# CSE291-14: The Number Field Sieve

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

Emmanuel Thomé

February 1, 2022

# Part 4b

# Polynomial selection in NFS

Kleinjung's 2008 algorithm

Size optimization with translation and rotation

# Plan

Kleinjung's 2008 algorithm

Size optimization with translation and rotation

# Kleinjung's 2008 algorithm

New polynomial selection algorithm, presented at a workshop in Nancy in 2008.

- Better than the previous algorithm.
- Has never been published.
- Implemented in Cado-NFS.

Some features of this algorithm:

- A new way to make $a_{d-2}$ small by construction.
- Yields very large skewness values.
- Large skewness is great for root-optimization (see later).

# Handy tool

How to find polynomials pairs of the form:

$$f_1 = x^d + a_{d-2}x^{d-2} + \cdots + a_0,\ f_0 = m_1 x - m_0,\ \mathrm{Res}(f_0, f_1) = N.$$

**Remark**: we are looking for $a_d = 1$ and $a_{d-1} = 0$.

We want $a_{d-2}$ small. Notation: $R = a_{d-2}m_0^{d-2} + \cdots + a_0 m_1^{d-2}$, so

$$N = \mathrm{Res}(f_0, f_1) = f(m_0/m_1)m_1^d = m_0^d + m_1^2 R$$

## Reformulation

Put otherwise, given $(N, d)$ we want $(m_1, m_0)$ such that:

$$m_1^2 \mid (N - m_0^d), \quad \frac{\left|N - m_0^d\right|}{m_1^2 m_0^{d-2}} = \frac{|R|}{m_0^{d-2}} \approx a_{d-2} \text{ small.}$$

Divisibility + $m_0 \approx \sqrt[d]{N}$ only give $R \approx d m_0^{d-1} m_1/m_1^2 \approx$ not small!

# Solving the auxiliary problem

Set $\bar{m}_0 = \left\lceil \sqrt[d]{N} \right\rceil$. Good $m_0$'s are close to $\bar{m}_0$.

- Let $\mathcal{P} = [P, 2P]$ be a range of prime numbers.
- List pairs

$$(p \in \mathcal{P}, i \in [-4P^2, +4P^2]) \text{ s.t. } p^2 \mid (N - (\bar{m}_0 + i)^d).$$

- Sort w.r.t. $i$. Find collisions.
- Each collision $(p_1, i), (p_2, i)$ yields $m_1 = p_1 p_2,\ m_0 = \bar{m}_0 + i$.

Applying Kleinjung "Lemma 2.1" to $N$, $d$, $m_1$, $m_0$, $a_d$ and $a_{d-1} = 0$ gives:

$$|a_{d-2}| \leq \frac{4dm_0}{P^2} + 2P^2.$$

With this approach, we can obtain polynomials with $a_d = 1$ and $a_{d-1} = 0$. But this isn't so appealing.

# More general situation

How can we apply previous tool to more general polynomials ?

$$f_1 = a_d x^d + a_{d-1} x^{d-1} + \cdots, \; f_0 = m_1 x - m_0, \; \mathrm{Res}(f_0, f_1) = N.$$

**Goal**: $N = m_1^d f(m_0/m_1) = a_d m_0^d + a_{d-1} m_0^{d-1} m_1 + m_1^2 R$ and $\frac{|R|}{m_0^{d-2}}$ small.

Consider the polynomial $f_1(x - \frac{a_{d-1}}{d a_d})$.

$$d^d a_d^{d-1} N = (d a_d m_0 + a_{d-1} m_1)^d + m_1^2 \left( d^d a_d^{d-1} R - \cdots \right).$$

## Reduction to previous problem

Define $N' = d^d a_d^{d-1} N$, and $m_0' = (d a_d m_0 + a_{d-1} m_1)$.

- Fix $a_d$ and compute $N'$.
- Search for $m_1, m_0'$ as solutions to previous problem.
- From a winning $m_0'$, find appropriate $m_0$ and $a_{d-1}$.

# Time-consuming steps

When running Kleinjung's 2008 algorithm, a computationally expensive part is the search for collisions.

- Naive:
  - list many pairs $(p, i)$.
  - sort w.r.t. $i$, and see if we have $(p_1, i)$ and $(p_2, i)$.
- In practice:
  - Generate many pairs. Dispatch only the $i$ values in lists indexed by, e.g. $\lfloor i/2^{16} \rfloor$:
    ```
    H[i // 2^16].append(i % 2^16)
    ```
  - For each list, hit an array of $2^{16}$ memory bytes with the values in the list, and report if a common $i$ is found.
  - When we do find a collision (a priori rarely), start over more cautiously.
  - And then there's possibly some extra work to do once we find $(p_1, p_2)$ too.

# Time-consuming steps

Another avenue for improvement reflects on the modular computations of the roots of $N - x^d \mod p_i$.

This can be amortized with a technique that reminds a bit of the special-$q$ technique. Aim at $m_1 = p_1 p_2 q$ with fixed $q$.

# Plan

Kleinjung's 2008 algorithm

Size optimization with translation and rotation

# Size optimization – main idea

A "raw" polynomial pair (i.e., generated by one of the previous algorithms) may be optimized:

- translation: change $(f_0, f_1)$ into
$$(f_0(X+t), f_1(X+t)),\ t \in \mathbb{Z}.$$
- rotation: change $(f_0, f_1)$ into
$$(f_0, f_1 + rf_0),\ r \in \mathbb{Z}[X] \text{ with } \deg r < \text{some bound}.$$
This is subject to multiple constraints that depend on the polynomial pair we started with.

## Goal of size-optimization

Given $(f_0, f_1)$, find $t \in \mathbb{Z}$ and $r \in \mathbb{Z}[X]$ of degree less than $d$ that minimize some size estimator for $f_1(X+t) + rf_0(X+t)$.

**Remark**: $\mathrm{Res}(f_1(X+t) + rf_0(X+t), f_0(X+t)) = \mathrm{Res}(f_0, f_1)$.

# Size optimization – local descent

Hard to find the global minimum.

We have to settle for a local minimum.

**Algo**: start with the raw pair $(f_0, f_1)$ and apply a local descent algorithm:

- apply a small translation or small rotation that reduce the norm;
- repeat until a (local) minimum is reached.

Works fine for $d \leq 5$.

Implemented in Cado-NFS. Used as a building block for others algorithms.

# Size optimization – larger degree

For larger degrees, the local descent algorithm is often stuck in minima close to the starting polynomial pair.

Ideas for improvment:

- Apply some initial translations before calling the local descent algorithm to increase the number of starting polynomial pairs and to avoid being stuck in local minima too far away from the global minimum.
- Apply the LLL algorithm before calling the local descent algorithm to pre-optimize the starting polynomial pair.

# Size optimization – initial translations

How to choose the initial translations ?

Choose integer approximations of roots of $\tilde{a}_{d-3}$, where the $\tilde{a}_i$'s are polynomials in $t$ defined by

$$f_1(X+t) = \sum_{i=0}^{d} \tilde{a}_i(t) X^i.$$

The polynomial $\tilde{a}_{d-3}$ has degree 3 so it has at least one real root.

**Example**: for $d = 6$, $\tilde{a}_3(t) = 20a_6t^3 + 10a_5t^2 + 4a_4t + a_3$.

In Cado-NFS, other methods are also used:

- constants values: $i \times 10^j$ for small integers $i$ and $j$
- more details in "Better polynomials for GNFS" and in the source code.

# Optimization with LLL: the BBKZ algorithm

(Bai, Bouvier, Kruppa, Zimmemann)

**Idea**: Use the LLL algorithm to search for short vectors in the lattice spanned by the coefficients vector (of length $d+1$) of $f_1, f_0, Xf_0, X^2 f_0, \ldots$.

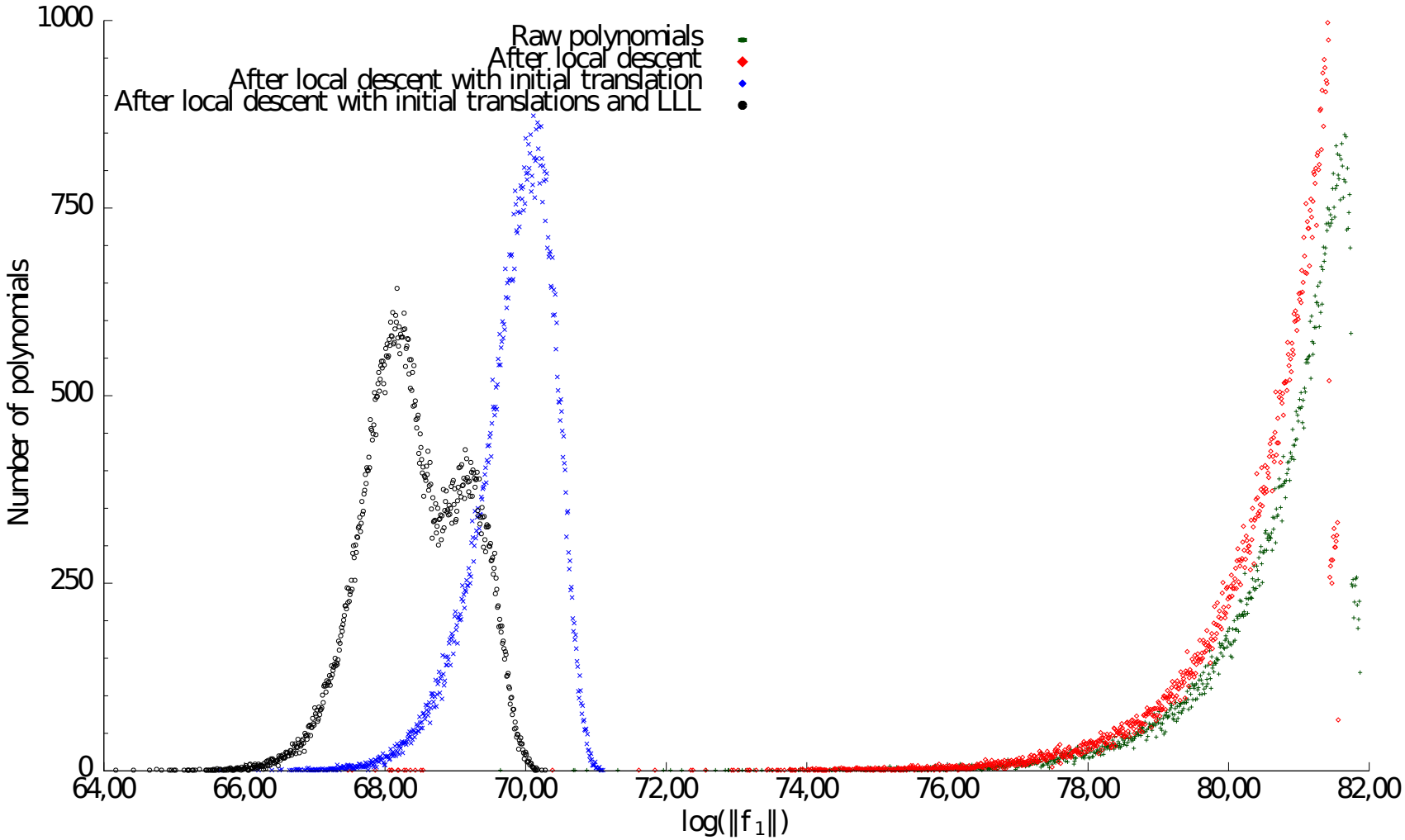
A vector of this lattice corresponds to a polynomial of the form $cf_1 + rf_0$, with $c \in \mathbb{Z}$ and $r$ an integer polynomial.

New degree of freedom: will output polynomial pair $(\tilde{f}_0, \tilde{f}_1)$ such that $\mathrm{Res}(\tilde{f}_0, \tilde{f}_1) = c\,\mathrm{Res}(f_0, f_1)$. With previous methods, we always had $c = 1$.

This new method is used before the local descent algorithm and after the computation of the initial translations.

New initial translations can be computed to take advantage of this new degree of freedom.

# Results – RSA-768 ($d = 6$)



# CSE291-14: The Number Field Sieve

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

Emmanuel Thomé

February 1, 2022

# Part 4c

## Polynomial selection in NFS: estimators

The sieved range

Valleys and the starfish picture

Root properties

Forcing good $\alpha(f)$: the root optimization

Polynomial selection in Cado-NFS

# Recap from last time

So far, we've only been interested in the size of the coefficients of our polynomials.

**(skewed) (infinity) norm of a polynomial $f = \sum a_i x^i$**

$$\|f\|_S = \|f\|_{\infty,S} = \max_i |a_i S^{i-(\deg f)/2}|.$$

It is ok as a first estimator, but can we do better?

- run the Number Field Sieve. It's costly.
- run a few sieving experiments.
  Could be well-suited to $\sim 10$ contenders.
- use other estimators, accurate/slow ones vs crude/fast ones.

# Polynomial selection workplan

- Use our favorite method to produce polynomials with small coefficients, that is small (skewed) infinity norm. Use size-optimization on each pair.
- Collect many of them (thousands, millions), compute some slightly more accurate quality estimator. Rank the results.
- Perhaps refine the results with an even more accurate estimator.
- Perhaps run another range of optimizations, if relevant.
- Eventually keep about a dozen polynomials, and do sieving tests on each.

# Plan

The sieved range

Valleys and the starfish picture

Root properties

Forcing good $\alpha(f)$: the root optimization

Polynomial selection in Cado-NFS

# Notations

Notations: $(f_0, f_1), \deg f_1 = d$, $\phi(x) = a - bx$, coefficients of $f_i$ unnamed.

# The sieved values

## Reminder

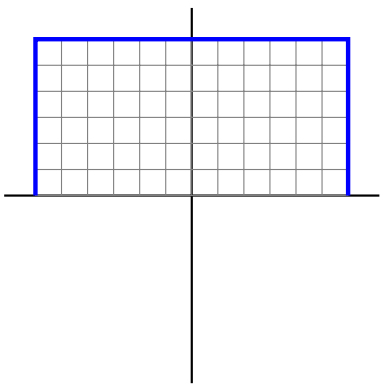
Given polynomials $(f_0, f_1)$, we search for pairs $(a, b)$ such that:

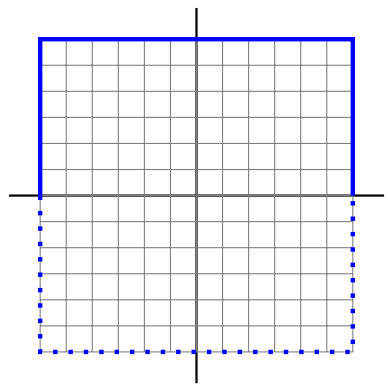
- $|a| \leq A$ and $0 \leq b \leq A$
  (no need to examine both $(a, b)$ and $(-a, -b)$)
- For both $i = 0$ and 1, $\mathrm{Res}(a - bx, f_i) = F_i(a, b)$ is smooth.

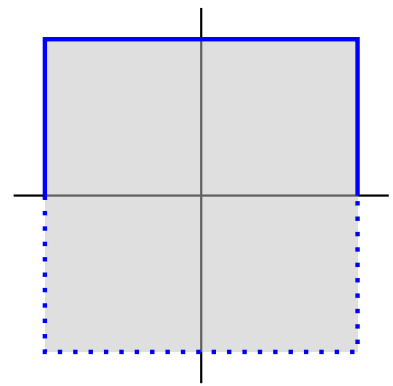
We are interested in the values taken by the bivariate, homogenous polynomials $F_0$ and $F_1$:

- on the sieve region $[-A, A] \times [0, A]$.
- or more simply on $[-A, A]^2$, since the two problems are equivalent.
- Yes, we're losing track of the distinction between $\mathbb{Z}^2$ and $\mathbb{R}^2$.

# Sieved range







# More estimations

Count the smoothness probability for all norms obtained from $[-A, A]^2$ (want to maximize):

$$\frac{6}{\pi^2} \iint_{[-A,A]^2} \rho\left(\frac{\log|F_0(x,y)|}{\log B_1}\right) \rho\left(\frac{\log|F_1(x,y)|}{\log B_2}\right) \mathrm{d}x\,\mathrm{d}y.$$

$\rho$ is the Dickman function. It is used to estimate the smoothness probability.

$B_0$ and $B_1$ are the smoothness bounds.

This adds the idea of averaging the values over the sieve region.

- It is more accurate than the infinity norm.
- Alas, it is **too costly**. No explicit formula for the function $\rho$.

# $L^2$ rectangular norm

We want to have some notion of average norm over $[-A, A]^2$.

By homogeneity, $F_i(\lambda X, \lambda Y) = \lambda^{\deg f_i} F_i(X, Y)$.
Only $[-1, 1] \times [-1, 1]$ really matters.

## $L^2$ norm

For $F$ being either $F_0$, $F_1$, or the product of both, we want to minimize:

$$\iint_{[-A,A]^2} F(x,y)^2 \,\mathrm{d}x\,\mathrm{d}y = A^{2\deg F} \iint_{[-1,1]^2} F(x,y)^2 \,\mathrm{d}x\,\mathrm{d}y$$

This $L^2$-norm is much, much easier to compute.
We use as our second main criterion of selection.

# Computing the $L^2$ rectangular norm

```
var('X Y')
R.<x> = PolynomialRing(ZZ)
IPR.<a> = InfinitePolynomialRing(R)
d = 6;
f = SR(sum(a[i]*x^i for i in range(d+1)))
F = (f(x=X/Y)*Y^d).expand()
v = integrate(integrate(F^2,(X,-1,1)),(Y,-1,1)).expand()
v.coefficients()
```

This $L^2$ rectangular norm is expressed as a simple expression involving the coefficients.

# Sieved range vs skewness

## If we have skewness $S$

- $|a| < A\sqrt{S}$ and $0 \leq b < A/\sqrt{S}$
- We want $\mathrm{Res}(a - bx, f_i) = F_i(a, b)$ to be smooth.

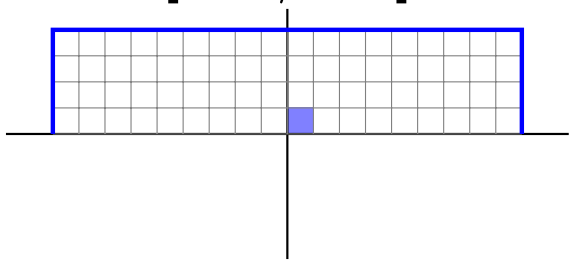
Let $F_{i,S}(X, Y) = F_i(X\sqrt{S}, Y/\sqrt{S}) \in \mathbb{R}[X, Y]$.

Again, looking at $F_{i,S}$ over $[-1, 1]^2$ is enough.

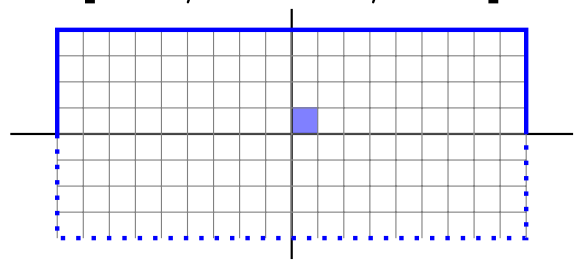
A squeeze mapping brings us back to the non-skewed case.
This will be implicit from now on.

# Squeeze mapping

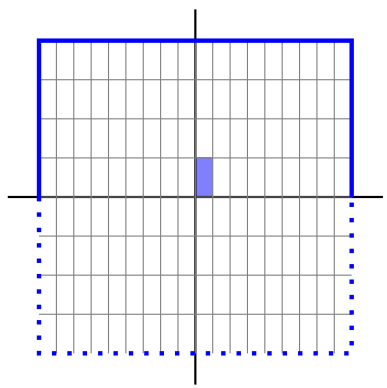
$[-A\sqrt{S}, A\sqrt{S}] \times [0, A/\sqrt{S}]$

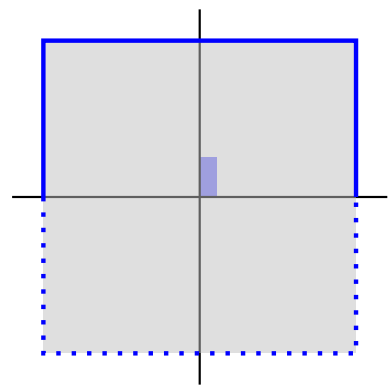


$[-A\sqrt{S}, A\sqrt{S}] \times [-A/\sqrt{S}, A/\sqrt{S}]$



Squeeze by $\begin{pmatrix} 1/\sqrt{S} & 0 \\ 0 & \sqrt{S} \end{pmatrix}$: the unit area maps to area 1.





# A digression: special-$q$ vs sieved range

When special-$q$ comes into play, something changes.

## Quick refresh about special-$q$

Special-$q$ is this idea of restricting our attention to situations where there is a known factor $q$ in the things that we want to be smooth.

We can use special-$q$:

- to force a factor $q$ in the factorization of $am_1 - bm_0$.
- or to force a certain prime ideal above $q$ to appear in the factorization of $a - b\alpha$.

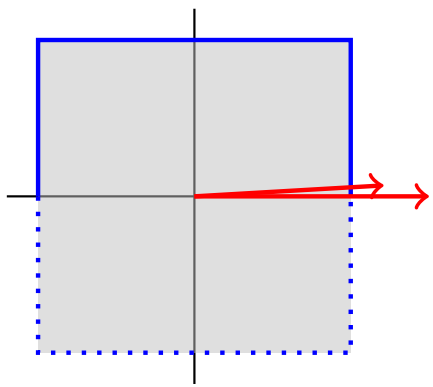
# Special-$q$

How do we force $q$ to divide $am_1 - bm_0$?

## Here is a lattice

The set of $(a, b)$ such that $q \mid am_1 - bm_0$ is a lattice (say $\mathcal{L}_q$).

Basis (if $\gcd(m_1, q) = 1$): $\begin{cases} (a_0, b_0) = (q, 0) \\ (a_1, b_1) = ((m_0/m_1) \bmod q, 1) \end{cases}$



combinations of the basis vectors quickly get out of hand.

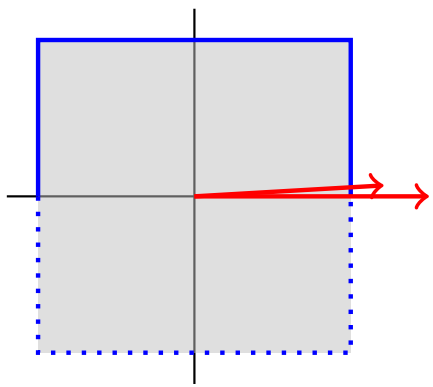
# Special-$q$

How do we force $q$ to divide $am_1 - bm_0$?

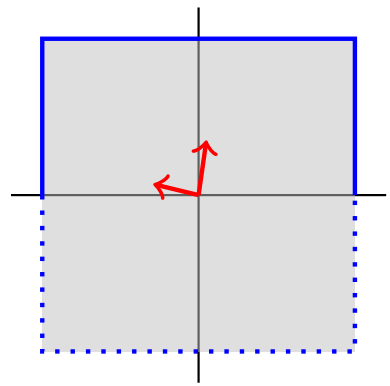
## Here is a lattice

The set of $(a, b)$ such that $q \mid am_1 - bm_0$ is a lattice (say $\mathcal{L}_q$).

Basis (if $\gcd(m_1, q) = 1$): $\begin{cases} (a_0, b_0) = (q, 0) \\ (a_1, b_1) = ((m_0/m_1) \bmod q, 1) \end{cases}$



combinations of the basis vectors quickly get out of hand.

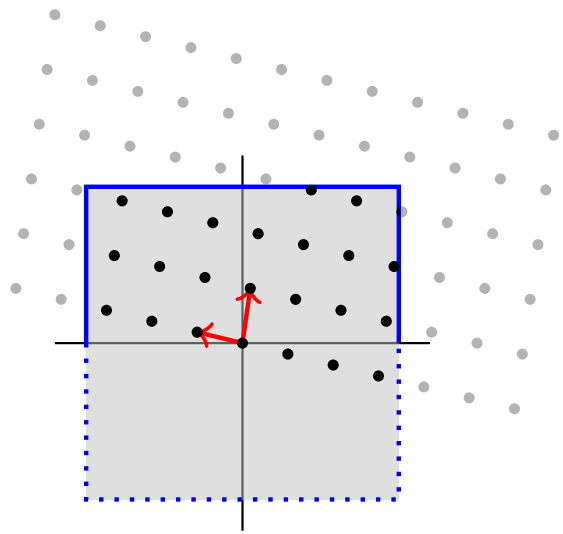


lattice reduction keeps things under control.

This approach is called lattice sieving.

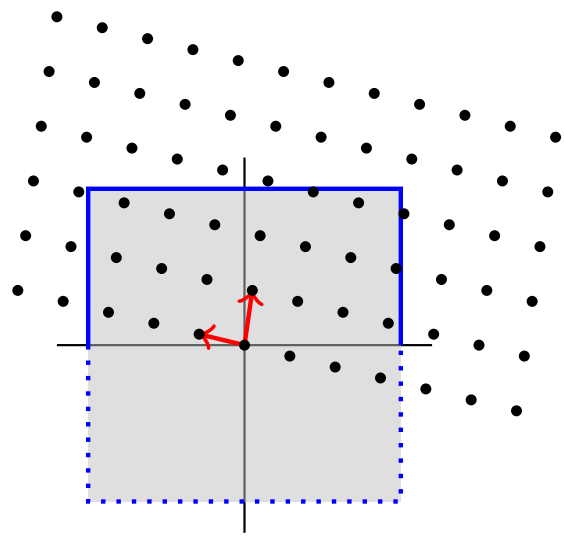
# Sieved range with lattice sieving

- Choose $q$.
- Compute a reduced basis of the lattice $\mathcal{L}_q$.
- Sieve through all the linear combinations of the basis vectors that fall within the target region.
  - Loop through all primes,
  - Mark the locations where they divide, etc.



# Sieved range with lattice sieving: better

- Fix in advance a set of combinations that we will explore each time. E.g. $[-2^{15}, 2^{15}) \times [0, 2^{15})$.
- Choose $q$.
- Compute a reduced basis of the lattice $\mathcal{L}_q$.
- Sieve through all the linear combinations of the basis vectors, whether or not they fit in our target range.



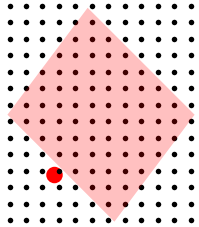
# Why would we do that?

Legitimate fear: the set of $(a, b)$ that we will sieve go way off range. But would they, really?

# Sieving rectangle

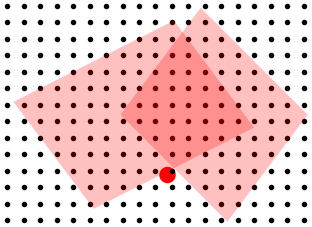
# Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many $q$'s.



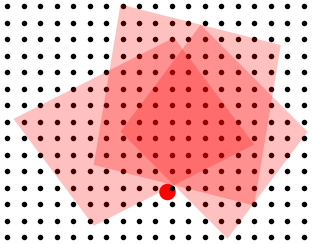
# Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many $q$'s.



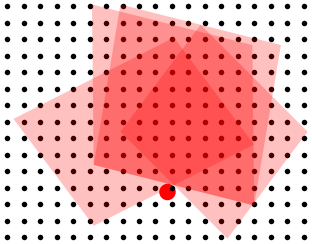
# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



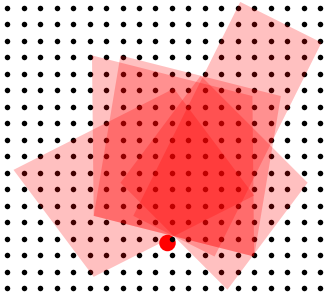
# Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many $q$'s.



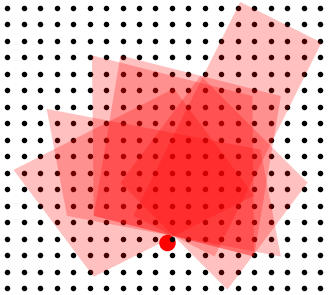
# Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many $q$'s.



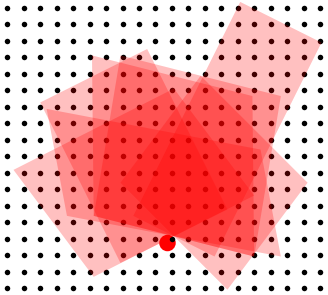
# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



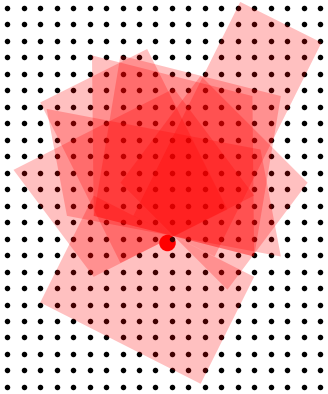
# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



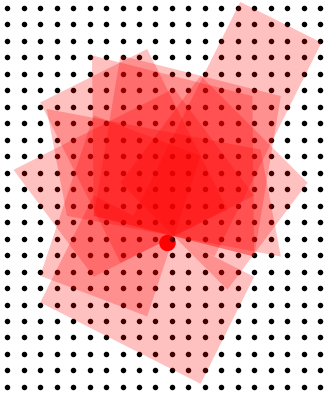
# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



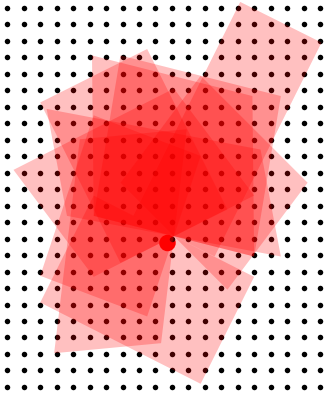
# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



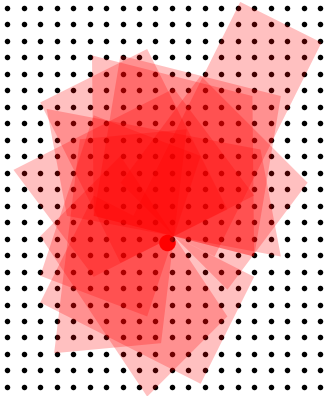
# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



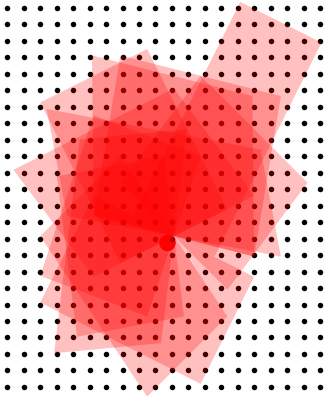
# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



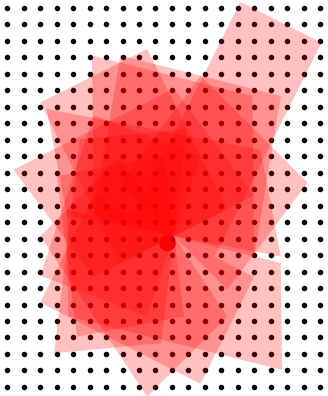
# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



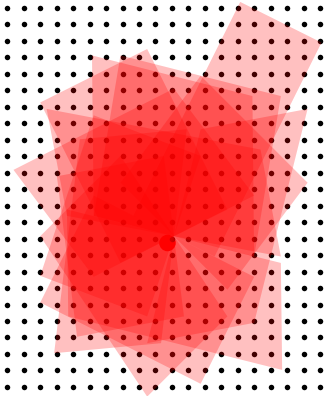
# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



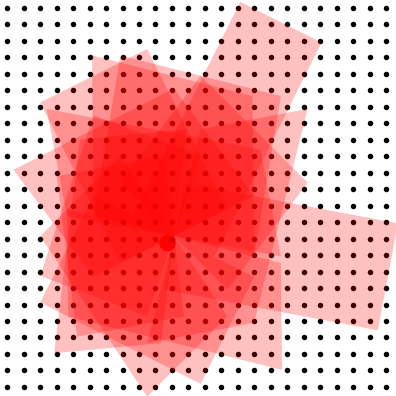
# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



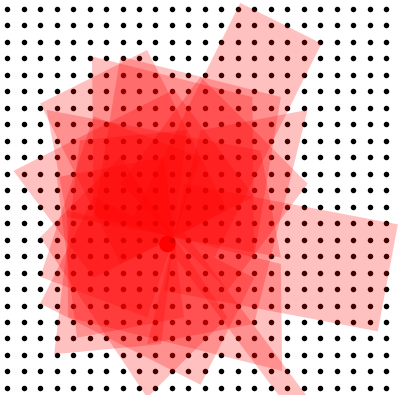
# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



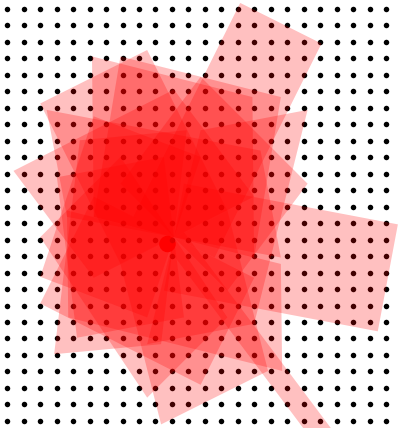
# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



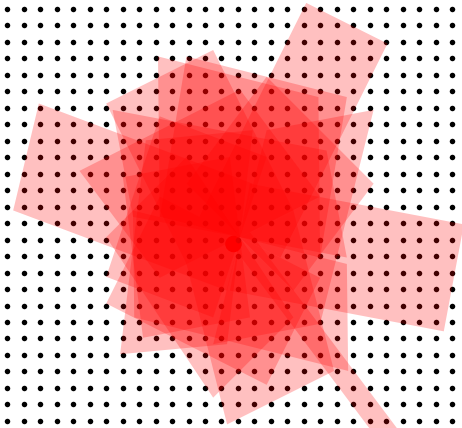
# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



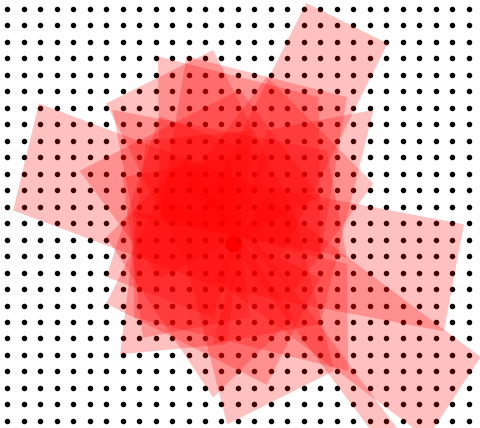
# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



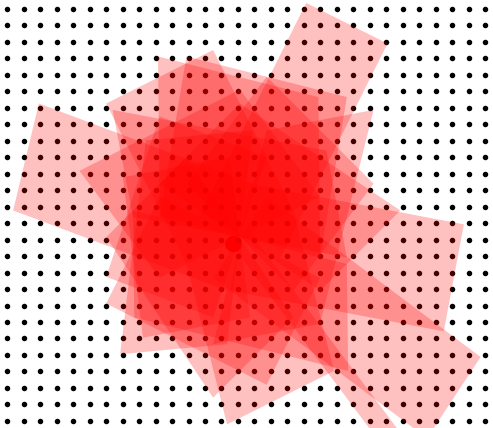
# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



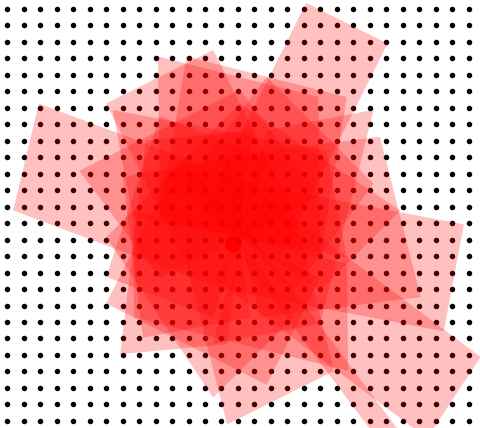
# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



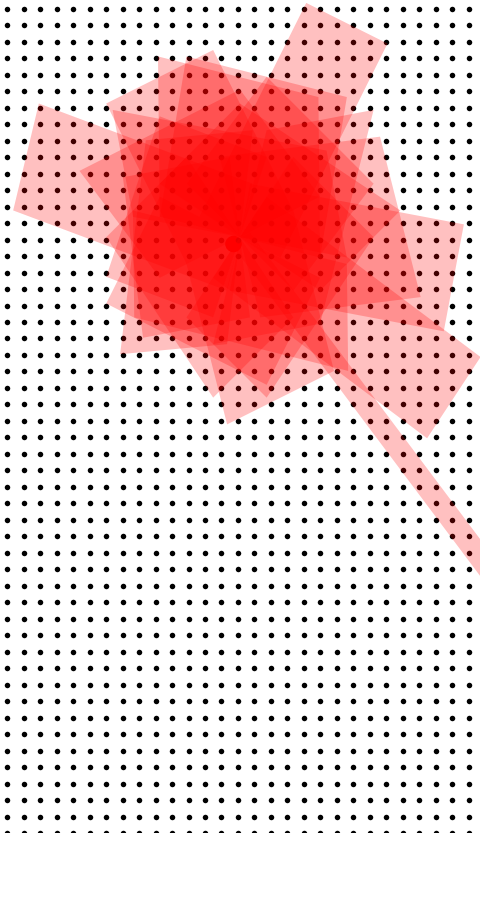
# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



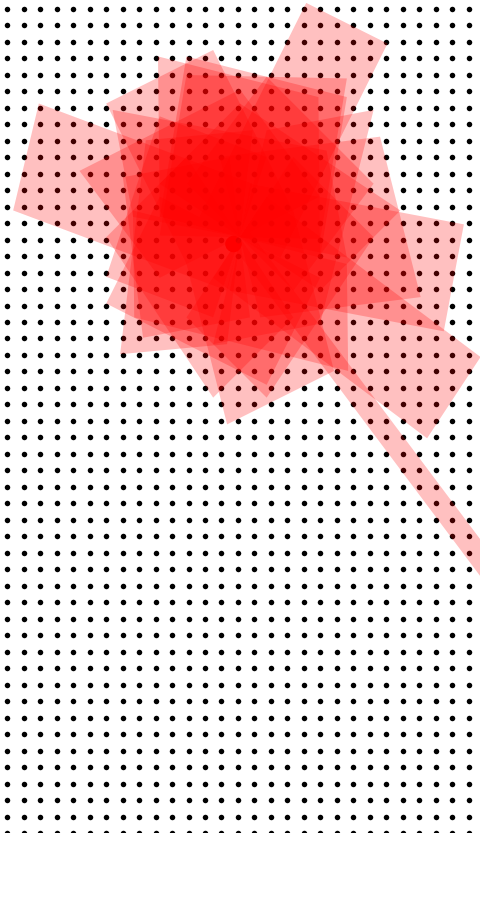
# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



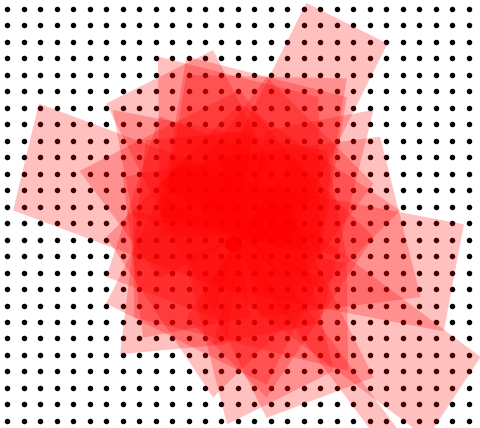
# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



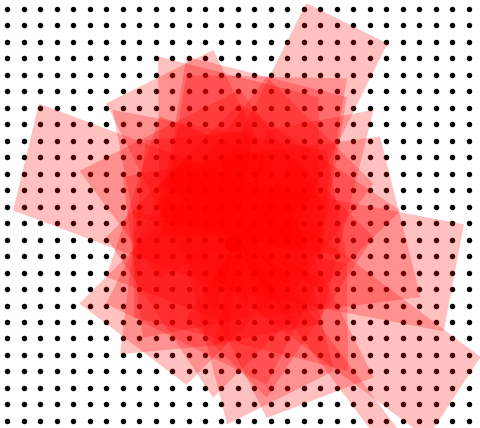
# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



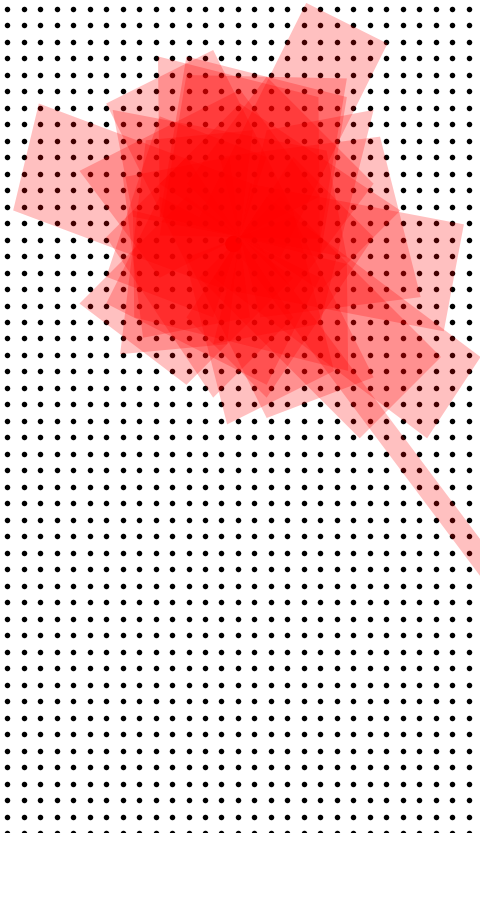
# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



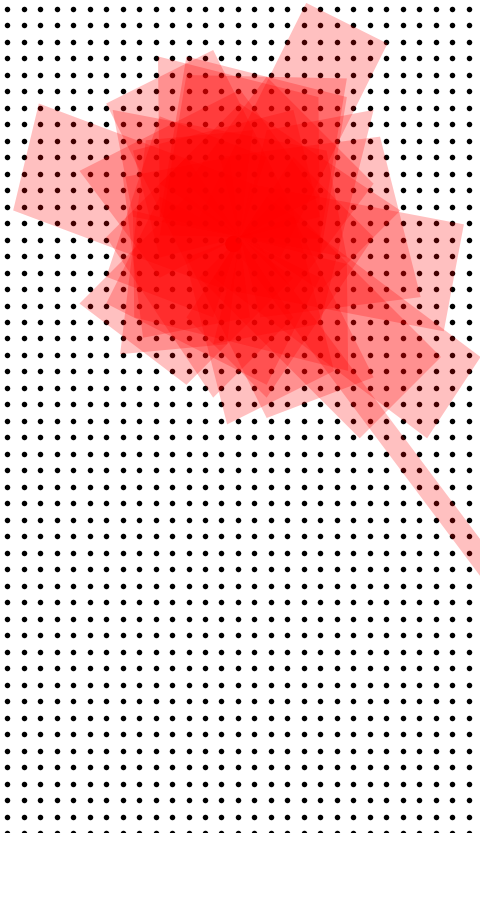
# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



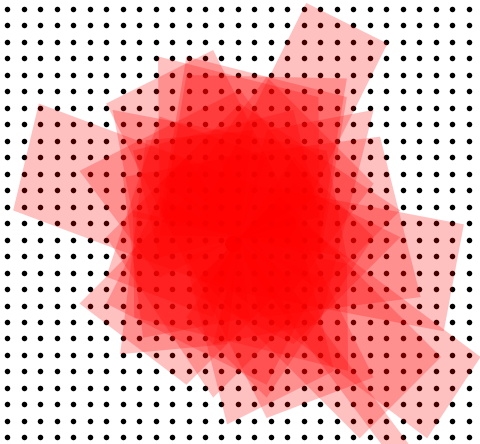
# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



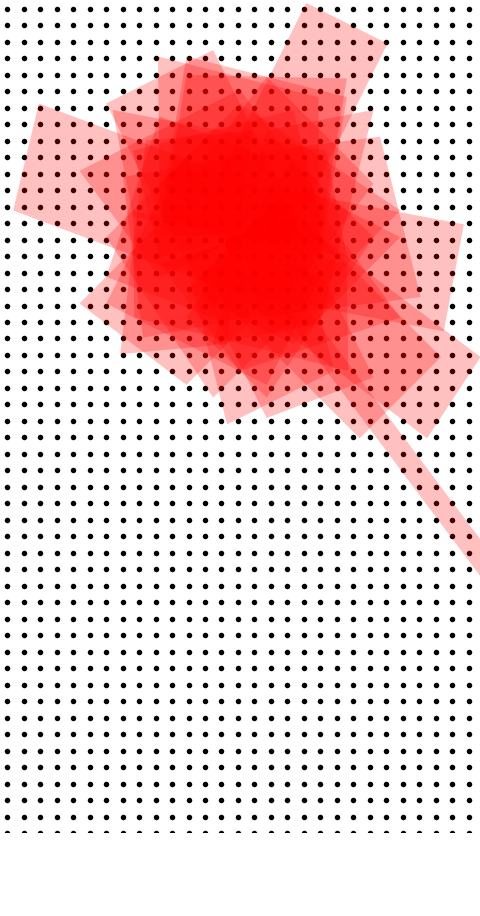
# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



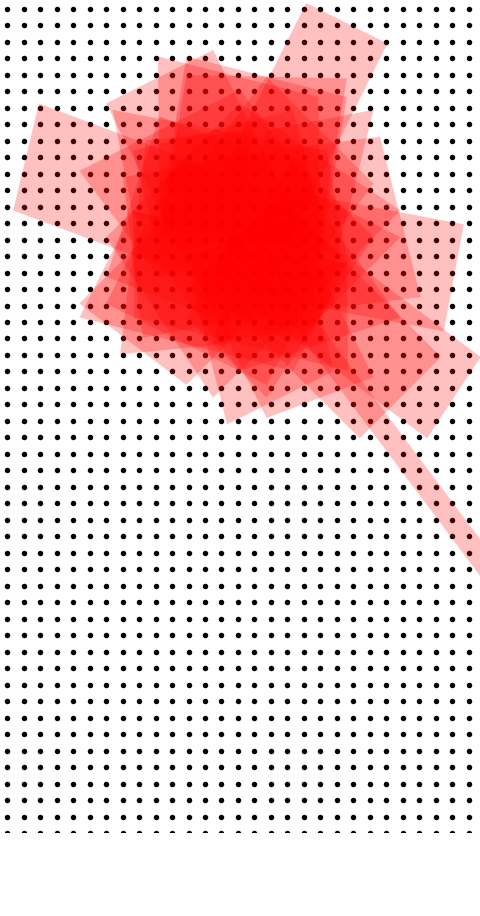
# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



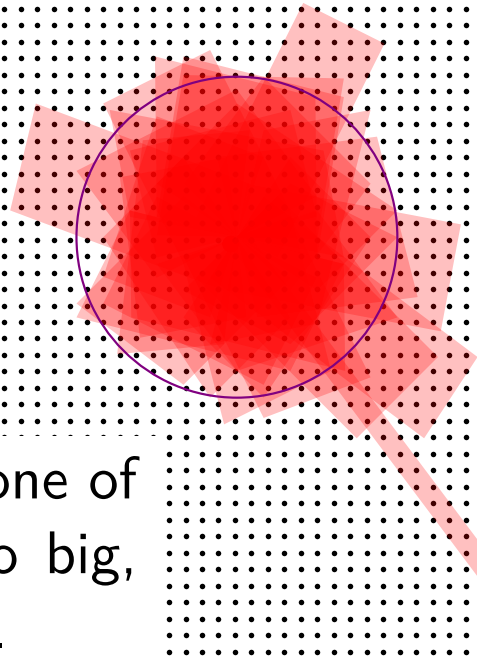
# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



# Sieving rectangle

Here, we depict the area of size $2^I \times 2^{I-1}$ for many $q$'s.



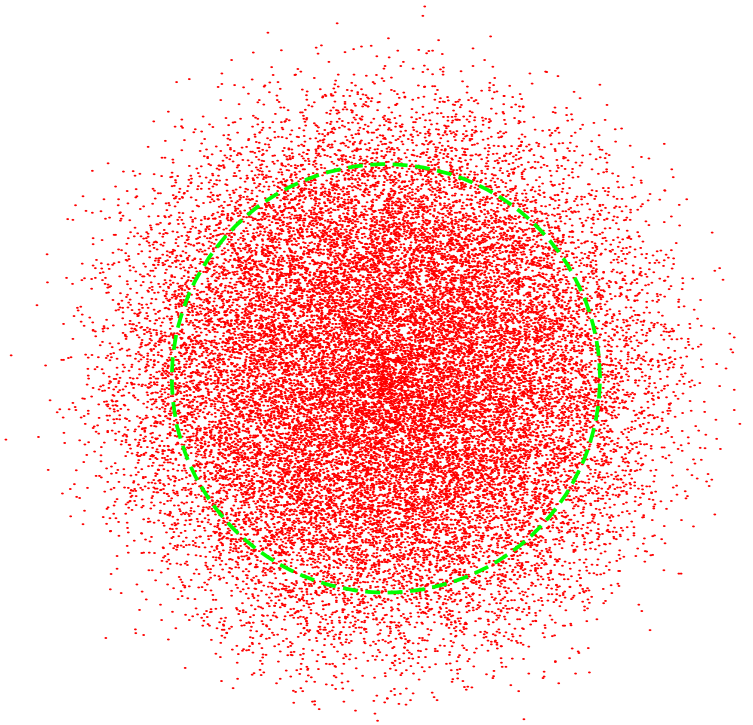
Except in rare cases where one of the basis vectors is way too big, things remain under control.

Idea by Franke and Kleinjung, early 2000s.

Notice that the region of reached $(a, b)$'s is now isotropic.

# $L^2$ circular norm

- In practice, we escape the rectangle.
- With lattice sieving: circle.



Use a circular integral:

$$\iint_{[0,1]\times[0,2\pi[} F(r\cos\theta, r\sin\theta)^2\, r\mathrm{d}r\,\mathrm{d}\theta$$

This $L^2$-norm is also easy to compute.

# Computing the circular $L^2$ norm

```
var('X Y r t')
R.<x> = PolynomialRing(ZZ)
IPR.<a> = InfinitePolynomialRing(R)
d = 6;
f = SR(sum(a[i]*x^i for i in range(d+1)))
F = f.subs(x=r*cos(t),y=r*sin(t))
v = integrate(integrate(F^2*r,(r,0,1)),(t,0,2*pi))
v.expand().collect(pi)
```

# $L^2$ circular norm – example

Example for circular $L^2$-norm for $d = 4$:

$$\|f\|^2 = \frac{\pi}{640}\Big(3a_2^2+5(a_3^2+a_1^2)+35(a_4^2+a_0^2) + 6(a_0a_4+a_1a_3)+10(a_4a_2+a_2a_0)\Big).$$

- Various formulæ for each degree (easy to hard-code).
- The fact that it is an $L^2$-norm may also be useful.

# Plan

The sieved range

Valleys and the starfish picture

Root properties

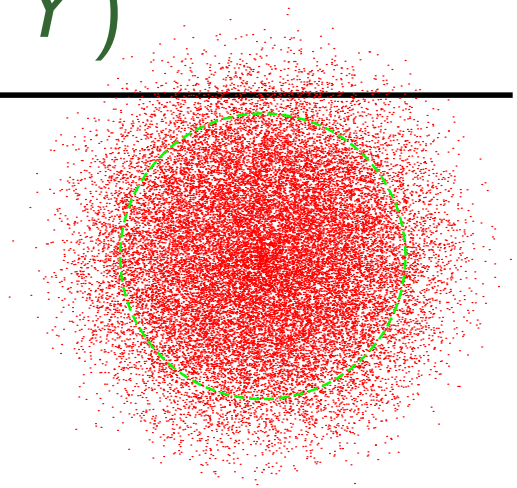
Forcing good $\alpha(f)$: the root optimization

Polynomial selection in Cado-NFS

# Notations

Notations: $(f, \cdot), \deg f = d, f = \sum f_i x^i$, $\phi(x) = a - bx$.
(We only care about one polynomial at a time, or conceivably their product if we feel like it)

# Small values of $F(X, Y)$



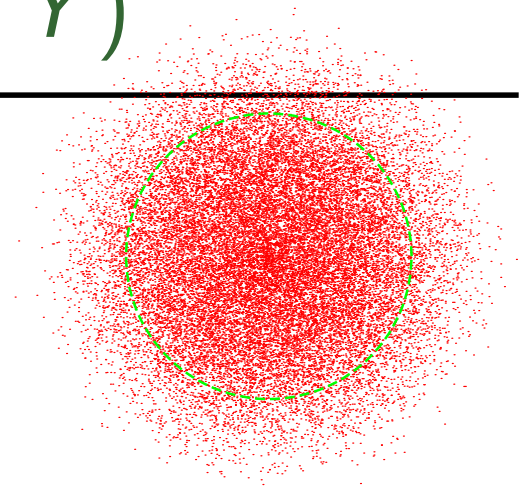
- The set of reached $(a, b)$ values is isotropic (skewness aside, as usual)

- For $(X, Y)$ on the unit circle: $F(X, Y) = Y^d f(X/Y) = (\sin\theta)^d \cdot f(1/\tan\theta)$.

On the unit circle, $F$ takes various values, small and large.

The smallest ones are when...

# Small values of $F(X, Y)$



- The set of reached $(a, b)$ values is isotropic (skewness aside, as usual)

- For $(X, Y)$ on the unit circle: $F(X, Y) = Y^d f(X/Y) = (\sin\theta)^d \cdot f(1/\tan\theta)$.

On the unit circle, $F$ takes various values, small and large.

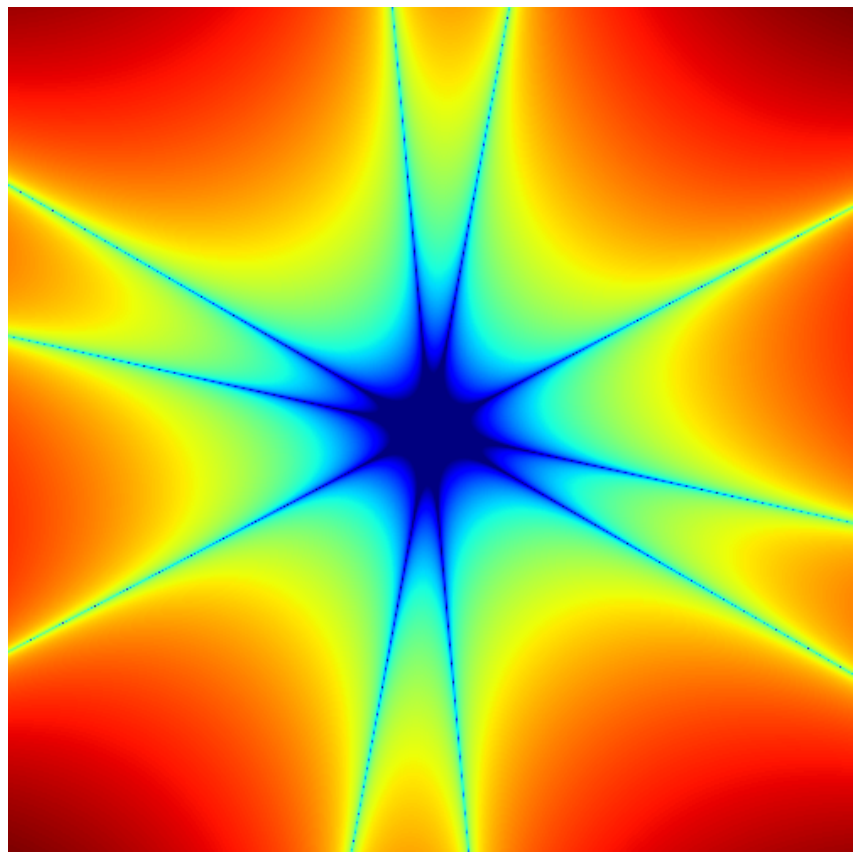
The smallest ones are when...
$X/Y = 1/\tan\theta$ is one of the real roots of $f$.

**$\log|F(a, b)|$**

The size of $\log|F(a, b)|$ depends on the number of real roots of $f$. More real roots = more places where we expect $F(a, b)$ to be very small and hence more likely to be smooth.

# The starfish-like picture

Let $f = 4x^5 + 17x^4 - 18x^3 - 58x^2 + 6x + 1$. Plot $\log|F(a, b)|$.

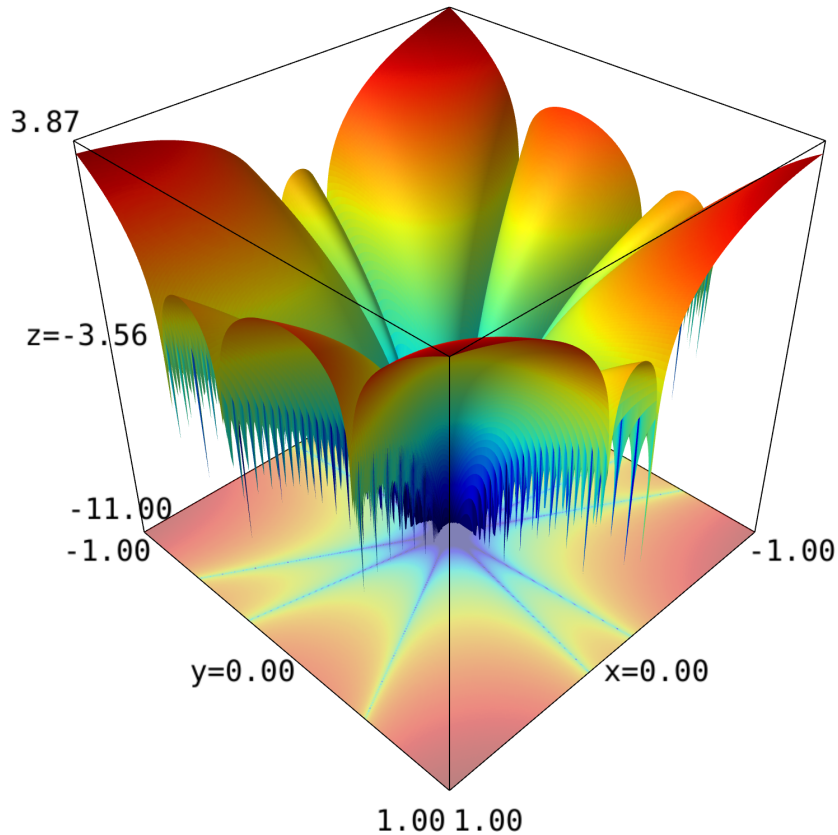


(sage code)

More real roots = hints about areas which are more worth looking into than others.

# The starfish-like picture

Let $f = 4x^5 + 17x^4 - 18x^3 - 58x^2 + 6x + 1$. Plot $\log|F(a, b)|$.




(sage code)

More real roots = hints about areas which are more worth looking into than others.

# Bernstein's integral calculation

What if we want to test for smoothness
only the $(a, b)$ such that $|F(a, b)| < H$.

E.g. only the deep blue area in the previous picture.

The number of such pairs is actually easy to approximate (Bernstein, 2004).

$$S(H) = \#\{(a, b) \in \mathbb{R} \times \mathbb{R},\ |\mathrm{Res}(a - bx, f)| \leq H\},$$
$$\approx H^{2/\deg f} \int_{-\infty}^{\infty} \frac{\mathrm{d}t}{(f(t)^2)^{1/\deg f}}.$$

There are possible shortcuts to estimate this integral quickly.

# Impact on how we collect relations

Does this integral calculation say that we should look only at the value below $H$?

- The branches along the real roots are very thin. Their contribution is expected to be very small.
- Sieving in these ranges is expected to be difficult.
- More generally, sieving in this oddly-shaped region (even if we do not explore branches) is not particularly appealing.
- Note though that sieving is not the only way to collect relations.

Current software does not explore the $(a, b)$ pairs specifically in this region (but: more on this later).

However, this integral could be used as an estimator to compare polynomials. Cado-NFS does not do this.

# Recap

At this point we know how to:

- generate polynomial pairs with small coefficient bounds.
- use translation and (thanks to skewness) rotation to reduce the coefficients even further.
- use various estimators to assess the average (log-)norm over a given region.
- draw fancy pictures.

# Plan

The sieved range

Valleys and the starfish picture

Root properties

Forcing good $\alpha(f)$: the root optimization

Polynomial selection in Cado-NFS

# Notations

Notations: $(f, \cdot), \deg f = d, f = \sum f_i x^i, \phi(x) = a - bx$.

# An obvious lie

So far, we've relied on the heuristic:

> If $|F(a,b)| \approx H$, then $F(a,b)$ is smooth about as often as a random integer $\approx H$.

This is obviously wrong.

Examples:

- $3a^2 + b^2$ is never divisible by 5, 11, or 17 (for coprime $a, b$).
- OTOH, it is more often divisible by 7, 13, or 19.

This is related to ...

# An obvious lie

So far, we've relied on the heuristic:

> If $|F(a,b)| \approx H$, then $F(a,b)$ is smooth about as often as a random integer $\approx H$.

This is obviously wrong.

Examples:

- $3a^2 + b^2$ is never divisible by 5, 11, or 17 (for coprime $a, b$).
- OTOH, it is more often divisible by 7, 13, or 19.

This is related to . . . the number of roots modulo small primes.

# Roots modulo small primes

- $3a^2 + b^2$ is never divisible by 5, 11, or 17 (for coprime $a, b$) because $3x^2 + 1$ has no roots modulo these primes.
- OTOH, it is more often divisible by 7, 13, or 19, because $3x^2 + 1$ has two roots modulo these primes.

This is a general phenomenon.

Restatement: in the number field $\mathbb{Q}[x]/(3x^2 + 1)$ we have:

- no prime ideal of norm 5, 11, or 17.
- two prime ideals of norm 7, 13, or 19.

# Roots modulo small primes

- $3a^2 + b^2$ is never divisible by 5, 11, or 17 (for coprime $a, b$) because $3x^2 + 1$ has no roots modulo these primes.
- OTOH, it is more often divisible by 7, 13, or 19, because $3x^2 + 1$ has two roots modulo these primes.

This is a general phenomenon.

Restatement: in the number field $\mathbb{Q}[x]/(3x^2 + 1)$ we have:

- no prime ideal of norm 5, 11, or 17.
- two prime ideals of norm 7, 13, or 19.

Does this make a difference?

# Does this make a difference

## Answer #1: no, it does not make a difference

- On average, the number of prime ideals of norm below $B$ is $\approx \pi(B)$.
- This follows from the so-called prime ideal theorem (Landau) or from the (much stronger) Chebotarev theorem.

## Answer #2: in fact it does

- A deviation from the average is entirely possible for some range of primes.
- If this is the case for the small primes, the average contribution of small primes will be quite large.

# Root properties: example

Example from RSA-250:

$$\begin{aligned} f = \quad & 86130508464000x^6 \\ & - 66689953322631501408x^5 \\ & - 5273322103496633966198x^4 \\ & + 46262124564021437136744523465879x^3 \\ & - 31136272536132022651269074205506483 26x^2 \\ & - 17216144295387401200117600348293857920193 95x \\ & - 81583513076429048837733781438376984122961112000 \end{aligned}$$

Number of roots of $f$ modulo small primes:

| $p$ | $r_p$ | $p$ | $r_p$ | $p$ | $r_p$ | $p$ | $r_p$ | $p$ | $r_p$ |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 13 | 2 | 31 | 4 | 53 | 3 | 73 | 1 |
| 3 | 3 | 17 | 1 | 37 | 4 | 59 | 2 | 79 | 1 |
| 5 | 2 | 19 | 1 | 41 | 4 | 61 | 1 | 83 | 3 |
| 7 | 3 | 23 | 2 | 43 | 1 | 67 | 0 | 89 | 2 |
| 11 | 3 | 29 | 1 | 47 | 6 | 71 | 0 | 97 | 3 |

# Contribution of small primes

A random integer has $p$-valuation$> 0$ with probability $1/p$, etc.

$$\begin{aligned}\mathrm{E}\left[\nu_p(x),\ x \text{ random}\right] &= \frac{1}{p} + \frac{1}{p^2} + \cdots \\ &= \frac{1}{p-1}.\end{aligned}$$

We need to compute the same for $F(a, b)$ with $(a, b)$ coprime.

# Definition of $\alpha(f)$

- define a score function that reflects the deviation:

$$\alpha(f) = \sum_{p \text{ prime}} \left( \frac{1}{p-1} - \mathrm{E}[\nu_p(F(a,b))] \right) \log p.$$

- Note that $\frac{\log p}{p} \to 0$. The contribution of small primes to $\alpha(f)$ is dominant.
- $F(a,b) \approx X$ is smooth with probability comparable to that of a random number $\approx Xe^{\alpha}$.
- We want $\alpha(f)$ well $< 0$.
  Having $\alpha(f) < -k \log 2$ is equivalent to a $k$-bit smaller norm !

A good $f$ has small $\alpha$ (typically $\leq -7$).

# Estimation of $\mathrm{E}[\nu_p(F(a,b))]$.

$F(a,b)$ (for coprime $a,b$) is divisible by $p$ when...

- number-theoretician answer:
  when an ideal of power-of-$p$ norm divides $(a - b\alpha) \times J$.
- hurried practitioner answer, only valid if if $p \nmid f_d \operatorname{disc} f$:
  when $a/b$ is one of the roots of $f$ modulo $p$.

# Splitting into multiple cases

As far as only what happens mod $p$ is concerned, how many non-equivalent values of $(a, b)$ for coprime $(a, b)$ do we have?

- $p^2 - 1$ if we count everyone but $(0, 0)$.
- But $F(\lambda a, \lambda b) = \lambda^{\deg f} F(a, b)$ when $\lambda$ is invertible.
  So we can divide by $p - 1$.
- There are $p + 1$ different classes.
  In mathematical terms: this is the projective line $\mathbb{P}^1(\mathbb{Z}/p\mathbb{Z})$.

# Estimation of $\mathrm{E}[\nu_p(F(a,b))]$.

For the primes for which $p \nmid f_d \operatorname{disc} f$, the contribution to $\alpha(f)$ is:

$$r_p \cdot \frac{1}{p+1} \cdot \left(1 + \frac{1}{p} + \frac{1}{p^2} + \cdots\right)$$
$$= r_p \cdot \frac{1}{p+1} \cdot \frac{p}{p-1}.$$

Problem: small primes are most likely to divide $p \nmid f_d \operatorname{disc} f$.
It is not reasonable to ignore this.

It takes some extra complications to compute the contributions of all primes, but this is very much doable.

# $\alpha(f)$ joins the estimation toolbox

## $\alpha(f)$ is easy to compute

It is easy to compute (a reasonable approximation of) $\alpha(f)$ by restricting to (say) primes $< 2000$.
It makes sense to range polynomials according to the value of

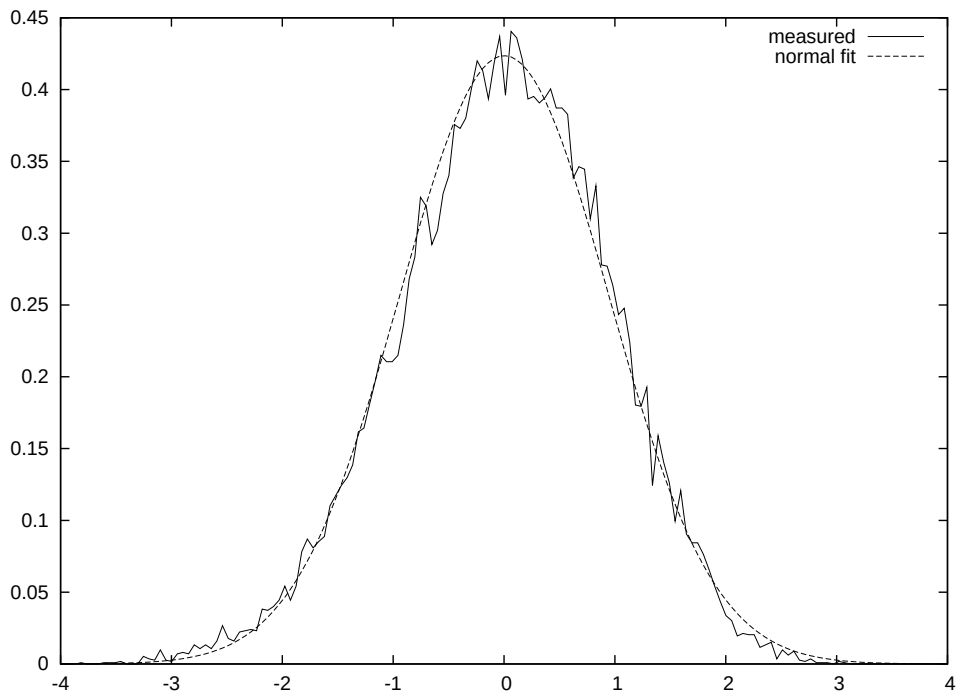
$$\log(\text{average } |F(a,b)|) + \alpha(f).$$

(`sage` code in Cado-NFS), (`C` code in Cado-NFS)

# Distribution of $\alpha(f)$

As $f$ varies over a large number of polynomials, $\alpha(f)$ follows a roughly normal distribution.

- It is not hard to estimate $\mu$ and $\sigma$ empirically.
- It may even be doable in theory, but I'm not sure.



# Examples of $\alpha(f)$

Examples:

- RSA-768, $\alpha(f) = -7.3$.
- RSA-240, $\alpha(f) = -8.48$.
- RSA-250, $\alpha(f) = -8.88$.

**$p \mid f_d$ is an important case!**

What happens for $(a : b) \sim (1 : 0) \mod p$ is as important as the other cases.
This is the reason why we often force $f_d$ to have many small prime factors: it forces extra roots.

# Predicting $\alpha(f)$

If we examine (say) $N = 2^{30}$ polynomials $f$, and we intend to keep the best (smallest) $\alpha(f)$, what should we hope for?

Experimentally, we can determine $\mu$ and $\sigma$ for the normal fit of the distribution of $\alpha(f)$. WLOG, assume $\mu = 0$ and $\sigma = 1$.

Our problem is the determination of the max value of $N$ random picks of a normal-distributed random variable $X$.

This is a well-known problem called order statistics.

- First-order $\sqrt{2 \log N}$.
- More terms are known, as well as empirical "corrections".
- This can be use to ponder whether "better" polynomials are still to be expected or not.

# Plan

The sieved range

Valleys and the starfish picture

Root properties

Forcing good $\alpha(f)$: the root optimization

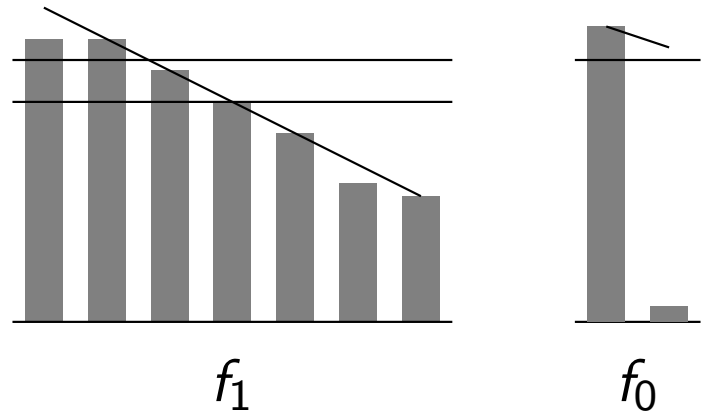
Polynomial selection in Cado-NFS

# Notations

Notations: $(f_0, f_1), \deg f_1 = d$, coefficients of $f_i$ unnamed.

# Upsides of skewness

When we obtain a skewed polynomial pair, it is rare that the lower-degree coefficients play a role.




In most cases, the rotation $(f_0, f_1) \to (f_0, f_1 + \lambda f_0)$ will not change the coefficient sizes by much (if at all).

- What happens mod $p$ is changed completely. We may hope for improvements.
- We used rotation for size optimization already. Here we want to do it on a more limited scale.

# Root properties of $f_1 + \lambda f_0$

The stupid way to do root optimization

- Iterate over all possible $\lambda$.
- For each, compute $\alpha(f_1 + \lambda f_0)$.
- Keep best.

Main problem: root finding mod many $p$ for each $\lambda$.

Is it possible to compute $\alpha(f_1 + \lambda f_0)$ quickly? Not really.

# Root optimization – the root sieve

## The root sieve

- Iterate over all $p^k < B$ below some fixed bound.
- Iterate over all possible root values $i \bmod p^k$.
- For all $\lambda$ such that $f_1(i) + \lambda f_0(i) \equiv 0 \mod p^k$, add to $T[\lambda]$ the contribution of having a root (typically $\frac{p \log p}{p^2-1}$).
- Keep the ones with largest recorded contribution.
- Beware: this costs $\widetilde{O}(B \cdot (B + \lambda_{\max}))$

It is not entirely straightforward.

- Some issues with finding the contribution of multiple roots correctly.
- Can gain a factor of two by looking at powers more precisely.

# Plan

The sieved range

Valleys and the starfish picture

Root properties

Forcing good $\alpha(f)$: the root optimization

Polynomial selection in Cado-NFS

# Polynomial selection in Cado-NFS

The degree is fixed beforehand. First phase:

- loop through all $a_d$ in some range `[admin,admax]`.
  - It makes sense to look at only the multiples of some prescribed number, hence `incr`.
  - This can (MUST) be distributed over many machines! The workload is divided into many pieces according to `adrange`.
- run algorithms to find good polynomials based on $a_d$.
  The Cado-NFS implementation of Kleinjung's 2008 algorithm has a few arcane parameters (`P`, `nq`).
- size-optimize.
  - quantify the amount of effort we put into it: `sopteffort`.
- collect all these size-optimized polynomials in a central place.
  Limited-size priority queue of best polynomials: `keep`.

# Polynomial selection in Cado-NFS (cont'd)

Second phase:

- For each of the best pairs of polynomials that are still in the priority queue, run root optimization.
- That can very well mean a sizable number of root optimization tasks, so distribution is again a good idea.
- Not as many parameters.
- Keep the very few best ones ($\approx 10$).
- For large-scale computations, run sieving tests with each.