

Example code for computing statistics and histogram

```
/**
 * Collect sample statistics and create a 4096-bin intermediate histogram.
 * When done collecting, convert the histogram to 256-bin zero-centric.
 * The intermediate histogram can also be accessed. But that will have many
 * empty bins and won't be zero-centric.
 *
 * When the builder is not fixed-width, the intermediate size a.k.a. bin count
 * must be at least twice that of the desired result. Choosing a too large
 * size might have a performance impact. Especially on the cost of operator+=.
 *
 * To get a fixed-width histogram, specify the range when creating the builder.
 * The intermediate size must in this case be the same as the final size.
 *
 * Example: SampleHistogramBuilder builder(256, -100, +100).
 *
 * The choice of range is not adjusted to become zero centric. You might want
 * to do that
 * yourself.
 *
 * For multi-threaded computation, use one builder for each thread and combine
 * the results at the end by using operator+=.
 */
static void
example()
{
    const std::vector<float> data1 {-99, 56, 23};
    const std::vector<float> data2 {42};
    const std::vector<float> data3 {0};

    // Add samples from two very short traces.
    SampleHistogramBuilder builder(4096);
    builder.add(data1.data(), data1.data() + data1.size());
    builder.add(data2.data(), data2.data() + data2.size());

    // Add 999 samples with value zero.
    SampleHistogramBuilder tmp(4096);
    tmp.add(data3.data(), data3.data() + data3.size());
    tmp *= 999;
    builder += tmp;

    const SampleStatistics& stats = builder.getstats();
    const SampleHistogram& histo = builder.finalize(256);
}
```

t

Design goals:

- The auto-expand feature is only relevant for floating-point data.
- The application does not specify the histogram range. The range is adjusted as data is added,
- The result will be Zero-centric (0.0 maps to a bin center).
- The result will be independent of the order of data being added, except for issues with floating point instability.
- A float dataset that happens to contain discrete values -128..+127 or similar should have a 1:1 mapping of sample to bin.
- The code should try to minimize issues with numerical instability and rounding mode. Inevitably there will be some sample values that are problematic. But try to avoid having "pretty" numbers such as integers fall in that category.

High level design:

- Internally the code will build a larger histogram (more bins). The internal histogram always has a symmetrical range, i.e. -N..+N.
- The bin width of the internal histogram is constrained to 2^N .
- A templated iterator-based add() is provided.
- If an application frequently calls add() with very small ranges (e.g. once per trace instead of per cube) then the application should provide the retry mechanism instead of add() doing that.
- Much existing code in OpenZGY can be re-used. Some new functions will be needed to adjust the histogram range.
- As an implementation detail, the symmetrical and 2^N requirement might end up tweaked slightly to help avoiding numerical instability.

Acceptable compromises:

- The resulting histogram might not end up using the full range of bins. Typically it will use most but not quite all of them.
- The result might end up using just one bin if input range is weird. Such as +10,000 to +10,001. Or if the input contains large spikes.
- The special -128..+127 case might be too messy to implement.
- Zero-centric might not be enforced if the final range does not in fact contain zero.
- If the histogram needs to be updated after it has been written to file, the range can no longer be changed because the intermediate histogram is no longer available.

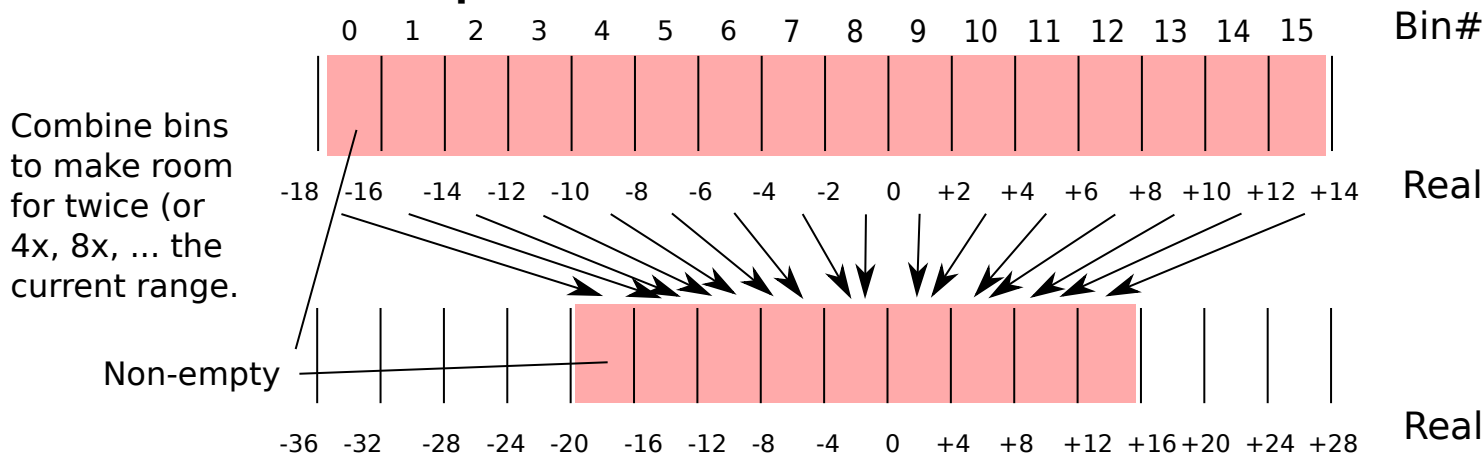
Low level design:

- See figure 1.

Usage:

See **dynamicExample()** and **staticExample()** in `native/src/test/test_histobuilder.cpp`

As needed while samples are collected



After all samples have been collected

