# NoiseThresholdPrivacyMap<L01InfDistance<AbsoluteDistance<RBig», Approximate<MaxDivergence» for ZExpFamily<1>

Michael Shoemate

This proof resides in **"contrib"** because it has not completed the vetting process.

Proves soundness of the implementation of NoiseThresholdPrivacyMap for ZExpFamily<1> in mod.rs at commit f5bb719 (outdated[1]).

## 1 Hoare Triple

### Precondition

#### Compiler-Verified

NoiseThresholdPrivacyMap is parameterized as follows:

- MI, the input metric, is of type L01InfDistance<AbsoluteDistance<RBig»
- MO, the output measure, is of type Approximate<MaxDivergence>

#### User-Verified

None

### Pseudocode

```python
# analogous to impl NoiseThresholdPrivacyMap<L01InfDistance<AbsoluteDistance<RBig>>,
    Approximate<MaxDivergence>> for ZExpFamily<1> in Rust
class ZExpFamily1:
    def noise_threshold_privacy_map(
        self,
        _input_metric: L01InfDistance[AbsoluteDistance[RBig]],
        output_measure: Approximate[MaxDivergence],
        threshold: UBig,
    ) -> PrivacyMap[L01InfDistance[AbsoluteDistance[RBig]], Approximate[MaxDivergence]]:
        #
        noise_privacy_map = self.noise_privacy_map(L1Distance.default(), output_measure[0])
        scale = self.scale

        def privacy_map(d_in: tuple[u32, RBig, RBig]):
            l0, l1, li = d_in

```

[1] See new changes with git diff f5bb719..dd73cfdc rust/src/measurements/noise_threshold/distribution/laplace/mod.rs

```
            l1_sign, l1 = l1.floor().into_parts() #
            if l1_sign != Sign.Positive: #
                raise f"l1 sensitivity ({l1}) must be non-negative"

            li_sign, li = li.floor().into_parts() #
            if li_sign != Sign.Positive: #
                raise f"l-infinity sensitivity ({li}) must be non-negative"

            l1 = l1.min(li * l0)  #
            li = li.min(l1)  #

            if l1.is_zero():  #
                return 0.0, 0.0

            if scale.is_zero(): #
                return f64.INFINITY, 1.0

            epsilon = noise_privacy_map.eval(l1) #

            if li > threshold: #
                raise f"threshold must not be smaller than {li}"

            d_instability = threshold - li #

            try:
                alpha_disc = conservative_discrete_laplacian_tail_to_alpha(
                    scale,
                    d_instability
                )
            except Exception:
                alpha_disc = None

            try:
                alpha_cont = conservative_continuous_laplacian_tail_to_alpha(
                    scale,
                    d_instability,
                )
            except Exception:
                alpha_cont = None

            delta_single = option_min(alpha_disc, alpha_cont)
            if delta_single is None:
                raise "failed to compute tail bound in privacy map"

            delta_joint: f64 = (1.0).inf_sub(
                (1.0).neg_inf_sub(delta_single).neg_inf_powi(IBig.from_(l0)),
            )

            # delta is only sensibly at most 1
            return epsilon, delta_joint.min(1.0)

        return PrivacyMap.new_fallible(privacy_map)
```

## Postcondition

**Theorem 1.1.** Given a distribution `self`, returns `Err(e)` if `self` is not a valid distribution. Otherwise the output is `Ok(privacy_map)` where `privacy_map` observes the following:

Define `function(x)` as a function that updates each pair $(k_i, v_i + Z_i)$, where $Z_i$ are iid samples from `self`, and discards pairs where $v_i + Z_i$ has smaller magnitude than `|threshold|`. The ordering of returned pairs is independent from the input ordering.

For every pair of elements $x, x'$ in `VectorDomain<AtomDomain<IBig>>`, and for every pair (`d_in`, `d_out`), where `d_in` has the associated type for `input_metric` and `d_out` has the associated type for `output_measure`, if $x, x'$ are `d_in`-close under `input_metric`, `privacy_map(d_in)` does not raise an exception, and `privacy_map(d_in)` $\leq$ `d_out`, then `function(x)`, `function(x')` are `d_out`-close under `output_measure`.

*Proof.* Line 9 rejects `self` if `self` does not represent a valid distribution, satisfying the error conditions of the postcondition.

We now construct the privacy map. Both the `l1` and `li` sensitivity can be floored, as neighboring integer datasets always differ in whole integer increments. This doesn't affect the check on line 21 to ensure `li` is non-negative.

There are two ways to bound the l1 sensitivity given the three bounds:

1. The `l1` bound directly.

2. Define $x \sim x'$ as $||x - x'||_0 \leq \texttt{l0}$ and $||x - x'||_\infty \leq \texttt{li}$. Then

$$\max_{x \sim x'} ||x - x'||_1 = \max x \sim x' \sum_i^n |x_i - x'_i| \leq \texttt{l0} \cdot \texttt{li} \tag{1}$$

Line 24 updates `l1` to the tighter of these bounds.

`li` similarly has multiple bounds:

1. The `li` bound directly.

2. Define $x \sim x'$ as $||x - x'||_1 \leq \texttt{l1}$. Then

$$\max_{x \sim x'} ||x - x'||_\infty = \max x \sim x' \max_i |x_i - x'_i| \leq \max_i [0, \cdots, \texttt{l1}, \cdots, 0] = \texttt{l1} \tag{2}$$

Line 25 updates `li` to the tighter of these bounds.

Now the `l1` sensitivity is zero if any of the three bounds are zero, due to the tightening of the `l1` bound on line 24. Line 27 then checks for zero sensitivity to return a privacy loss of zero epsilon, zero delta, because all neighboring datasets have identical pairs, and the ordering is randomized.

Otherwise sensitivity is nonzero, so if the scale is zero, the privacy loss is unbounded, shown on line 30.

Now that the edge cases are handled, all sensitivities and scales are finite and strictly positive.

The mechanism that adds noise to the values in the hashmap, as described in the conditions of the postcondition of `NoiseThresholdPrivacyMap`, matches the mechanism that adds noise to a vector, as described in the conditions of the postcondition of `NoisePrivacyMap`. Therefore releasing the values incurs the privacy loss computed by `noise_privacy_map` as defined on line 33, The privacy map also ensures that `l1` is non-negative.

We now focus on computing the remaining privacy loss parameter delta. To ensure that the distance to instability remains positive, the threshold must be at least as large as the sensitivity, as checked on line 35.

Adapting the proof from [Rog23] (Theorem 7). Consider $S$ to be the set of labels that are common between $x$ and $x'$. Define event $E$ to be any potential outcome of the mechanism for which all labels are in $S$ (where only stable partitions are released). We then lower bound the probability of the mechanism returning an event $E$. In the following, $c_j$ denotes the exact count for partition $j$, and $Z_j$ is a random variable distributed according to `self`.

$$\begin{aligned}
\Pr[E] &= \prod_{j \in x \setminus x'} \Pr[c_j + Z_j \leq T] \\
&\geq \prod_{j \in x \setminus x'} \Pr[\Delta_\infty + Z_j \leq T] \\
&\geq \Pr[\Delta_\infty + Z_j \leq T]^{\Delta_0}
\end{aligned}$$

The probability of returning a set of stable partitions ($\Pr[E]$) is the probability of not returning any of the unstable partitions. We now solve for the choice of threshold $T$ such that $\Pr[E] \geq 1 - \delta$.

$$\begin{aligned}\Pr[\Delta_\infty + Z_j \leq T]^{\Delta_0} &= \Pr[Z_j \leq T - \Delta_\infty]^{\Delta_0} \\ &= (1 - \Pr[Z_j > T - \Delta_\infty])^{\Delta_0}\end{aligned}$$

Let `d_instability` denote the distance to instability of $T - \Delta_\infty$.

Notice that when $C$ is a continuous laplace random variable and $D$ is a discrete laplace random variable,

$$\Pr[C > t] = \frac{e^{-t/s}}{2} > \frac{e^{-t/s}}{e^{1/s} + 1} = \Pr[D > t], \tag{3}$$

since $e^{1/s} > 1$. Unfortunately, the discrete laplace tail bound is not feasible to compute for large parameter choices, so we attempt to compute both tail bounds and take the minimum of the two.

By the postconditions of `conservative_discrete_laplacian_tail_to_alpha` and `conservative_continuous_laplacian_tail_to_alpha`, the tail mass is bounded above by both `alpha_disc` and `alpha_cont`. `delta_single` is the smaller of the two, as both are conservative estimates. Both bounds are computed because while the discrete bound is more accurate, it can only be computed for relatively small scales. The continuous bound is a conservative estimate for larger scales that is always computable, and converges to the discrete bound at higher scales.

The probability that a random noise sample exceeds `d_instability` is at most `delta_single`. Therefore $\delta = 1 - (1 - \texttt{delta\_single})^{\Delta_0}$.

The privacy map returns `(epsilon, delta.min(1))`, as $\delta$ is bounded by 1. It is shown that `function`(x), `function`(x') are `d_out`-close under `output_measure`. □

# References


[Rog23] Ryan Rogers. A unifying privacy analysis framework for unknown domain algorithms in differential privacy, 2023.