right)^2.$$

  (it is ok to compute all that modulo $N$, of course.)
- Dealing with the sign is not too hard.
  - A sign is attached to each relation given by a smooth $f(i)$.
  - We can insert a sign column in our matrix ($+1 \to 0$; $-1 \to 1$) so that a winning combination is always positive.

In the NFS context, this step becomes somewhat more complicated (but remains negligible overall in terms of computation time).

Some improvements of NFS have broader consequences.

# QS: all steps deserve a look

We want to factor $N$. Let $f(x) = (\lceil \sqrt{N} \rceil + x)^2 - N$.

- Choose bounds $A$ and $B$. Analysis will tell how.
- Create array of low-precision values $T$, indices $i \in [-A, A]$.
- For all primes below $B$:
  - Find roots of $f$ mod $p$.
  - Add $\log_\beta p$ to the appropriate locations.
  - Do this also for powers.
- For all $i$, compare $\log_\beta |f(i)|$ and $T[i]$.
  Deduce the $i$'s such that $f(i)$ is $B$-smooth.
- Factor the smooth $f(i)$, possibly with resieving (lecture 1).
- Form a big linear system. Find a nullspace element.
- Determine a congruence of squares. Attempt to factor $N$.

# Special-$q$

As described, QS uses a huge sieving table.

There are innumerable downsides to this.

- Diminishing returns;
- Need to have good aim;
- Splitting the computation into pieces is possible, but very irregular;
- Addressing a huge array in memory is not very efficient.

# Special-$q$

The special-$q$ idea was introduced in the early 1980s by Davis and Holridge, in the QS context.

- Pick a $q$ that is slightly above $B$.
- Work only with $i$'s that are such that $q|f(i)$.
  i.e., if $f(r) \equiv 0 \mod q$, work with $g(i) = f(qi + r)$.
- Do this for many different $q$'s.

In effect, this divides the sieving work into many subtasks:

- One subtask per $q$. There are many primes around $B$.
- Each subtask only needs to address $A/q$ indices.
- All subtasks cost roughly the same.
- There are a few downsides. Possible duplicates, and abundance of $q$'s in the relation matrix.

This is a significant overhaul of how the whole algorithm works, but it pays off significantly. We will see this with NFS.

# QS: all steps deserve a look

We want to factor $N$. Let $f(x) = (\lceil \sqrt{N} \rceil + x)^2 - N$.

- Choose bounds $A$ and $B$. Analysis will tell how.
- Create array of low-precision values $T$, indices $i \in [-A, A]$.
- For all primes below $B$:
  - Find roots of $f$ mod $p$.
  - Add $\log_\beta p$ to the appropriate locations.
  - Do this also for powers.
- For all $i$, compare $\log_\beta |f(i)|$ and $T[i]$.
  Deduce the $i$'s such that $f(i)$ is $B$-smooth.
- Factor the smooth $f(i)$, possibly with resieving (lecture 1).
- Form a big linear system. Find a nullspace element.
- Determine a congruence of squares. Attempt to factor $N$.

# Be tolerant

The comparison of $\log_\beta |f(i)|$ and $T[i]$ leads to an important choice: do we keep this $i$ or not?

Note that $\left| T[i] - \log_\beta |f(i)| \right|$ measures the size of the non-smooth cofactor part in $f(i)$.

- We know that this cofactor is free of prime factors below $B$ (except maybe powers).
- As a consequence, if the cofactor is less than $B^2$, it has to be a (large) prime number.

So this is "almost" a relation, and the extra prime is found at no extra cost.

This was the starting point of the large prime idea.

# Large primes

Historical point of view. When the cofactor is a large prime, we have a "partial" relation.

- Set a bound $L \leq B^2$ on the large primes.
- When $k$ partial relations are found, the birthday paradox tells us that we can form $\frac{k^2}{2L}$ combinations with large primes canceled in pairs.

Note that this point of view is vastly outdated. What remains is that the cofactor needs further inspection.

# QS: all steps deserve a look

We want to factor $N$. Let $f(x) = (\lceil \sqrt{N} \rceil + x)^2 - N$.

- Choose bounds $A$ and $B$. Analysis will tell how.
- Create array of low-precision values $T$, indices $i \in [-A, A]$.
- For all primes below $B$:
  - Find roots of $f$ mod $p$.
  - Add $\log_\beta p$ to the appropriate locations.
  - Do this also for powers.
- For all $i$, compare $\log_\beta |f(i)|$ and $T[i]$.
  Deduce the $i$'s such that $f(i)$ is $B$-smooth.
- Factor the smooth $f(i)$, possibly with resieving (lecture 1).
- Form a big linear system. Find a nullspace element.
- Determine a congruence of squares. Attempt to factor $N$.

# MPQS (Montgomery)

Probably the most important (and most specific) improvement of QS was its multiple polynomial variant.

Annoying feature of QS: $|f(x)|$ gets bigger as $x$ grows.

$$\max_{-A \le i \le A} |f(i)| = 2A\sqrt{N}.$$

We want to find other functions playing the role of $f(x)$.

# MPQS (Montgomery)

What happens if we look at $(ux + v)^2$ for some $u, v$ ?

$$(ux + v)^2 = u^2x^2 + 2uvx + v^2 - uw + uw \text{ for any } w,$$

If we have $v^2 - uw = N$:

$$\frac{1}{u}(ux + v)^2 \equiv ux^2 + 2vx + w \mod N.$$

Fix $u$ s.t. $\left(\frac{N}{u}\right) = 1$. Choose $v \leq \frac{u}{2}$ s.t. $v^2 \equiv N \mod u$.
Set $w$ accordingly. We have $w \approx -N/u$.

$$-\frac{N}{y} \lesssim ux^2 + 2vx + w \lesssim A^2u - \frac{N}{u}].$$

For a given $A$, smallest values for $u \approx \frac{2\sqrt{N}}{A}$. $\Rightarrow$ Bound $\frac{1}{\sqrt{2}}A\sqrt{N}$.

# MPQS

- For a given sieve interval size, we have found a better polynomial.
- More important, we have many such polynomials.
- If $u$ is a product of factor base primes, a large number of polynomials can be used (other option: $u = \square$).
- Shorter intervals per polynomial $\Rightarrow$ smaller residues.
- Initialization cost per polynomial: solving $v^2 \equiv N \mod u$. See SIQS for a way to amortize this (e.g. in Crandall-Pomerance).

MPQS/SIQS (with previous improvements) is the leading algorithm today for $p$ below 100-120 decimal digits.

Software: msieve and yafu probably have the best QS code around.

# Plan

Combining congruences

CFRAC and QS

QS-era improvements

The golden age of QS and MPQS

Analysis of QS

# MPQS is efficient

MPQS is a great leap forward compared to CFRAC, for two reasons.

- The numbers that are checked for smoothness are considerably smaller.
- Massive distribution is possible. (although special-$q$'s were a way to achieve that as well, it was not used that way).

Starting around 1986, and especially around the turn of the 1990s, many factoring records were broken with QS.

# Factoring by electronic mail

Very influential paper by Lenstra and Manasse, 1989.

> "*how big are the integers we can factor within one month of elapsed time, if we only want to use computing time that we can get for free?*"

This pretty much defined the way to establish the state of the art in academic cryptanalysis for the following decades.

Of course, there is a built-in gap with the cryptanalytic power of nation-state adversaries.

# Plan

Combining congruences

CFRAC and QS

QS-era improvements

The golden age of QS and MPQS

Analysis of QS

# Analysis of QS

The analysis of QS employs techniques from various domains.

The most important things that we try to estimate are:

- The size of the numbers $f(i)$ that we try to factor.
- The probability that these are $B$-smooth.
- The time it takes to collect enough relations.
- The time it takes to solve the linear system.

Most of the analysis techniques (for smoothness probabilities, in particular) were only nascent in the beginning of the 1980s.

# Smoothness: Estimating $\Psi(x, y)$

There's one main theorem known as:

- Canfield-Erdős-Pomerance (1983),
- Construction kit lemma,
- whatever credit people give... (Odlyzko / Balasubramanian)

It's also valid in various contexts.

## Canfield-Erdős-Pomerance (CEP) Theorem

Let $x, y \to +\infty$ and $\epsilon > 0$ s.t. $(\log x)^\epsilon < \log y < (\log x)^{1-\epsilon}$.
Let $\Psi(x, y) = \#\{n,\ 1 \le n \le x,\ n \text{ is } y\text{-smooth}\}$.

$$\frac{1}{x}\Psi(x, y) \sim \rho(u) = u^{-u(1+o(1))}$$

where $u = \frac{\log x}{\log y}$, and $\rho$ is the Dickman-de Bruijn function.

# The Dickman-De Bruijn function

$\rho(u)$ is the solution of a delay differential equation.

$$u\rho'(u) + \rho(u-1) = 0.$$

Note that computing $\rho$ is totally possible.

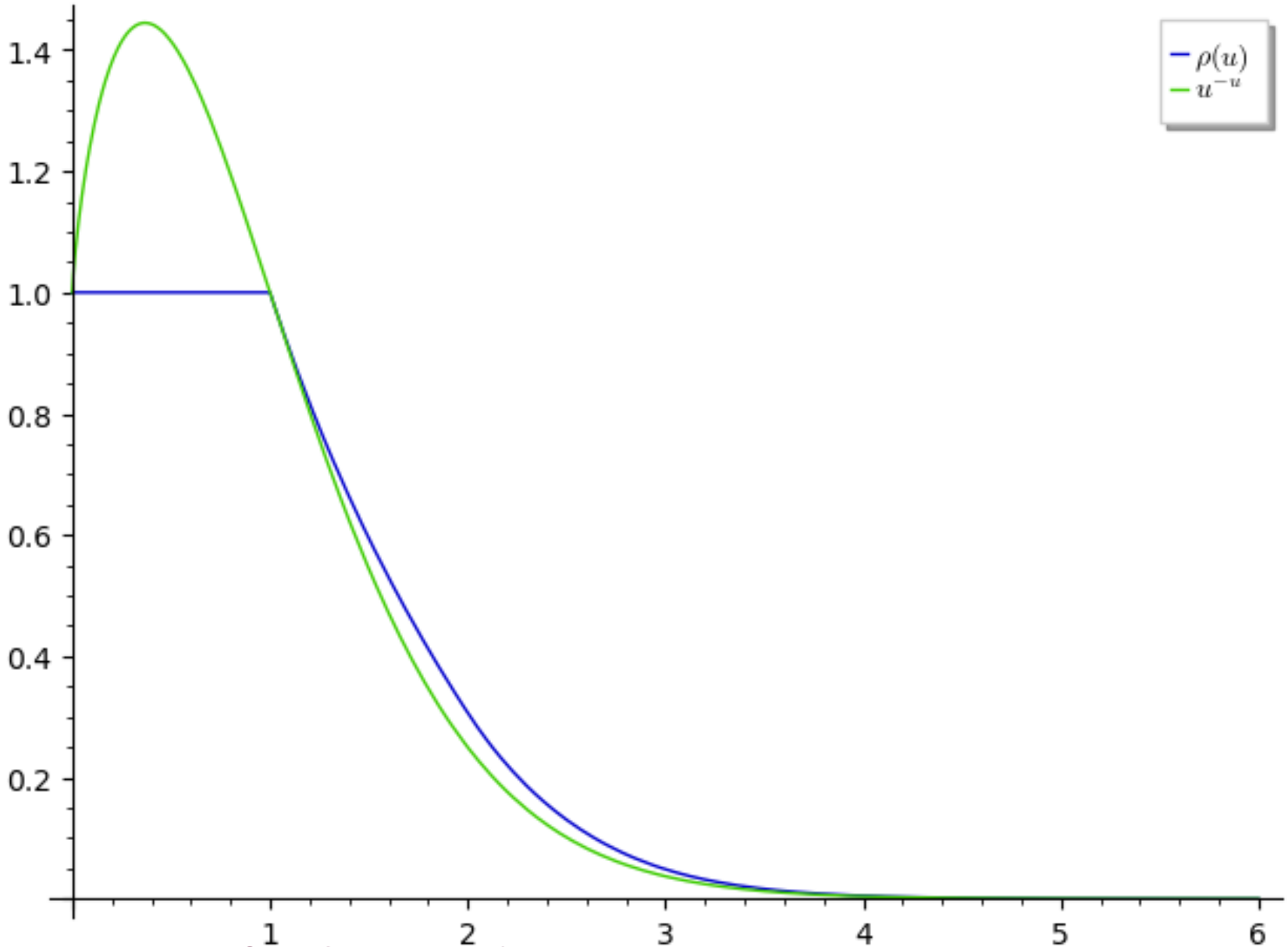
```
sage: plot(dickman_rho,x,0,10)
sage: plot(dickman_rho,x,0,10,scale='semilogy')
```

The asymptotic estimation of $\rho$ is a pain, however, and this is what we use for the analysis of sieving algorithms:

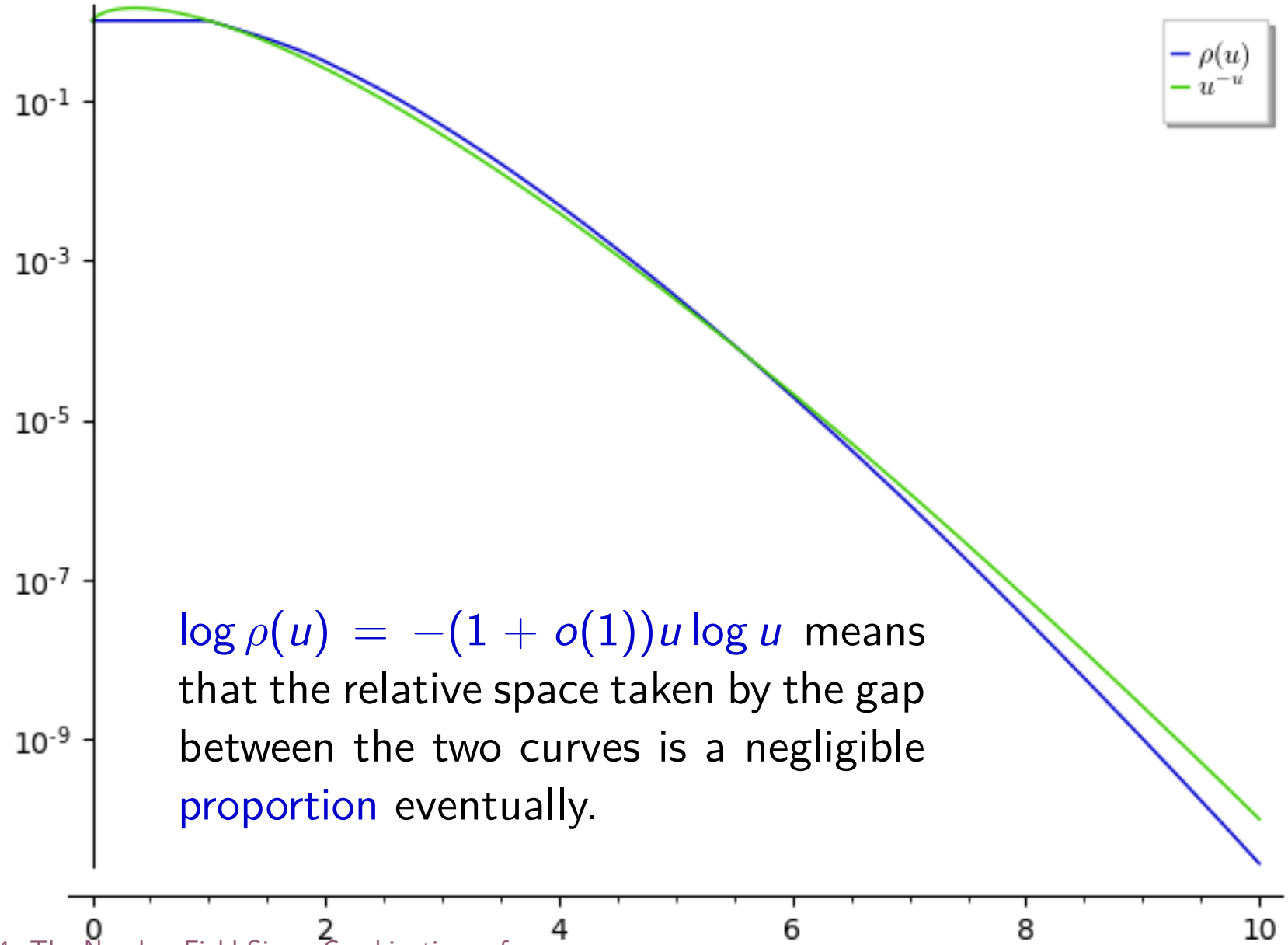
$$\rho(u) = u^{-u(1+o(1))}.$$

Note that $(1 + o(1))$ is quite inaccurate.

# $\rho(u)$



# $\rho(u)$ (log scale in $y$)




$\log \rho(u) = -(1 + o(1))u \log u$ means that the relative space taken by the gap between the two curves is a negligible proportion eventually.

# The $L$ function

The following notation is attributed to R. Schroeppel.

$$L_x[a, \alpha] = \exp\left(\alpha(\log x)^a(\log\log x)^{1-a}\right).$$

## CEP with the $L$ function

A random integer $n \leq L_x[a, \alpha]$ is $L_x[b, \beta]$-smooth with probability:

$$L_x\left[a - b, -\frac{\alpha}{\beta}(a - b)(1 + o(1))\right].$$

This formulation is very important for analyzing sieve algorithms.

# Calculus with $L_x[a,\alpha] = \exp\left(\alpha(\log x)^a(\log\log x)^{1-a}\right).$

## Basic formulae with $L$

$$L_x[a,\alpha] \times L_x[b,\beta] = \begin{cases} L_x[a,\alpha+o(1)] & \text{if } a > b, \\ L_x[b,\beta+o(1)] & \text{if } b > a, \\ L_x[a,\alpha+\beta] & \text{if } a = b. \end{cases}$$

$$L_x[a,\alpha] + L_x[b,\beta] = \begin{cases} L_x[a,\alpha+o(1)] & \text{if } a > b, \\ L_x[b,\beta+o(1)] & \text{if } b > a, \\ L_x[a,\max(\alpha,\beta)] & \text{if } a = b. \end{cases}$$

$$L_x[b,\beta]^{\log_{\log x} L_x[a,\alpha]} = L_x[a+b,\alpha\beta].$$

$$L_{L_x[b,\beta]}[a,\alpha] = L_x[ab,\alpha\beta^a b^{1-a}+o(1)].$$

# Computation model

Analysis is done in the RAM model: memory access is for free.

Of course, given the leeway in the final asymptotic complexity estimate, this does not matter much.

# Vanilla QS

Let $A = L_N[a, \alpha]$ be the bound on $i$.
Let $B = L_N[b, \beta]$ be the smoothness bound.

The number of collected relations is:

$$\underbrace{L_N[a, \alpha]}_{\text{exhaust } i} \times (\text{smoothness prob.}).$$

The cost of collecting relations (with detection by sieving) is:

$$L_N[a, \alpha] \times (\log\log B + (\text{smoothness prob.}) \times (\text{factoring time})).$$

The cost of linear algebra ($\pi(B)$ equations and unknowns) is:

$$(L_N[b, \beta + o(1)])^{\omega} = L_N[b, \omega\beta + o(1)].$$

# Collecting relations

Assume $a < 1$. We have

$$\max_{-A \le i \le A} |f(i)| = 2A\sqrt{N} = L_N[1, 1/2 + o(1)].$$

Smoothness probability is $L_N[1-b, -\frac{1}{2\beta}(1-b)(1+o(1))]$.

We want enough relations:

$$L_N[a, \alpha] \times L_N[1-b, -\frac{1}{2\beta}(1-b)(1+o(1))] = L_N[b, \beta + o(1)].$$

In particular:

- we need $\beta > 0$, therefore we must have $a \ge 1 - b$.
- we cannot have $b < 1/2$: an LHS-RHS match would be impossible.

So let's try $a = b = 1/2$.

# With $a = b = 1/2$

We want enough relations:

$$L_N[1/2, \alpha] \times L_N[1/2, -\frac{1}{4\beta}(1 + o(1))] = L_N[1/2, \beta + o(1)],$$

$$\alpha - \frac{1}{4\beta}(1 + o(1)) = \beta + o(1),$$

$$\alpha = (\beta + \frac{1}{4\beta})(1 + o(1)).$$

Note that this implies in particular $\alpha > \beta$.

# Still some work to do

There are still a few undecided things.

- How much does it cost to factor the smooth values?
  Options: resieving, TD, ECM, Batch smoothness detection.
- How much does the linear algebra cost?
  Options: Gauss, or (later developed) sparse linear algebra.

# TD + $\omega \geq 2$

Assume we do TD for each smooth $f(i)$.

- Relation collection: $A \log\log B + \pi(B)^2$.
- Linear algebra $\pi(B)^\omega$.

Total cost $L_N[1/2, \max(\alpha, 2\beta, \omega\beta)(1 + o(1))]$. Minimize as follows:

$$\begin{aligned} \alpha &= \omega\beta, \\ \beta + 1/(4\beta) &= \omega\beta, \\ 4(\omega - 1)\beta^2 &= 1, \\ \beta &= \frac{1}{2\sqrt{\omega - 1}}. \end{aligned}$$

So that the complexity (with TD) is $L_N[1/2, \frac{\omega}{2\sqrt{\omega-1}}(1 + o(1))$.

# Factoring does not matter

TD is not a very smart mechanism.

Yet, as long as we use sieving to detect the smooth values $f(i)$, what algorithm we use to actually factor them does not matter.

However, if even sieving were to be replaced by some exponential algorithm, the complexity would be different.

The complexity of QS is

$$L_N[1/2, \frac{\omega}{2\sqrt{\omega - 1}}(1 + o(1))].$$

If $\omega = 2$, this becomes $L_N[1/2, 1 + o(1)]$.

# Impact of improvements

Many improvements were made to MPQS. Some are rather minor, and some had a major practical impact.

What is their impact on the complexity?

Take MPQS. $|f(i)|$ drops from $2A\sqrt{N}$ to $\frac{1}{\sqrt{2}}A\sqrt{N}$.

In both cases, this is $L_N[1, 1/2 + o(1)]$.

# Impact of improvements

Many improvements were made to MPQS. Some are rather minor, and some had a major practical impact.

What is their impact on the complexity?

Take MPQS. $|f(i)|$ drops from $2A\sqrt{N}$ to $\frac{1}{\sqrt{2}}A\sqrt{N}$.

In both cases, this is $L_N[1, 1/2 + o(1)]$.

## Disappointment

None of the practical improvements to QS (even MPQS) has the slightest impact on its asymptotic complexity.

# Wrap-up

- Combination of congruences.
- QS successfully introduces sieving.
- Many improvements, some minor, some major.
  MPQS, special-Q.
- Massive distribution becomes possible (MPQS).
- Analysis yields $L_N[1/2, 1 + o(1)]$, but unfortunately most improvements are invisible.