Safe Programming in Finite Precision: Controlling Errors and Information Leaks

Ivan Gazeau

under the supervision of Dale Miller and Catuscia Palamidessi

October 14, 2013

Introduction

We study the influence of errors due to the finite representation of numbers.
We mix information from

- automated static analysis of program
- statements made in exact semantics

Plan of the talk:

1. *Robustness*: background and definitions of robustness.
2. *Global analysis*: provides a new method that mixes standard static analysis and mathematical statements for some hard cases
3. *Differential privacy*: how errors perturb the generations of probabilistic noises.

Robustness

Internal representation of real number

Two main finite representations are used:

- Fixed point numbers: the exponent is fixed statically.

$$0100010.10010$$

  - Representable numbers belong to a small interval
  - Good control on rounding errors

- Floating point numbers: the exponent is set dynamically.

$$1.00101010 \times 2^{-101}$$

  - Larger range of representable numbers
  - Rounding errors are less predictable:
    $a + (b + c)$ and $(a + b) + c$ evaluations can give different results.

We look for a generic description of errors independent from the representation.

Static program analysis

Static analysis is an analysis that:

- takes a source code as input but does not execute it,
- outputs some properties about this code.

Example

Abstract interpretation

$x \in [0,1] \pm 0.01$, $y \in [0,2] \pm 0.001$

x = 3 $*$ y $-$ x;

$x \in [-1,6] \pm 0.013$, $y \in [0,2] \pm 0.001$

y = x $*$ y;

$x \in [-1,6] \pm 0.013$, $y \in [-2,12] \pm 0.0267$

There are also Hoare Triple methods where a proof tree is built.

We look for a definition of robustness general enough to be derived from any static analysis.

To measure errors

Static analyzers keep track of detailed information to provide accurate results.

We just keep essential information from the analysis:

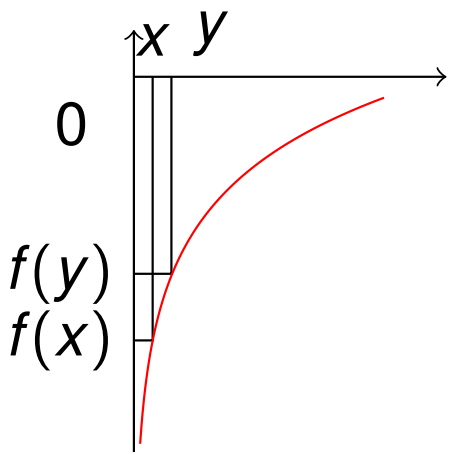
- all values of the program are seen as points in a metric space $(\mathbb{R}^m, d_m)$,
- we just consider the distance corresponding to the maximal error.

We have to distinguish between two kinds of errors:

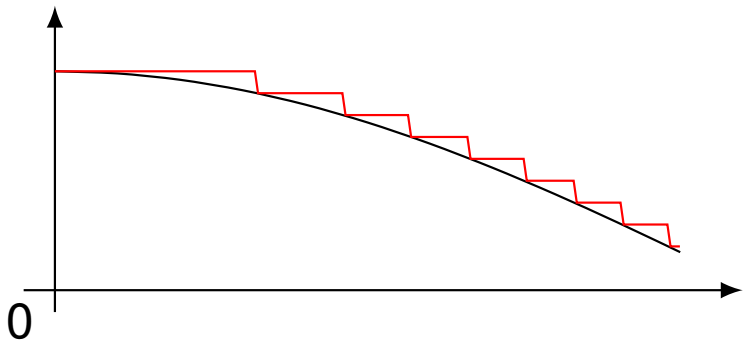
- Errors that appear during the computation due to successive rounding.
- Errors that propagate from previous computations.

Input/output errors

The slope of the function:
it acts as an expansion factor



Intern approximations:
small local gaps are allowed



Definition (The property $P(k, \epsilon)$)

Let $(\mathbb{R}^m, d_m)$ and $(\mathbb{R}^n, d_n)$ two metric spaces. Let $f : \mathbb{R}^m \to \mathbb{R}^n$, $k, \epsilon \in \mathbb{R}^+ \cup \{0\}$, we say that f is $P(k, \epsilon)$ if

$$\forall x, y \in \mathbb{R}^m, d_n(f(x), f(y)) \leq k \cdot d_m(x, y) + \epsilon$$

Internal errors

They can only be defined from both the exact and the finite-precision semantics.

Notations

f : program code
$[\![\mathrm{f}]\!]$: exact semantics of f
$[\![\mathrm{f}]\!]'$: finite-precision semantics of f

Definition ((k, ϵ)-Closeness property)

Let $(\mathbb{R}^m, d_m)$ and $(\mathbb{R}^n, d_n)$ be metric spaces. Let f and g be two functions from $\mathbb{R}^m$ to $\mathbb{R}^n$ and let $k, \epsilon \in \mathbb{R}^+$. We say that g is (k, ϵ)-close to f if the following holds:

$$\forall x, y \in \mathbb{R}^m, d_n(f(x), g(y)) \leq k d_m(x, y) + \epsilon$$

Global analysis

Presentation of the problem

Problem of non locality: although the whole program is robust, its components may not be (due to discontinuous conditional branchings). Due to non-locality, standard compositional methods do not apply.

We propose a method:

- based on a global pattern
- that uses properties of the program in exact computation
- that uses a static analysis of the program that assumes the control flow is correct.

We bound the distance between the exact function and its floating-point approximation.

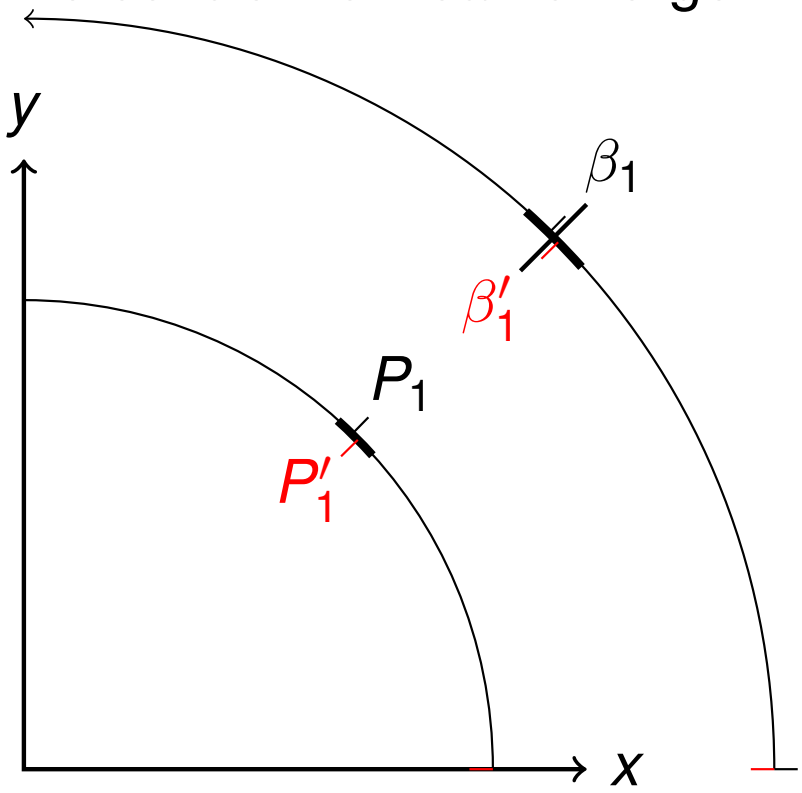
Example: the CORDIC algorithm for computing cosine

Variables of the program:
α: the angle given in input
β: an angle initialized to 0
(x, y): the coordinates of a point P in a unit circle, initialized to $(1, 0)$.
The algorithm principle:

- Invariant: the angle of P in polar coordinates is β
- Depending on whether α is greater than β, P is rotated left or right
- This dichotomy is made a fixed number of iterations

Problem of finite precision

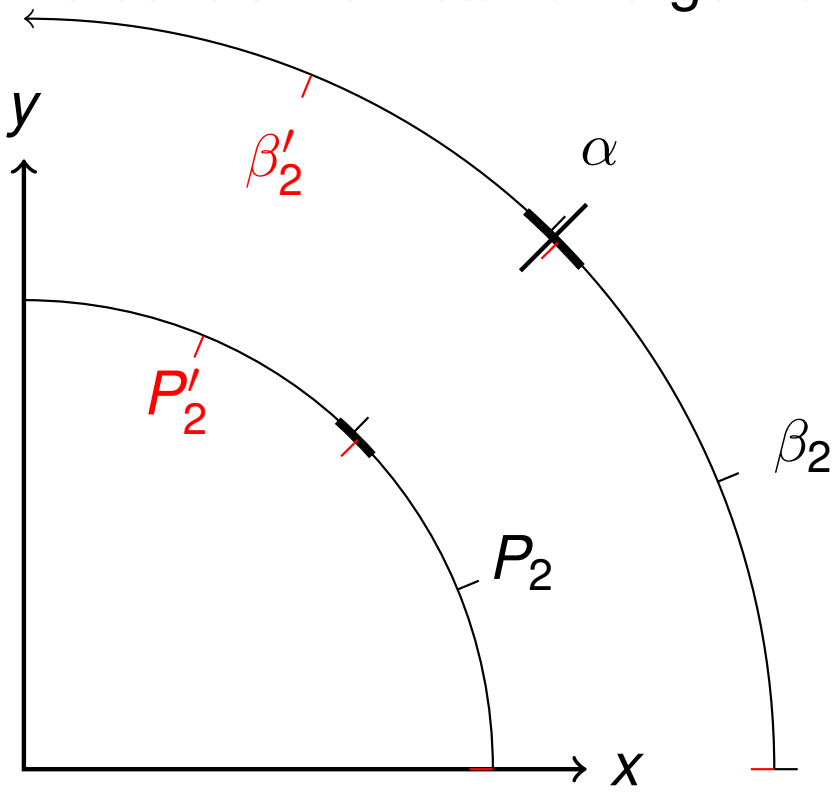
The control flow can diverge from the expected one.



- Error after 2 steps is unacceptable.
- Error after n steps can be small.

Problem of finite precision

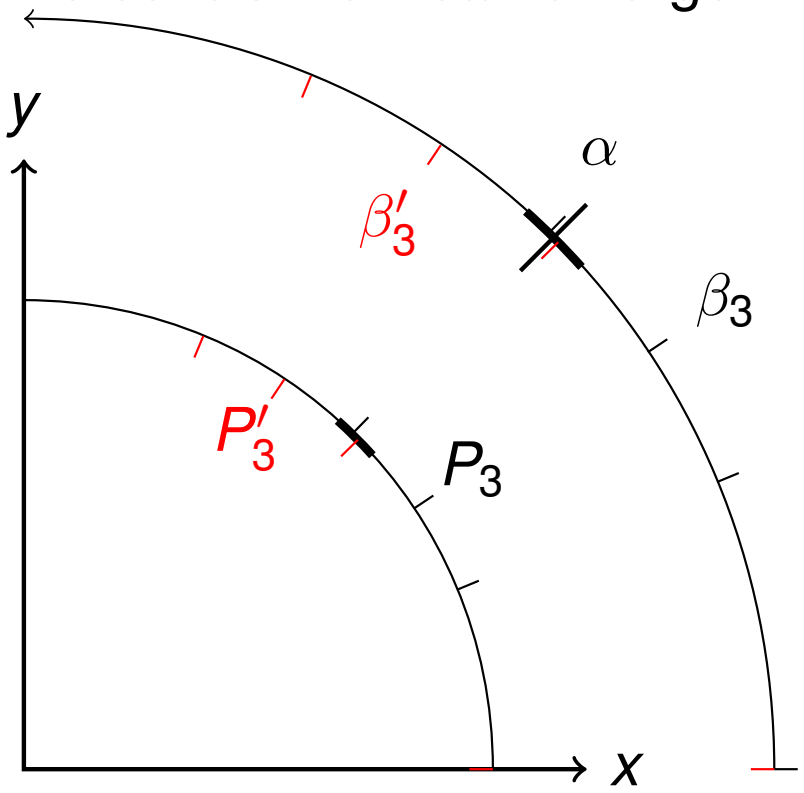
The control flow can diverge from the expected one.



- Error after 2 steps is unacceptable.
- Error after *n* steps can be small.

Problem of finite precision

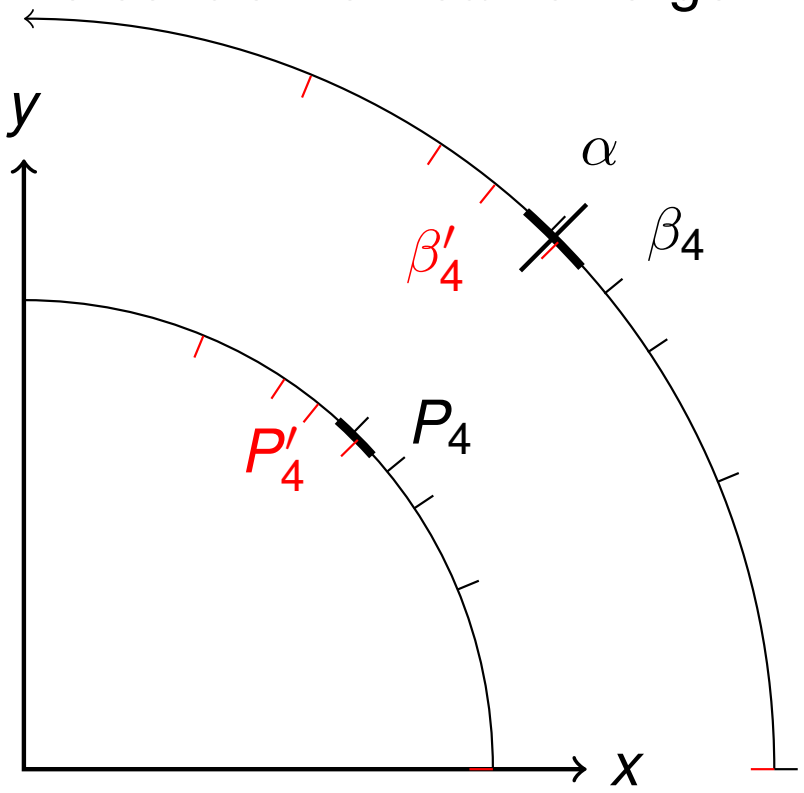
The control flow can diverge from the expected one.



- Error after 2 steps is unacceptable.
- Error after *n* steps can be small.

Problem of finite precision

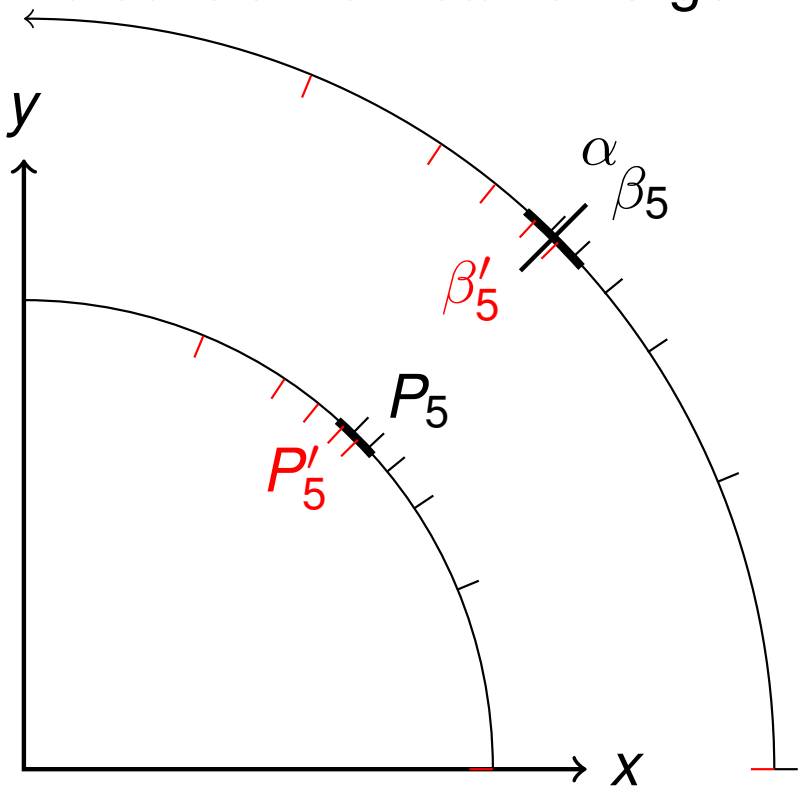
The control flow can diverge from the expected one.



- Error after 2 steps is unacceptable.
- Error after *n* steps can be small.

Problem of finite precision

The control flow can diverge from the expected one.

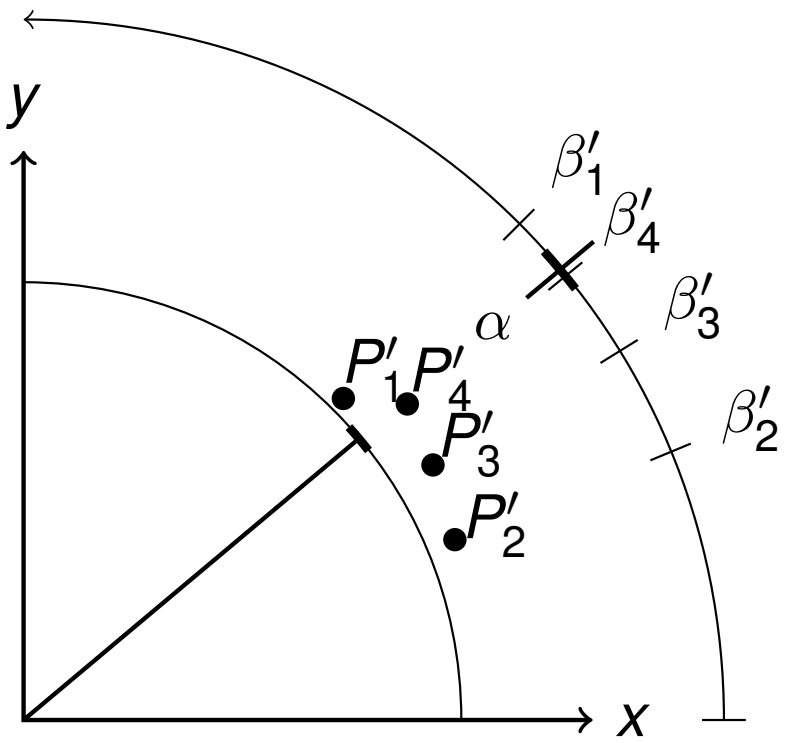


- Error after 2 steps is unacceptable.
- Error after *n* steps can be small.

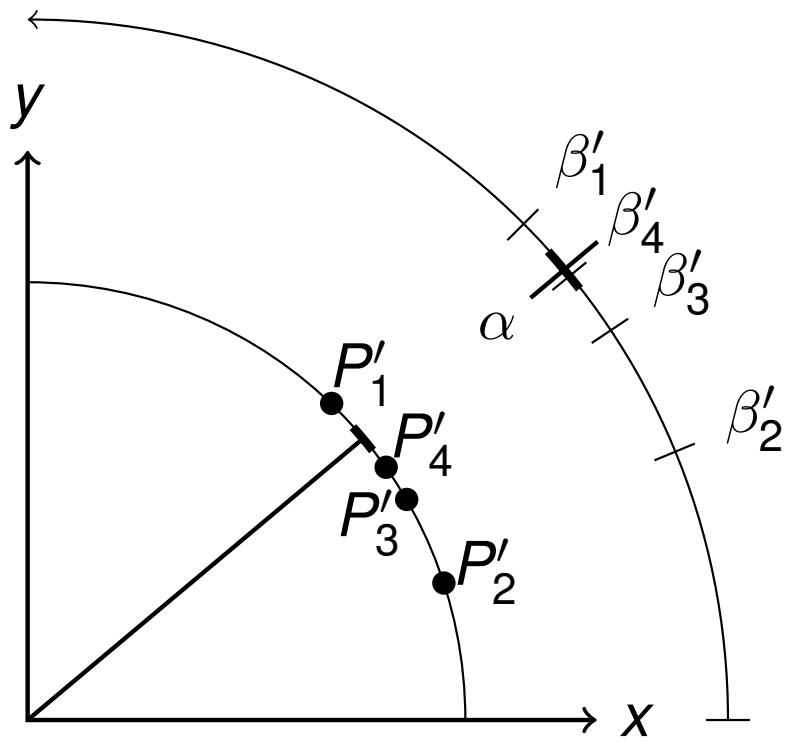
Adapt the proof in exact semantics

The exact proof of confluence is straightforward
But the assumptions are no longer valid in finite precision.

Invariants



Termination properties



Our approach

- Control flow errors are hard to analyze with standard static analyzers due to the discontinuities.
- Correctness proofs of algorithms exist but are valid only for the exact semantics.

To quantify the errors, we make use of:

- A static analysis that assumes there is no control flow error.
- Some properties about the exact semantics.

The model

Abstract rewrite systems provide a good framework for confluence.
We consider the program is rewriting a given term:

- Conditional branches correspond to the non-deterministic choices of the rewrite system.
- The program stops once the term is in a final form.

To provide such an understanding of the code, we match it against a pattern.

First step: decompose the code

The pattern asks for read and write access to variables.

```
foo(m){
    (n,i) = Init();
    while(! S(m,n,i)){
        (i,n) = C(m,n,i);
            m = R(i,m); }
    return m; }
```

The interpretation as a rewrite system:

- C(m,n,i) is the scheduler.
- R(i,m) allows us to define rewrite rules: $m \xrightarrow{i} R(i, m)$.

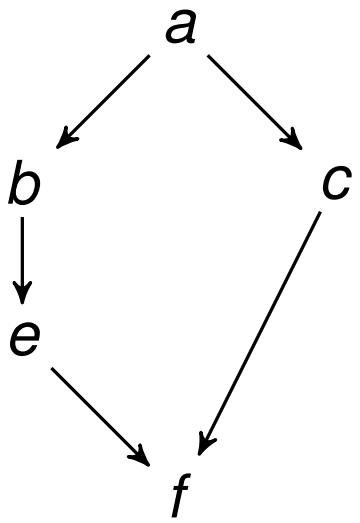
Constraint the rewrite system

We want this rewrite system to be:

Locally confluent : two different rewritings of a term can be continued to reach a common term.

Terminating : no infinite chain $a \to b \to c \ldots ...$

From Newman's lemma, these two properties imply all terms have a unique normal form.



If we can prove that, when the program stops, the term is irreducible, then any rewrite choice leads to the same result.

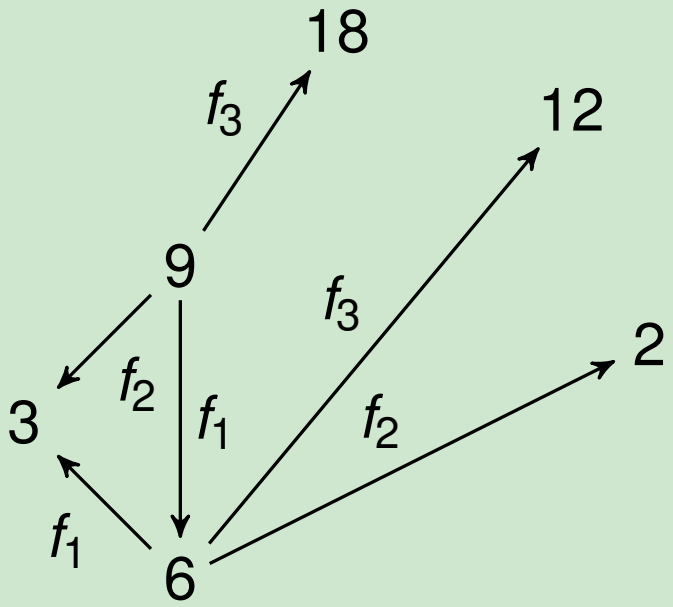
To provide a structure to our rewrite system

We consider a function $h : \mathbb{R}^m \to \mathbb{R}$.
We define $a \xrightarrow{>} b$ iff $a \to b$ and $h(a) > h(b)$.
We denote by $\bar{\mathbb{R}^m}$ the set of normal forms.

Example

$f_1(x) = x - 3$, $f_2(x) = x/3$ and $f_3(x) = 2x$.

$h(x) = (x-5)^2$



The conditions for robustness

1. $\overset{\geq}{\rightarrow}$ is locally confluent.
2. The rewriting system $\overset{\geq}{\rightarrow}$ is terminating.
3. The following property holds.

$$\forall a, b \in \mathbb{R}^m, a \rightarrow b \implies \exists c \in \mathbb{R}^m a \overset{\geq}{\rightarrow}{}^* c {}^* \overset{\leq}{\leftarrow} b$$

4. The function $[\![foo]\!]$ is $P(k_e, \epsilon_e)$.
5. In the exact semantics, when the stopping condition is reached, the final value m is such that $m \overset{\geq}{\rightarrow}{}^* z$ implies $d(m, z) \leq \epsilon_s$.
6. We require the closeness property between $[\![foo]\!]'$ and $[\![\mathrm{foo}]\!]_p$, the exact function that corresponds to the same control flow

$$\forall x, y \in \mathbb{R}^m,\ d([\![\mathrm{foo}]\!]_p(x), [\![foo]\!]'(y)) \leq k_f d(x, y) + \epsilon_f.$$

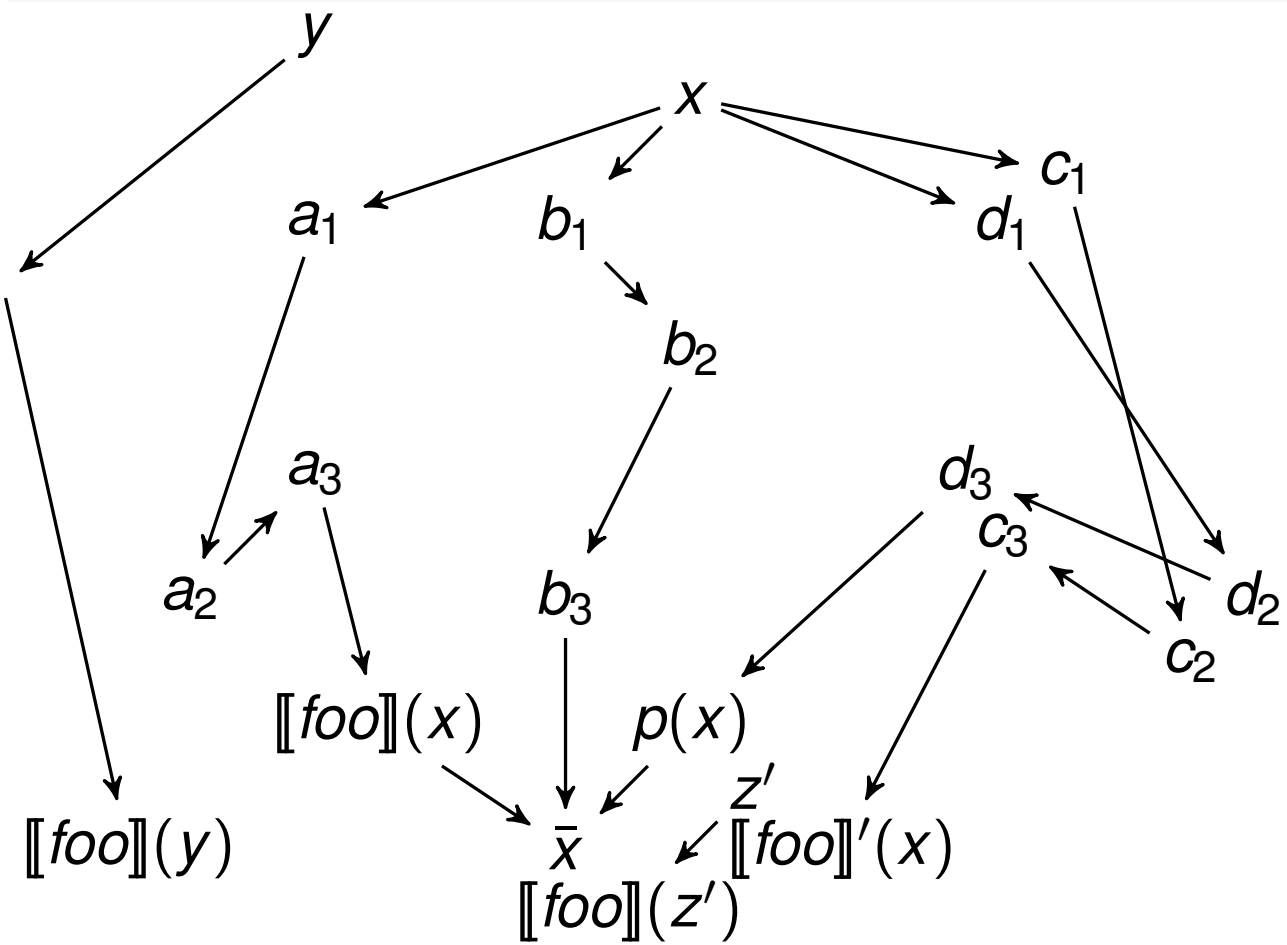
7. In the finite-precision semantics, when the stopping condition is reached, the final value m' is such that

$$\exists z' \in \mathbb{R}^{\bar{m}},\ d(m', z') \leq \epsilon'_s$$

The theorem

Theorem

If all conditions are satisfied, $[\![foo]\!]$ and $[\![foo]\!]'$ are (k_e, ϵ)-close (with $\epsilon = k_e(\epsilon_f + \epsilon_s) + \epsilon_e + 2\epsilon_s + \epsilon'_s$).



Conclusion about global analysis

- A method to analyze finite-precision semantics that relies on exact semantics properties.
- The method deals with local discontinuities.
- A proof through rewriting techniques.
- The pattern can work for very different programs (tested with CORDIC and Dijkstra's algorithms).

Differential privacy

General motivations

How to control computational errors in a security setting?

Example

h : high (confidential) variable when $h \in [v_1, v_2]$.
l : low (public) variable.
If $f(h) > 0$ then $l = 0$ else $l = 1$.

If, in the exact semantics $f([v_1, v_2]) \subseteq \mathbb{R}^+$: no leakage.
But if $f'([v_1, v_2]) \cap \mathbb{R}^- \neq \emptyset$: possible leakage.

What is differential privacy?

Purpose:

- Release global properties of the databases (like correlations, averages).
- Protect personal information at an individual level.

The analyst is free to ask any query to the database.

Requirements

Differential privacy requires that the information obtained from a database are almost identical,

- whether or not an individual participates to a database,
- whatever is the query.

Differential privacy

We write $D_1 \sim D_2$ if D_1 and D_2 differ in exactly one row.

Definition (ϵ-differential privacy)

A randomized mechanism $\mathcal{A} : \mathcal{D} \to \mathbb{R}^m$ is ϵ-differentially private if for all databases D_1 and D_2 in $\mathcal{D}$ with $D_1 \sim D_2$, and all $S \in \mathcal{S}$ (the Lebesgue σ-algebra), we have :

$$P[\mathcal{A}(D_1) \in S] \leq e^{\epsilon} P[\mathcal{A}(D_2) \in S]$$

An equivalent formulation:

$$e^{-\epsilon} P[\mathcal{A}(D_2) \in S] \leq P[\mathcal{A}(D_1) \in S] \leq e^{\epsilon} P[\mathcal{A}(D_2) \in S]$$

The analyst owns a privacy budget ϵ_t.
He can ask ϵ_i-private queries while $\sum_i \epsilon_i \leq \epsilon_t$.

Standard technique

Oblivious mechanism: the returned answer depends only on the true result.
In case of real valued results, it consists in adding a random variable:

Mechanism (standard)

$$\mathcal{A}_0(D) = f(D) + X$$

The suitable scale for X depends on the sensitivity:

Definition (sensitivity)

The sensitivity Δ_f of a function $f : \mathcal{D} \to \mathbb{R}^m$ is

$$\Delta_f = \sup_{D_1, D_2 \in \mathcal{D}, D_1 \sim D_2} d(f(D_1), f(D_2))$$

Example

Name	Age	Incomes
Alice	16	14,500
Bob	18	36,000
Charly	27	22,000
...	...	...

Analyst's queries:

a. " Number of people under the age of n? "

b. " Total incomes of people under the age of n? "

n	Query	True answer	Sensitivity	Typical returned answer
65	a	1500	1	1504.82
65	b	45,700,453	100,000	47,834,345
16	a	1	1	-0.23
16	b	14,500	100,000	158,345

- For general queries, results are accurate.
- For queries about few individuals, results do not leak information.

Why this mechanism is correct in exact semantics

Often, the added noise is a Laplace noise with scale parameter $\frac{\Delta_f}{\epsilon}$.

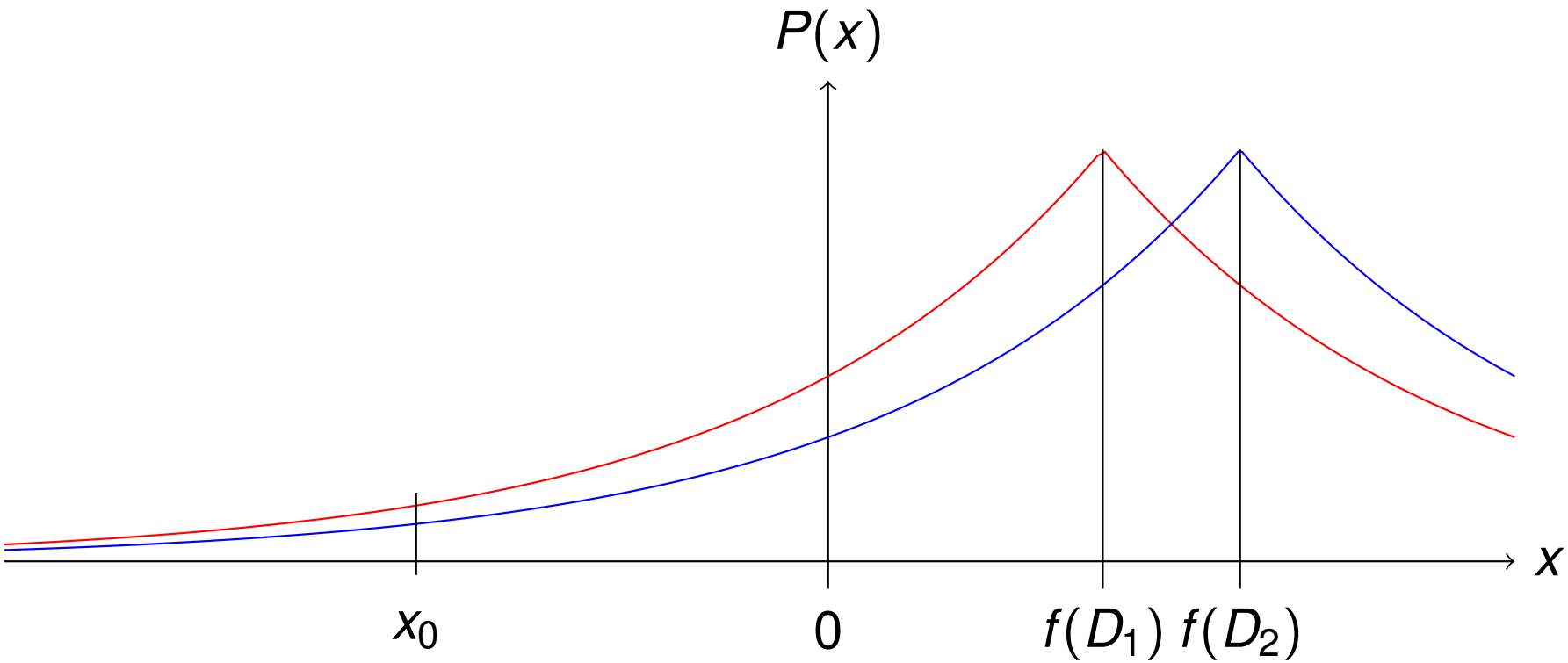


Figure: The ratio between the two distributions is bounded by $4/3$

This distribution is optimal since the ratio is exactly $4/3$ for $x < f(D_1)$ and $x > f(D_2)$.

Standard technique to generate random variable

1. Draw one or several values $u_1, \ldots, u_q$ uniformly distributed in $[0, 1]$.
2. Compute $n(u_1, \ldots, u_q)$.

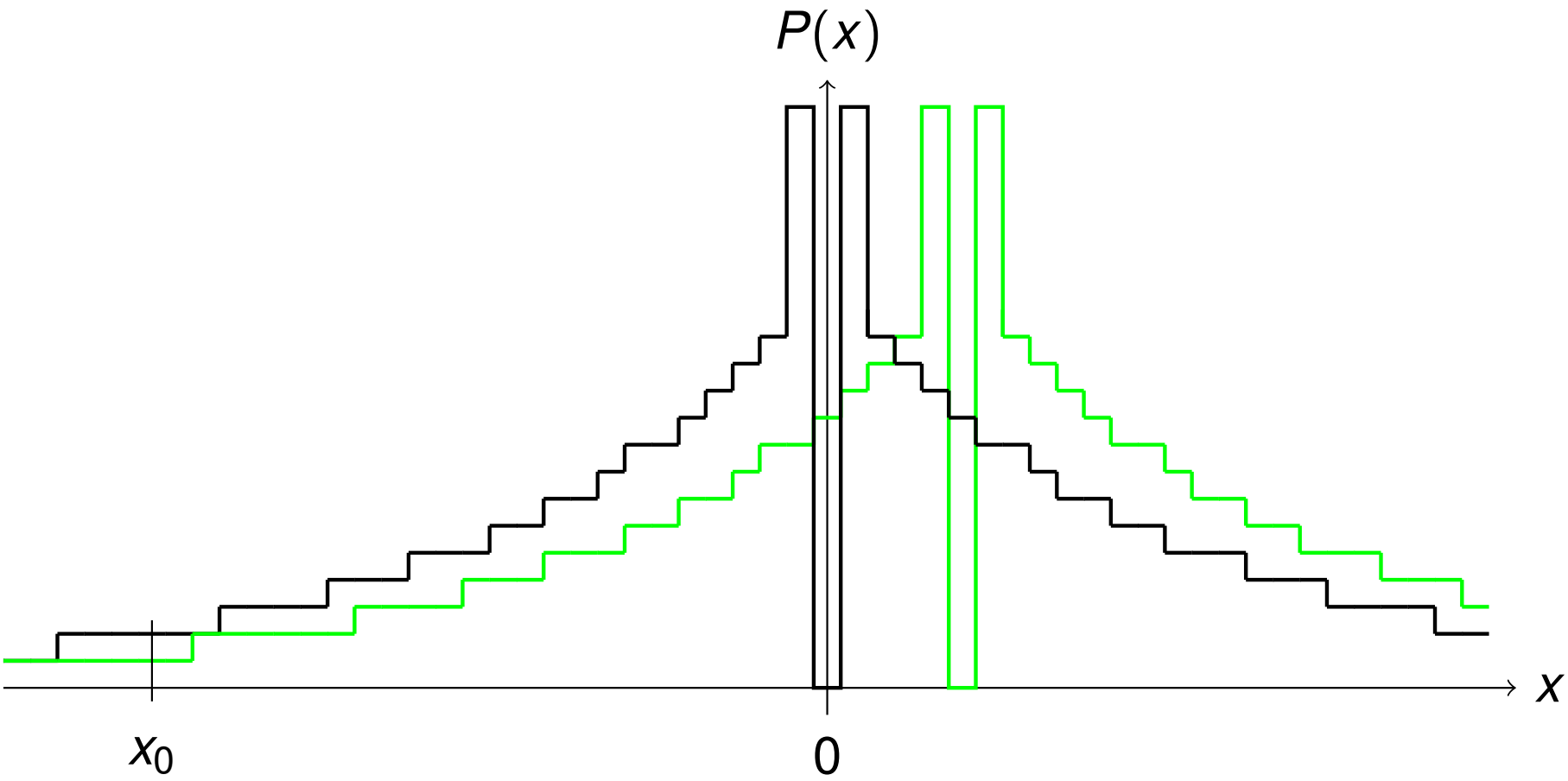
Example (Generation of a Laplace noise in $\mathbb{R}$)

$$n(u) = \frac{\Delta_f}{\epsilon} \mathrm{sgn}(u - 1/2) \ln\left(1 - 2|u - 1/2|\right)$$

Problems of the pseudo-distribution in finite precision

Finite set of inputs: any distribution is a step function.

- Rounding process and errors may “avoid” to return some values.
- Very low probabilities for large value are badly generated.



Our contribution

Implemented algorithms use finite representation of numbers.

Problem

- A direct implementation breaks privacy: need to provide safeguards.
- Once protections are added, how to measure the leakage due to the implementation?

Our assumptions on the exact mechanism are weak:

- the domain of the answer belongs to any $\mathbb{R}^m$ space
- the distribution of the added noise is not fixed

About the architecture:

- any finite representation of numbers
- any algorithm to implement the noise generation

Initial error from the uniform random value

Standard technique: take an integer between 0 and $N-1$ and divide by N.
The distribution is discrete while it should be continuous.

A model for this error

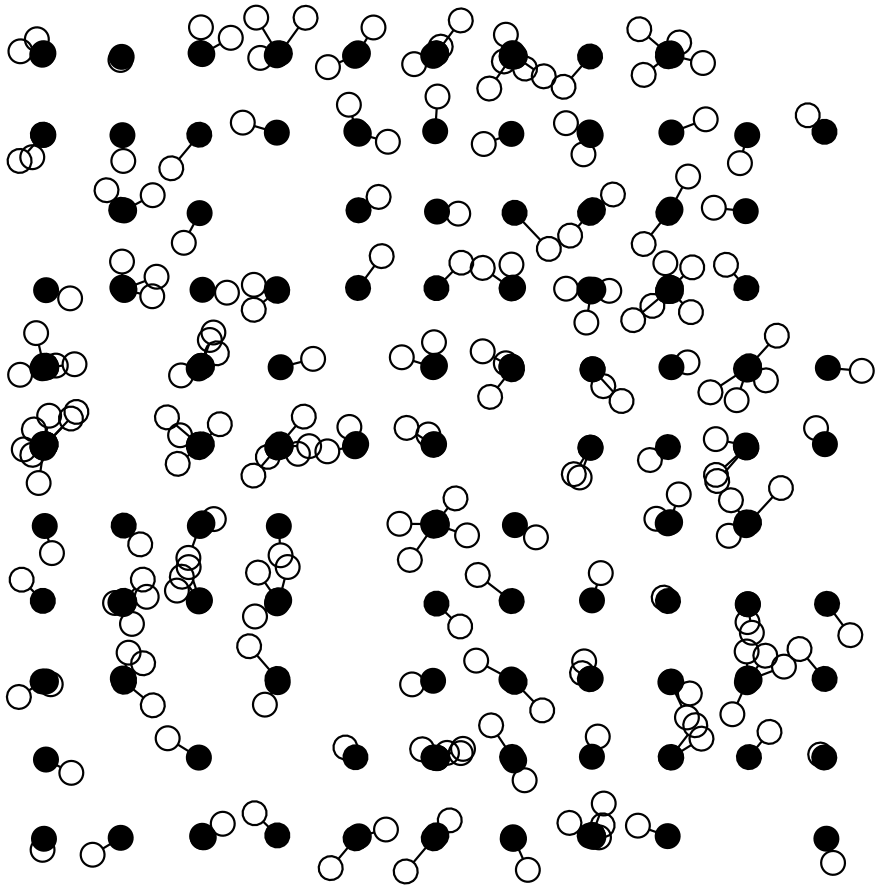
We can consider this process as drawing a perfect uniform random variable u then adding a perturbation mechanism like

$$u' = n_0(u) = \mathrm{round}(u)$$

Modeling the initial error

We denote by δ_0 the maximal error such that:

$$\forall u \in [0,1]^q, d(n_0(u) - u) \leq \delta_0 \tag{1}$$



Rendering a specific noise

Compute a function n that takes a uniform random value as an argument.

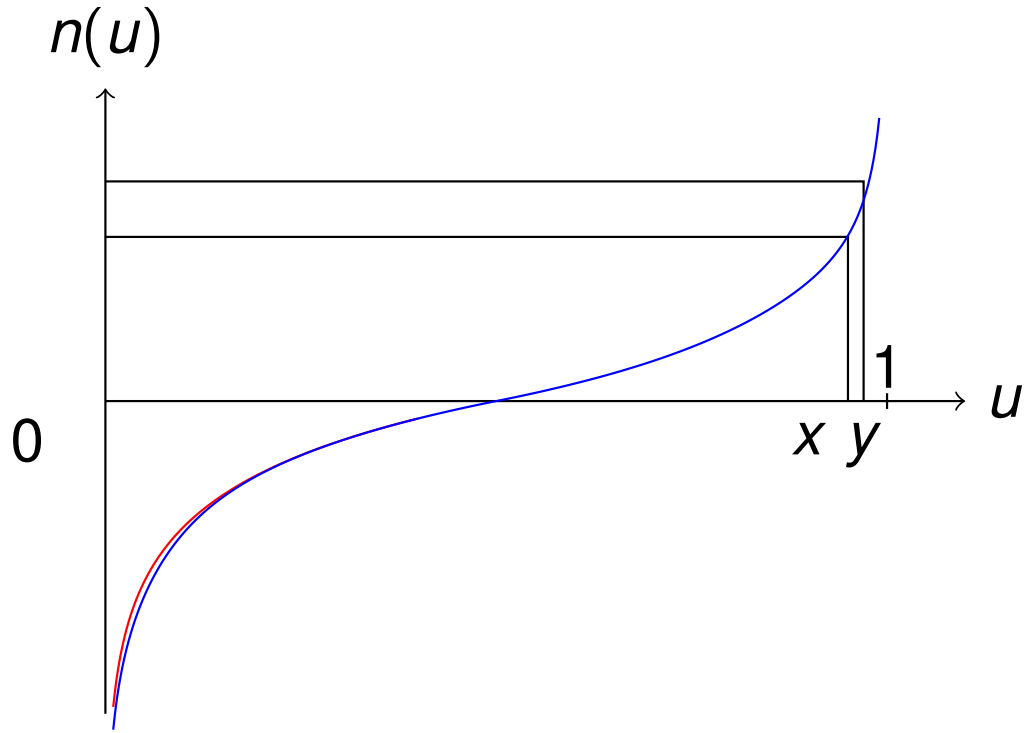
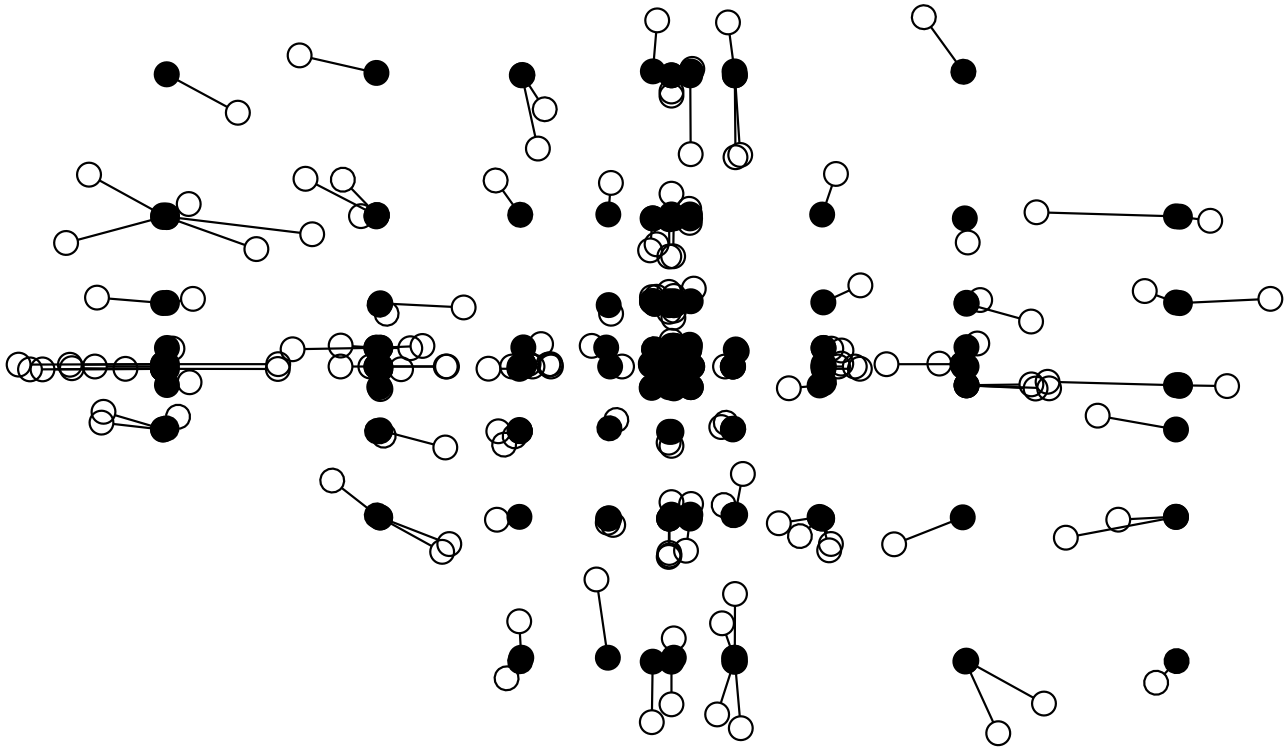


Figure: Generation of a Laplace distribution

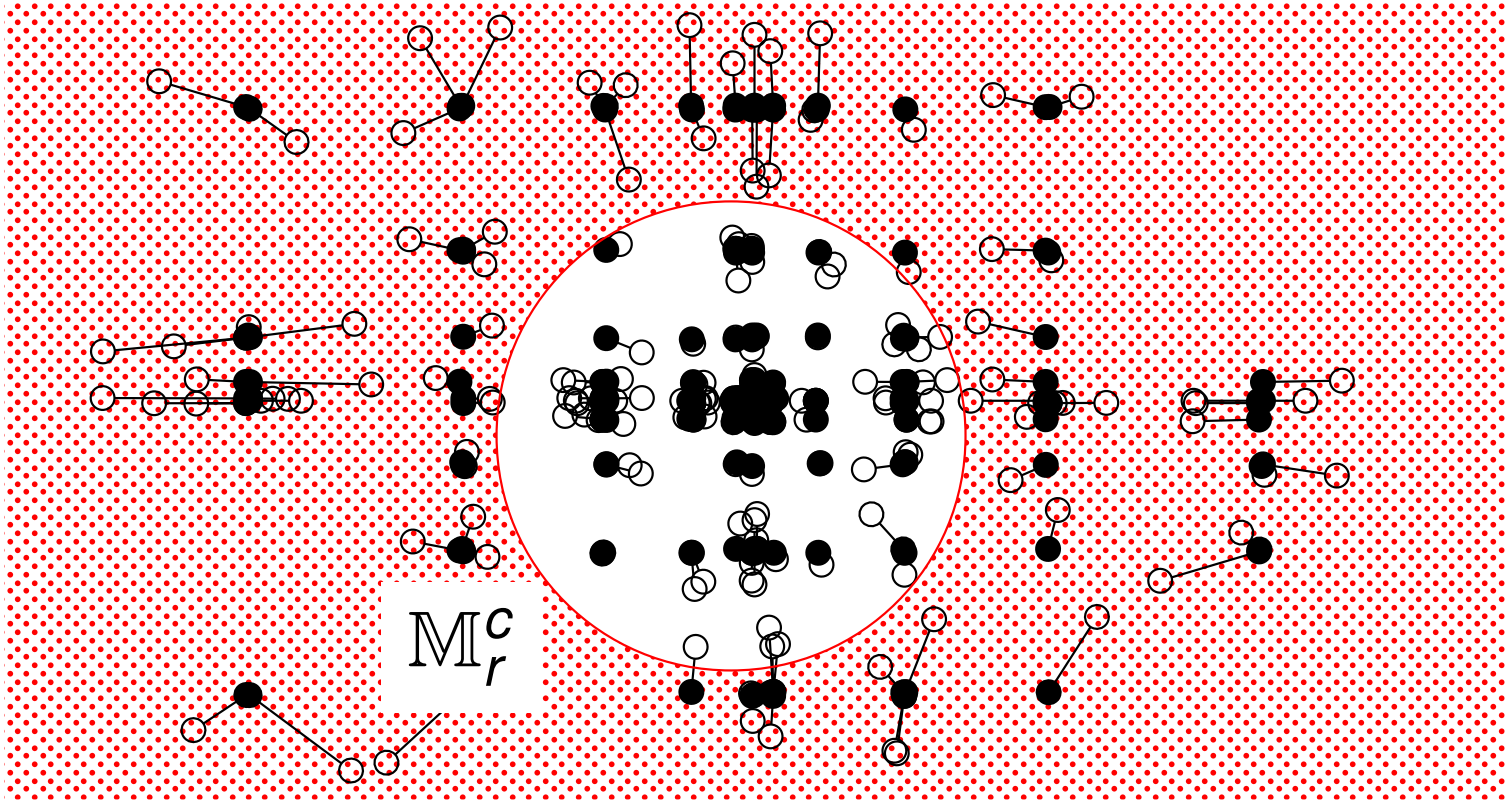
This computation multiplies the initial error (and may induce additional error).

Large values are unsafe



For extremal values of u, the slope of n is too high:
the multiplication factor is too big to provide safety property.

Truncating the result



Mechanism (truncated)

$$\mathcal{A}(D) = \begin{cases} f(D) + X & \text{if } f(D) + X \in \mathbb{M}_r \\ \infty & \text{otherwise} \end{cases}$$

Computational assumptions

Condition

We require n and n′ to be (k, δ_n)-close on a set U_r such that $\mathbb{M}_r \subseteq n(U_r)$

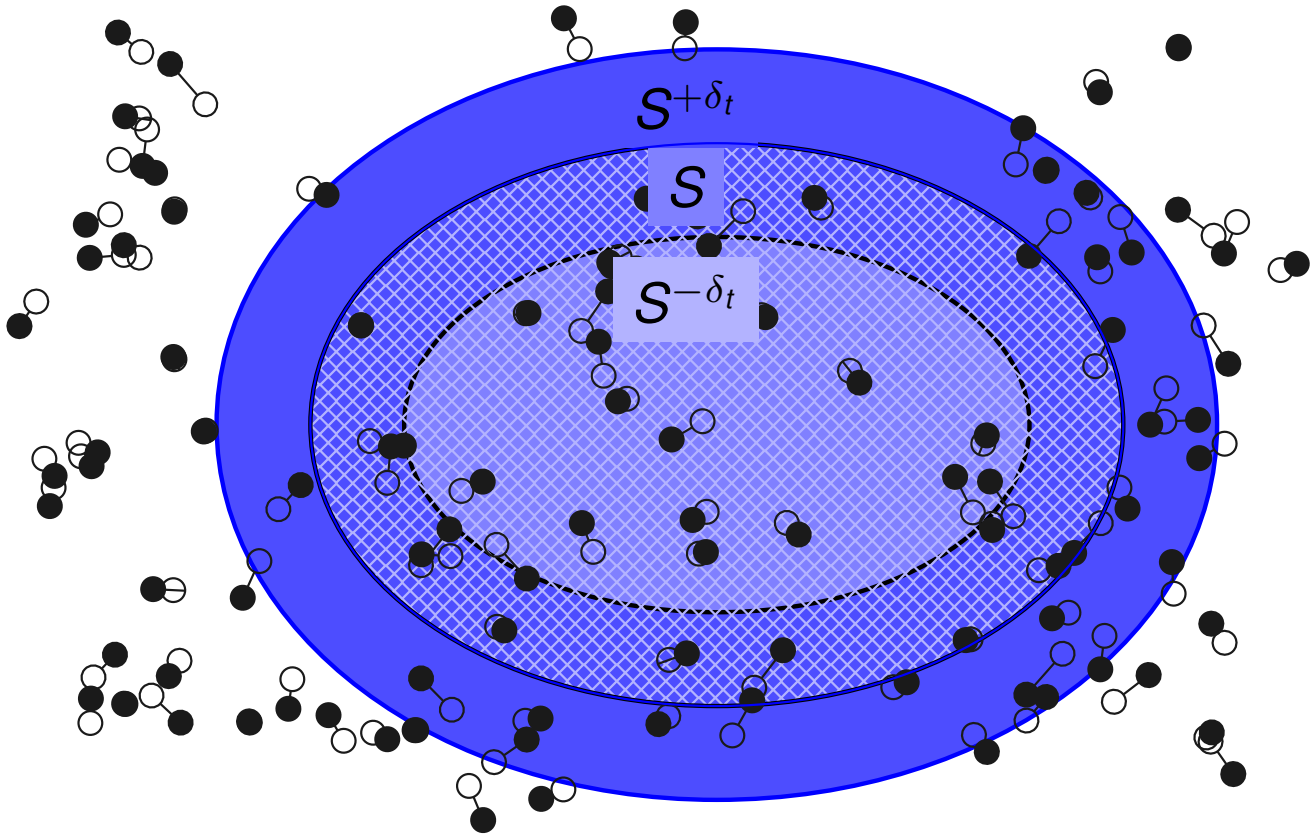
Total deviation

The deviation cannot exceed $\delta_t = k\delta_0 + \delta_n$

This error has to be translated in term of probability.

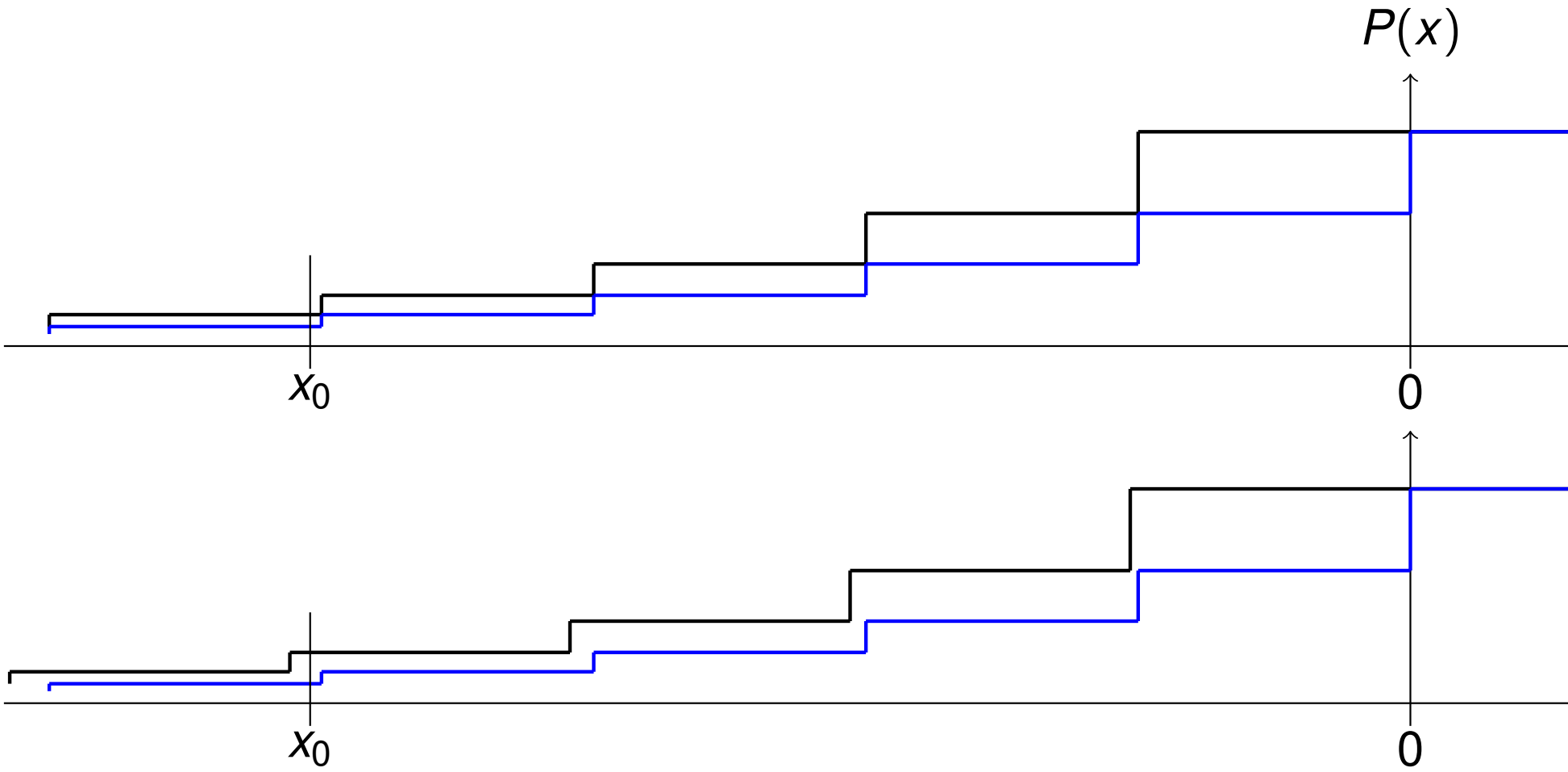
How to bound probability from deviation error?

$$P(X \in S^{-\delta_t}) \leq P(X' \in S) \leq P(X \in S^{+\delta_t})$$



Constraining the set of eligible distributions

Some specific distributions have weaker robustness to rounding errors.



Strengthening the differential privacy property

Because of deviations, we need to grant properties on a larger domain than the theoretical one.
Original differential privacy:

$$\forall S \in \mathcal{S}, P[\mathcal{A}(f(D_1) + X) \in S] \leq e^{\epsilon} P[f(D_2) + X \in S]$$

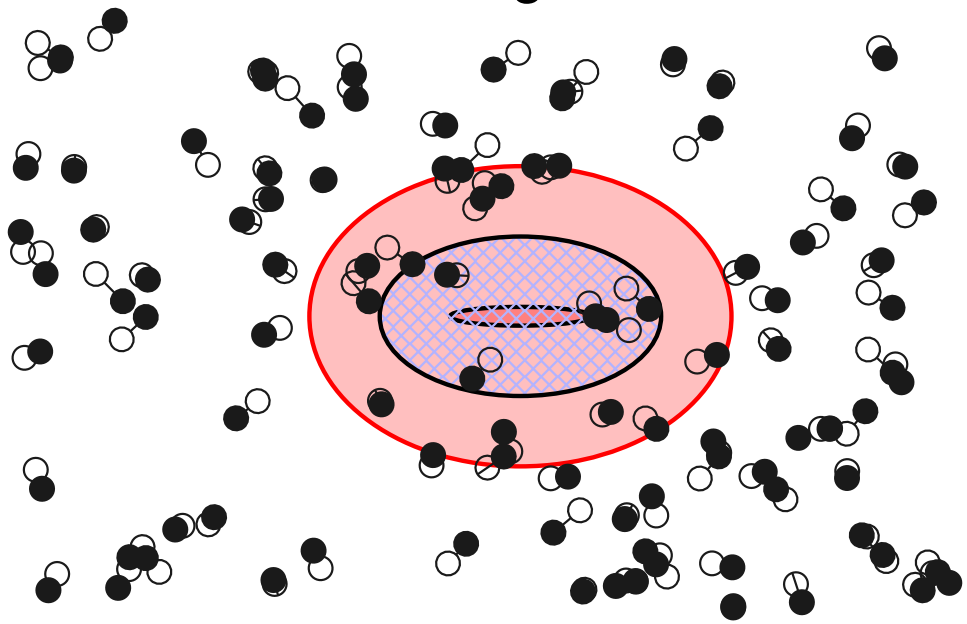
Strengthen formula:

$$\forall S \in \mathcal{S}, r_1, r_2 \in \mathbb{R}^m, P[r_1 + X \in S] \leq e^{\epsilon \frac{d(r_1, r_2)}{\Delta_{f'}}} P[r_2 + X \in S]$$

In case of Laplace distribution, this condition does not require a larger scale factor.

Rounding the answer

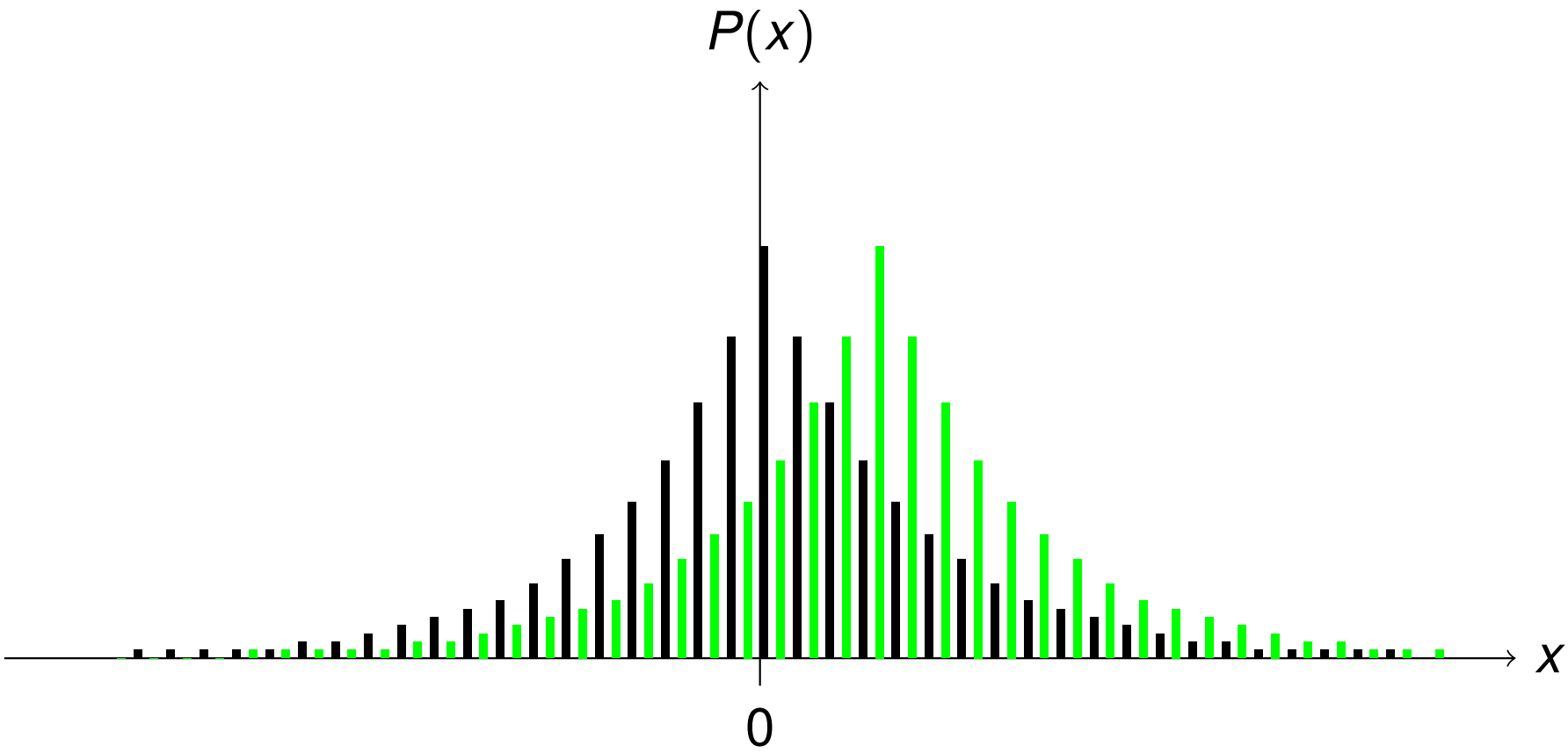
When the set S is too small, the previous bounds are not accurate enough.



Mechanism (rounded)

The mechanism rounds the result by returning the value closest to $f(D) + n'$ *in some discrete subset* S'*.*
$\mathcal{K}(D) = \mathrm{round}(\mathcal{A}(D))$ *where* round *is the rounding function.*

A concrete example



Here, in the implementation of n, the last operation multiplies a value by 4 and returns it.

Compute the ϵ' parameter in the implemented system

Theorem

Our strengthen mechanism is ϵ'-differentially private, with:

$$\epsilon' = \epsilon + \ln\left(1 + Re^{\epsilon\frac{L+\delta_t}{\Delta_{f'}}}\right)$$

- $\delta_t = k\delta_0 + \delta_n$
- $L = \max_{S\in\mathcal{S}'_0} \varnothing S$
- $R = \max_{S\in\mathcal{S}'_0} \frac{\lambda(S^{\delta_t}\setminus S^{-\delta_t})}{\lambda(S^{-\delta_t})} \approx 4m\frac{\delta_t}{L}$

Since $\delta_t \ll L \ll \Delta_f$ and $\epsilon \ll 1$, we have a first order approximation:

$$\epsilon' \approx \epsilon + 4m\frac{\delta_t}{L}$$

Concrete limitations

Actual smallest value for uniform random generator is 2^{-53}.
To compute a Laplace noise, this formula is often used.

$$n(u) = \frac{\Delta_f}{\epsilon} \mathrm{sgn}(u - 1/2) \ln(1 - 2|u - 1/2|)$$

However, $\ln(2^{-53}) \approx -36.7$: the range of n is too small for big databases.

Improvement for the standard case (Laplacian in one dimension)

In this study, we have considered standard noise generation :

- draw uniformly a value u
- *then* compute $n(u)$.

From the logarithm property: $\ln(m2^e) = \ln(m) + e\ln(2)$, we can mix these two previous steps:

- generate m uniformly in $[1, 2]$
- generate e with an exponential law (flip a coin until fair is gotten and return the number of flips)

- The domain for ln is now smooth.
- There remain some issues but they are less critical.

Related work

Ilya Mironov, in a CCS 2012 paper, studied independently the same problem.
He focus mainly on how to implement an attack against the unprotected mechanism.
He provides the same safeguards (rounding and truncation) as well as a bound on the leakage but hypothesis were less general:

- only floating-point representation (not fixed point)
- only the Laplacian noise
- only one dimension (our result is valid for geolocation protocols that need $\mathbb{R}^2$)
- only for a given full-precision implementation

Conclusion and future work

Conclusion:

- Computational errors can be seen as a side channel that leaks information.
- We provide a framework to measure leakage from computational errors.
- We improve the mechanism in one dimension to avoid truncation.

Future work:

- Improve the noise generator in the general case.
- Consider other definitions similar to differential privacy like (ϵ, δ)-differential privacy.
- Use our method to analyze other privacy protocols that use real numbers.

Thank you all for your attention!