

# fn make\_laplace

Michael Shoemate

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

Proves soundness of the implementation of `make_laplace` in `mod.rs` at commit `f5bb719` (outdated<sup>1</sup>). Perturbative noise mechanisms may be parameterized along many different axes:

- domain: scalar or vector
- domain dtype: i8, i16, i32, i64, u8, u16, u32, u64, f32, f64, UBig, IBig, RBig
- metric: absolute distance, l1 distance, modular distance
- metric dtype: i8, i16, i32, i64, u8, u16, u32, u64, f32, f64, UBig, IBig, RBig
- measure: max divergence, zero concentrated divergence
- distribution: laplace, gaussian

All parameterizations reduce to a single core mechanism that perturbs a vector of signed big integers with noise sampled from the appropriate discrete distribution.

The implementation of this function constructs a random variable denoting the noise distribution to add, and then dispatches to the `MakeNoise<DI, MI, MO>` trait which constructs the core mechanism and wraps it in pre-processing transformations and post-processors to match the desired parameterization.

## 1 Hoare Triple

### Precondition

#### Compiler-Verified

- generic DI implements trait `Domain`
- generic MI implements trait `Metric`
- generic MO implements trait `Measure`
- type `DiscreteLaplace` implements trait `MakeNoise<DI, MI, MO>` This trait bound constrains the choice of input domain, input metric and output measure to those that can form valid measurements.
- type `(DI, MI)` implements trait `MetricSpace`

#### User-Verified

None

---

<sup>1</sup>See new changes with `git diff f5bb719..356f7ee rust/src/measurements/noise/distribution/laplace/mod.rs`

## Pseudocode

```
1 def make_laplace(
2     input_domain: DI,
3     input_metric: MI,
4     scale: f64,
5     k: Option[i32],
6 ) -> Measurement[DI, DI_Carrier, MI, MO]:
7     return DiscreteLaplace(scale, k).make_noise((input_domain, input_metric))
```

## Postcondition

**Theorem 1.1.** For every setting of the input parameters (`input_domain`, `input_metric`, `scale`, `k`, `DI`, `MI`, `MO`) to `make_laplace` such that the given preconditions hold, `make_laplace` raises an error (at compile time or run time) or returns a valid measurement. A valid measurement has the following properties:

1. (Data-independent runtime errors). For every pair of members  $x$  and  $x'$  in `input_domain`, `invoke(x)` and `invoke(x')` either both return the same error or neither return an error.
2. (Privacy guarantee). For every pair of members  $x$  and  $x'$  in `input_domain` 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 error, and  $\text{privacy\_map}(d_{in}) = d_{out}$ , then `function(x), function(x')` are  $d_{out}$ -close under `output_measure`.

*Proof.* We first construct a random variable `DiscreteLaplace` representing the desired noise distribution. Since `MakeNoise.make_noise` has no preconditions, the postcondition follows, which matches the postcondition for this function.  $\square$