

---

# Boost.Accumulators

Eric Niebler

Copyright © 2005, 2006 Eric Niebler

Distributed under the Boost Software License, Version 1.0. (See accompanying file LICENSE\_1\_0.txt or copy at [http://www.boost.org/LICENSE\\_1\\_0.txt](http://www.boost.org/LICENSE_1_0.txt))

## Table of Contents

Preface .....	4
User's Guide .....	5
The Accumulators Framework .....	5
Using <code>accumulator_set&lt;&gt;</code> .....	7
Extracting Results .....	8
Passing Optional Parameters .....	9
Weighted Samples .....	10
Numeric Operators Sub-Library .....	11
Extending the Accumulators Framework .....	11
Defining a New Accumulator .....	11
Defining a New Feature .....	15
Defining a New Extractor .....	16
Controlling Dependencies .....	17
Specializing Numeric Operators .....	18
Concepts .....	20
The Statistical Accumulators Library .....	20
count .....	21
covariance .....	21
density .....	22
error_of<mean> .....	23
extended_p_square .....	24
extended_p_square_quantile <i>and variants</i> .....	25
kurtosis .....	27
max .....	28
mean <i>and variants</i> .....	28
median <i>and variants</i> .....	31
min .....	32
moment .....	33
p_square_cumulative_distribution .....	34
p_square_quantile <i>and variants</i> .....	35
peaks_over_threshold <i>and variants</i> .....	36
pot_quantile <i>and variants</i> .....	37
pot_tail_mean .....	40
rolling_count .....	40
rolling_sum .....	41
rolling_mean .....	42
rolling_moment .....	43
rolling_variance .....	44
skewness .....	45
sum <i>and variants</i> .....	46
tail .....	47
coherent_tail_mean .....	48
non_coherent_tail_mean .....	49
tail_quantile .....	50

tail_variate .....	52
tail_variate_means <i>and variants</i> .....	54
variance <i>and variants</i> .....	57
weighted_covariance .....	58
weighted_density .....	59
weighted_extended_p_square .....	60
weighted_kurtosis .....	62
weighted_mean <i>and variants</i> .....	63
weighted_median <i>and variants</i> .....	65
weighted_moment .....	66
weighted_p_square_cumulative_distribution .....	67
weighted_p_square_quantile <i>and variants</i> .....	70
weighted_peaks_over_threshold <i>and variants</i> .....	72
weighted_skewness .....	73
weighted_sum <i>and variants</i> .....	74
non_coherent_weighted_tail_mean .....	75
weighted_tail_quantile .....	77
weighted_tail_variate_means <i>and variants</i> .....	78
weighted_variance <i>and variants</i> .....	82
Acknowledgements .....	84
Reference .....	85
Accumulators Framework Reference .....	85
Header <boost/accumulators/accumulators.hpp> .....	85
Header <boost/accumulators/accumulators_fwd.hpp> .....	85
Header <boost/accumulators/framework/accumulator_base.hpp> .....	94
Header <boost/accumulators/framework/accumulator_concept.hpp> .....	96
Header <boost/accumulators/framework/accumulator_set.hpp> .....	96
Header <boost/accumulators/framework/accumulators/droppable_accumulator.hpp> .....	97
Header <boost/accumulators/framework/accumulators/external_accumulator.hpp> .....	98
Header <boost/accumulators/framework/accumulators/reference_accumulator.hpp> .....	99
Header <boost/accumulators/framework/accumulators/value_accumulator.hpp> .....	101
Header <boost/accumulators/framework/depends_on.hpp> .....	102
Header <boost/accumulators/framework/extractor.hpp> .....	102
Header <boost/accumulators/framework/features.hpp> .....	102
Header <boost/accumulators/framework/parameters/accumulator.hpp> .....	102
Header <boost/accumulators/framework/parameters/sample.hpp> .....	103
Header <boost/accumulators/framework/parameters/weight.hpp> .....	103
Header <boost/accumulators/framework/parameters/weights.hpp> .....	104
Statistics Library Reference .....	105
Header <boost/accumulators/statistics.hpp> .....	105
Header <boost/accumulators/statistics/count.hpp> .....	105
Header <boost/accumulators/statistics/covariance.hpp> .....	107
Header <boost/accumulators/statistics/density.hpp> .....	110
Header <boost/accumulators/statistics/error_of.hpp> .....	111
Header <boost/accumulators/statistics/error_of_mean.hpp> .....	112
Header <boost/accumulators/statistics/extended_p_square.hpp> .....	113
Header <boost/accumulators/statistics/extended_p_square_quantile.hpp> .....	115
Header <boost/accumulators/statistics/kurtosis.hpp> .....	120
Header <boost/accumulators/statistics/max.hpp> .....	121
Header <boost/accumulators/statistics/mean.hpp> .....	122
Header <boost/accumulators/statistics/median.hpp> .....	129
Header <boost/accumulators/statistics/min.hpp> .....	134
Header <boost/accumulators/statistics/moment.hpp> .....	134
Header <boost/accumulators/statistics/p_square_cumul_dist.hpp> .....	135
Header <boost/accumulators/statistics/p_square_quantile.hpp> .....	137
Header <boost/accumulators/statistics/peaks_over_threshold.hpp> .....	139
Header <boost/accumulators/statistics/pot_quantile.hpp> .....	144
Header <boost/accumulators/statistics/pot_tail_mean.hpp> .....	147

Header <boost/accumulators/statistics/rolling_count.hpp> .....	150
Header <boost/accumulators/statistics/rolling_mean.hpp> .....	151
Header <boost/accumulators/statistics/rolling_moment.hpp> .....	156
Header <boost/accumulators/statistics/rolling_sum.hpp> .....	157
Header <boost/accumulators/statistics/rolling_variance.hpp> .....	158
Header <boost/accumulators/statistics/rolling_window.hpp> .....	164
Header <boost/accumulators/statistics/skewness.hpp> .....	165
Header <boost/accumulators/statistics/stats.hpp> .....	166
Header <boost/accumulators/statistics/sum.hpp> .....	167
Header <boost/accumulators/statistics/sum_kahan.hpp> .....	169
Header <boost/accumulators/statistics/tail.hpp> .....	173
Header <boost/accumulators/statistics/tail_mean.hpp> .....	175
Header <boost/accumulators/statistics/tail_quantile.hpp> .....	177
Header <boost/accumulators/statistics/tail_variate.hpp> .....	179
Header <boost/accumulators/statistics/tail_variate_means.hpp> .....	181
Header <boost/accumulators/statistics/times2_iterator.hpp> .....	185
Header <boost/accumulators/statistics/variance.hpp> .....	185
Header <boost/accumulators/statistics/variates/covariate.hpp> .....	188
Header <boost/accumulators/statistics/weighted_covariance.hpp> .....	189
Header <boost/accumulators/statistics/weighted_density.hpp> .....	190
Header <boost/accumulators/statistics/weighted_extended_p_square.hpp> .....	191
Header <boost/accumulators/statistics/weighted_kurtosis.hpp> .....	192
Header <boost/accumulators/statistics/weighted_mean.hpp> .....	193
Header <boost/accumulators/statistics/weighted_median.hpp> .....	195
Header <boost/accumulators/statistics/weighted_moment.hpp> .....	197
Header <boost/accumulators/statistics/weighted_p_square_cumul_dist.hpp> .....	197
Header <boost/accumulators/statistics/weighted_p_square_quantile.hpp> .....	198
Header <boost/accumulators/statistics/weighted_peaks_over_threshold.hpp> .....	200
Header <boost/accumulators/statistics/weighted_skewness.hpp> .....	201
Header <boost/accumulators/statistics/weighted_sum.hpp> .....	202
Header <boost/accumulators/statistics/weighted_sum_kahan.hpp> .....	203
Header <boost/accumulators/statistics/weighted_tail_mean.hpp> .....	205
Header <boost/accumulators/statistics/weighted_tail_quantile.hpp> .....	205
Header <boost/accumulators/statistics/weighted_tail_variate_means.hpp> .....	206
Header <boost/accumulators/statistics/weighted_variance.hpp> .....	208
Header <boost/accumulators/statistics/with_error.hpp> .....	210
Header <boost/accumulators/statistics_fwd.hpp> .....	210
Numeric Operators Library Reference .....	272
Header <boost/accumulators/numeric/functional.hpp> .....	272
Header <boost/accumulators/numeric/functional/complex.hpp> .....	308
Header <boost/accumulators/numeric/functional/valarray.hpp> .....	309
Header <boost/accumulators/numeric/functional/vector.hpp> .....	314

# Preface

“It is better to be approximately right than exactly wrong.”  
-- *Old adage*

## Description

Boost.Accumulators is both a library for incremental statistical computation as well as an extensible framework for incremental calculation in general. The library deals primarily with the concept of an *accumulator*, which is a primitive computational entity that accepts data one sample at a time and maintains some internal state. These accumulators may offload some of their computations on other accumulators, on which they depend. Accumulators are grouped within an *accumulator set*. Boost.Accumulators resolves the inter-dependencies between accumulators in a set and ensures that accumulators are processed in the proper order.

# User's Guide

This section describes how to use the Boost.Accumulators framework to create new accumulators and how to use the existing statistical accumulators to perform incremental statistical computation. For detailed information regarding specific components in Boost.Accumulators, check the [Reference](#) section.

## Hello, World!

Below is a complete example of how to use the Accumulators Framework and the Statistical Accumulators to perform an incremental statistical calculation. It calculates the mean and 2nd moment of a sequence of doubles.

```
#include <iostream>
#include <boost/accumulators/accumulators.hpp>
#include <boost/accumulators/statistics/stats.hpp>
#include <boost/accumulators/statistics/mean.hpp>
#include <boost/accumulators/statistics/moment.hpp>
using namespace boost::accumulators;

int main()
{
    // Define an accumulator set for calculating the mean and the
    // 2nd moment ...
    accumulator_set<double, stats<tag::mean, tag::moment<2> > > acc;

    // push in some data ...
    acc(1.2);
    acc(2.3);
    acc(3.4);
    acc(4.5);

    // Display the results ...
    std::cout << "Mean:   " << mean(acc) << std::endl;
    std::cout << "Moment: " << accumulators::moment<2>(acc) << std::endl;

    return 0;
}
```

This program displays the following:

```
Mean:   2.85
Moment: 9.635
```

## The Accumulators Framework

The Accumulators Framework is framework for performing incremental calculations. Usage of the framework follows the following pattern:

- Users build a computational object, called an `accumulator_set<>`, by selecting the computations in which they are interested, or authoring their own computational primitives which fit within the framework.
- Users push data into the `accumulator_set<>` object one sample at a time.
- The `accumulator_set<>` computes the requested quantities in the most efficient method possible, resolving dependencies between requested calculations, possibly caching intermediate results.

The Accumulators Framework defines the utilities needed for defining primitive computational elements, called *accumulators*. It also provides the `accumulator_set<>` type, described above.

## Terminology

The following terms are used in the rest of the documentation.

Sample	A datum that is pushed into an <code>accumulator_set&lt;&gt;</code> . The type of the sample is the <i>sample type</i> .
Weight	An optional scalar value passed along with the sample specifying the weight of the sample. Conceptually, each sample is multiplied with its weight. The type of the weight is the <i>weight type</i> .
Feature	An abstract primitive computational entity. When defining an <code>accumulator_set&lt;&gt;</code> , users specify the features in which they are interested, and the <code>accumulator_set&lt;&gt;</code> figures out which <i>accumulators</i> would best provide those features. Features may depend on other features. If they do, the accumulator set figures out which accumulators to add to satisfy the dependencies.
Accumulator	A concrete primitive computational entity. An accumulator is a concrete implementation of a feature. It satisfies exactly one abstract feature. Several different accumulators may provide the same feature, but may represent different implementation strategies.
Accumulator Set	A collection of accumulators. An accumulator set is specified with a sample type and a list of features. The accumulator set uses this information to generate an ordered set of accumulators depending on the feature dependency graph. An accumulator set accepts samples one datum at a time, propagating them to each accumulator in order. At any point, results can be extracted from the accumulator set.
Extractor	A function or function object that can be used to extract a result from an <code>accumulator_set&lt;&gt;</code> .

## Overview

Here is a list of the important types and functions in the Accumulator Framework and a brief description of each.

**Table 1. Accumulators Toolbox**

Tool	Description
<code>accumulator_set&lt;&gt;</code>	This is the most important type in the Accumulators Framework. It is a collection of accumulators. A datum pushed into an <code>accumulator_set&lt;&gt;</code> is forwarded to each accumulator, in an order determined by the dependency relationships between the accumulators. Computational results can be extracted from an accumulator at any time.
<code>depends_on&lt;&gt;</code>	Used to specify which other features a feature depends on.
<code>feature_of&lt;&gt;</code>	Trait used to tell the Accumulators Framework that, for the purpose of feature-based dependency resolution, one feature should be treated the same as another.
<code>as_feature&lt;&gt;</code>	Used to create an alias for a feature. For example, if there are two features, <code>fast_X</code> and <code>accurate_X</code> , they can be mapped to <code>X(fast)</code> and <code>X(accurate)</code> with <code>as_feature&lt;&gt;</code> . This is just syntactic sugar.
<code>features&lt;&gt;</code>	An <a href="#">MPL</a> sequence. We can use <code>features&lt;&gt;</code> as the second template parameter when declaring an <code>accumulator_set&lt;&gt;</code> .
<code>external&lt;&gt;</code>	Used when declaring an <code>accumulator_set&lt;&gt;</code> . If the weight type is specified with <code>external&lt;&gt;</code> , then the weight accumulators are assumed to reside in a separate accumulator set which will be passed in with a named parameter.
<code>extractor&lt;&gt;</code>	A class template useful for creating an extractor function object. It is parameterized on a feature, and it has member functions for extracting from an <code>accumulator_set&lt;&gt;</code> the result corresponding to that feature.

## Using `accumulator_set<>`

Our tour of the `accumulator_set<>` class template begins with the forward declaration:

```
template< typename Sample, typename Features, typename Weight = void >
struct accumulator_set;
```

The template parameters have the following meaning:

**Sample**      The type of the data that will be accumulated.

**Features**    An [MPL](#) sequence of features to be calculated.

**Weight**      The type of the (optional) weight paramter.

For example, the following line declares an `accumulator_set<>` that will accept a sequence of doubles one at a time and calculate the min and mean:

```
accumulator_set< double, features< tag::min, tag::mean > > acc;
```

Notice that we use the `features<>` template to specify a list of features to be calculated. `features<>` is an [MPL](#) sequence of features.

**Note**

`features<>` is a synonym of `mpl::vector<>`. In fact, we could use `mpl::vector<>` or any MPL sequence if we prefer, and the meaning would be the same.

Once we have defined an `accumulator_set<>`, we can then push data into it, and it will calculate the quantities you requested, as shown below.

```
// push some data into the accumulator_set ...
acc(1.2);
acc(2.3);
acc(3.4);
```

Since `accumulator_set<>` defines its accumulate function to be the function call operator, we might be tempted to use an `accumulator_set<>` as a UnaryFunction to a standard algorithm such as `std::for_each`. That's fine as long as we keep in mind that the standard algorithms take UnaryFunction objects by value, which involves making a copy of the `accumulator_set<>` object. Consider the following:

```
// The data for which we wish to calculate statistical properties:
std::vector< double > data( /* stuff */ );

// The accumulator set which will calculate the properties for us:
accumulator_set< double, features< tag::min, tag::mean > > acc;

// Use std::for_each to accumulate the statistical properties:
acc = std::for_each( data.begin(), data.end(), acc );
```

Notice how we must assign the return value of `std::for_each` back to the `accumulator_set<>`. This works, but some accumulators are not cheap to copy. For example, the `tail` and `tail_variate<>` accumulators must store a `std::vector<>`, so copying these accumulators involves a dynamic allocation. We might be better off in this case passing the accumulator by reference, with the help of `boost::bind()` and `boost::ref()`. See below:

```
// The data for which we wish to calculate statistical properties:
std::vector< double > data( /* stuff */ );

// The accumulator set which will calculate the properties for us:
accumulator_set< double, features< tag::tail<left> > > acc(
    tag::tail<left>::cache_size = 4 );

// Use std::for_each to accumulate the statistical properties:
std::for_each( data.begin(), data.end(), bind<void>( ref(acc), _1 ) );
```

Notice now that we don't care about the return value of `std::for_each()` anymore because `std::for_each()` is modifying `acc` directly.

**Note**

To use `boost::bind()` and `boost::ref()`, you must `#include <boost/bind.hpp>` and `<boost/ref.hpp>`

## Extracting Results

Once we have declared an `accumulator_set<>` and pushed data into it, we need to be able to extract results from it. For each feature we can add to an `accumulator_set<>`, there is a corresponding extractor for fetching its result. Usually, the extractor has the same name as the feature, but in a different namespace. For example, if we accumulate the `tag::min` and `tag::max` features, we can extract the results with the `min` and `max` extractors, as follows:



```
// Calculate the minimum and maximum for a sequence of integers.
accumulator_set< int, features< tag::min, tag::max > > acc;
acc( 2 );
acc( -1 );
acc( 1 );

// This displays "(-1, 2)"
std::cout << '(' << min( acc ) << ", " << max( acc ) << ")\n";
```

The extractors are all declared in the `boost::accumulators::extract` namespace, but they are brought into the `boost::accumulators` namespace with a `using` declaration.



## Tip

On the Windows platform, `min` and `max` are preprocessor macros defined in `WinDef.h`. To use the `min` and `max` extractors, you should either compile with `NOMINMAX` defined, or you should invoke the extractors like: `(min)( acc )` and `(max)( acc )`. The parentheses keep the macro from being invoked.

Another way to extract a result from an `accumulator_set<>` is with the `extract_result()` function. This can be more convenient if there isn't an extractor object handy for a certain feature. The line above which displays results could equally be written as:

```
// This displays "(-1, 2)"
std::cout << '(' << extract_result< tag::min >( acc )
          << ", " << extract_result< tag::max >( acc ) << ")\n";
```

Finally, we can define our own extractor using the `extractor<>` class template. For instance, another way to avoid the `min` / `max` macro business would be to define extractors with names that don't conflict with the macros, like this:

```
extractor< tag::min > min_;
extractor< tag::min > max_;

// This displays "(-1, 2)"
std::cout << '(' << min_( acc ) << ", " << max_( acc ) << ")\n";
```

## Passing Optional Parameters

Some accumulators need initialization parameters. In addition, perhaps some auxiliary information needs to be passed into the `accumulator_set<>` along with each sample. Boost.Accumulators handles these cases with named parameters from the [Boost.Parameter](#) library.

For example, consider the `tail` and `tail_variate<>` features. `tail` keeps an ordered list of the largest  $N$  samples, where  $N$  can be specified at construction time. Also, the `tail_variate<>` feature, which depends on `tail`, keeps track of some data that is co-variate with the  $N$  samples tracked by `tail`. The code below shows how this all works, and is described in more detail below.

```
// Define a feature for tracking covariate data
typedef tag::tail_variate< int, tag::covariate1, left > my_tail_variate_tag;

// This will calculate the left tail and my_tail_variate_tag for N == 2
// using the tag::tail<left>::cache_size named parameter
accumulator_set< double, features< my_tail_variate_tag > > acc(
    tag::tail<left>::cache_size = 2 );

// push in some samples and some covariates by using
// the covariate1 named parameter
acc( 1.2, covariate1 = 12 );
acc( 2.3, covariate1 = -23 );
acc( 3.4, covariate1 = 34 );
acc( 4.5, covariate1 = -45 );

// Define an extractor for the my_tail_variate_tag feature
extractor< my_tail_variate_tag > my_tail_variate;

// Write the tail statistic to std::cout. This will print "4.5, 3.4, "
std::ostream_iterator< double > dout( std::cout, ", " );
std::copy( tail( acc ).begin(), tail( acc ).end(), dout );

// Write the tail_variate statistic to std::cout. This will print "-45, 34, "
std::ostream_iterator< int > iout( std::cout, ", " );
std::copy( my_tail_variate( acc ).begin(), my_tail_variate( acc ).end(), iout );
```

There are several things to note about the code above. First, notice that we didn't have to request that the `tail` feature be calculated. That is implicit because the `tail_variate<>` feature depends on the `tail` feature. Next, notice how the `acc` object is initialized: `acc( tag::tail<left>::cache_size = 2 )`. Here, `cache_size` is a named parameter. It is used to tell the `tail` and `tail_variate<>` accumulators how many samples and covariates to store. Conceptually, every construction parameter is made available to every accumulator in an accumulator set.

We also use a named parameter to pass covariate data into the accumulator set along with the samples. As with the constructor parameters, all parameters to the `accumulate` function are made available to all the accumulators in the set. In this case, only the accumulator for the `my_tail_variate` feature would be interested in the value of the `covariate1` named parameter.

We can make one final observation about the example above. Since `tail` and `tail_variate<>` are multi-valued features, the result we extract for them is represented as an iterator range. That is why we can say `tail( acc ).begin()` and `tail( acc ).end()`.

Even the extractors can accept named parameters. In a bit, we'll see a situation where that is useful.

## Weighted Samples

Some accumulators, statistical accumulators in particular, deal with data that are *weighted*. Each sample pushed into the accumulator has an associated weight, by which the sample is conceptually multiplied. The Statistical Accumulators Library provides an assortment of these weighted statistical accumulators. And many unweighted statistical accumulators have weighted variants. For instance, the weighted variant of the `sum` accumulator is called `weighted_sum`, and is calculated by accumulating all the samples multiplied by their weights.

To declare an `accumulator_set<>` that accepts weighted samples, you must specify the type of the weight parameter as the 3rd template parameter, as follows:

```
// 3rd template parameter 'int' means this is a weighted
// accumulator set where the weights have type 'int'
accumulator_set< int, features< tag::sum >, int > acc;
```

When you specify a weight, all the accumulators in the set are replaced with their weighted equivalents. For example, the above `accumulator_set<>` declaration is equivalent to the following:

```
// Since we specified a weight, tag::sum becomes tag::weighted_sum
accumulator_set< int, features< tag::weighted_sum >, int > acc;
```

When passing samples to the accumulator set, you must also specify the weight of each sample. You can do that with the `weight` named parameter, as follows:

```
acc(1, weight = 2); // 1 * 2
acc(2, weight = 4); // 2 * 4
acc(3, weight = 6); // + 3 * 6
                    // -----
                    // =      28
```

You can then extract the result with the `sum()` extractor, as follows:

```
// This prints "28"
std::cout << sum(acc) << std::endl;
```



### Note

When working with weighted statistical accumulators from the Statistical Accumulators Library, be sure to include the appropriate header. For instance, `weighted_sum` is defined in `<boost/accumulators/statistics/weighted_sum.hpp>`.

## Numeric Operators Sub-Library

This section describes the function objects in the `boost::numeric` namespace, which is a sub-library that provides function objects and meta-functions corresponding to the infix operators in C++.

In the `boost::numeric::operators` namespace are additional operator overloads for some useful operations not provided by the standard library, such as multiplication of a `std::complex<>` with a scalar.

In the `boost::numeric::functional` namespace are function object equivalents of the infix operators. These function object types are heterogeneous, and so are more general than the standard ones found in the `<functional>` header. They use the `Boost.Typeof` library to deduce the return types of the infix expressions they evaluate. In addition, they look within the `boost::numeric::operators` namespace to consider any additional overloads that might be defined there.

In the `boost::numeric` namespace are global polymorphic function objects corresponding to the function object types defined in the `boost::numeric::functional` namespace. For example, `boost::numeric::plus(a, b)` is equivalent to `boost::numeric::functional::plus<A, B>()(a, b)`, and both are equivalent to using namespace `boost::numeric::operators`; `a + b`;

The Numeric Operators Sub-Library also gives several ways to sub-class and a way to sub-class and specialize operations. One way uses tag dispatching on the types of the operands. The other way is based on the compile-time properties of the operands.

## Extending the Accumulators Framework

This section describes how to extend the Accumulators Framework by defining new accumulators, features and extractors. Also covered are how to control the dependency resolution of features within an accumulator set.

### Defining a New Accumulator

All new accumulators must satisfy the [Accumulator Concept](#). Below is a sample class that satisfies the accumulator concept, which simply sums the values of all samples passed into it.

```
#include <boost/accumulators/framework/accumulator_base.hpp>
#include <boost/accumulators/framework/parameters/sample.hpp>

namespace boost {
    namespace accumulators {
        namespace impl {
            // Putting your accumulators in the
            // impl namespace has some
            // advantages. See below.

            template<typename Sample>
            struct sum_accumulator
            : accumulator_base
            {
                typedef Sample result_type;
                // The type returned by result() below.

                template<typename Args>
                sum_accumulator(Args const & args)
                : sum(args[sample | Sample()])
                {
                    // The constructor takes an argument pack.
                    // Maybe there is an initial value in the
                    // argument pack. ('sample' is defined in
                    // sample.hpp, included above.)
                }

                template<typename Args>
                void operator ()(Args const & args)
                {
                    // The accumulate function is the function
                    // call operator, and it also accepts an
                    // argument pack.
                    this->sum += args[sample];
                }

                result_type result(dont_care) const
                {
                    // The result function will also be passed
                    // an argument pack, but we don't use it here,
                    // so we use "dont_care" as the argument type.
                    return this->sum;
                }
            private:
                Sample sum;
            };
        }
    }
}
```

Much of the above should be pretty self-explanatory, except for the use of argument packs which may be confusing if you have never used the [Boost.Parameter](#) library before. An argument pack is a cluster of values, each of which can be accessed with a key. So `args[sample]` extracts from the pack the value associated with the `sample` key. And the cryptic `args[sample | Sample()]` evaluates to the value associated with the `sample` key if it exists, or a default-constructed `Sample` if it doesn't.

The example above demonstrates the most common attributes of an accumulator. There are other optional member functions that have special meaning. In particular:

### Optional Accumulator Member Functions

`on_drop(Args)` Defines an action to be taken when this accumulator is dropped. See the section on [Droppable Accumulators](#).

## Accessing Other Accumulators in the Set

Some accumulators depend on other accumulators within the same accumulator set. In those cases, it is necessary to be able to access those other accumulators. To make this possible, the `accumulator_set<>` passes a reference to itself when invoking the member functions of its contained accumulators. It can be accessed by using the special `accumulator` key with the argument pack. Consider how we might implement `mean_accumulator`:

```
// Mean == (Sum / Count)
template<typename Sample>
struct mean_accumulator : accumulator_base
{
    typedef Sample result_type;
    mean_accumulator(dont_care) {}

    template<typename Args>
    result_type result(Args const &args) const
    {
        return sum(args[accumulator]) / count(args[accumulator]);
    }
};
```

mean depends on the `sum` and `count` accumulators. (We'll see in the next section how to specify these dependencies.) The result of the mean accumulator is merely the result of the sum accumulator divided by the result of the count accumulator. Consider how we write that: `sum(args[accumulator]) / count(args[accumulator])`. The expression `args[accumulator]` evaluates to a reference to the `accumulator_set<>` that contains this `mean_accumulator`. It also contains the `sum` and `count` accumulators, and we can access their results with the extractors defined for those features: `sum` and `count`.



### Note

Accumulators that inherit from `accumulator_base` get an empty operator `()`, so accumulators like `mean_accumulator` above need not define one.

All the member functions that accept an argument pack have access to the enclosing `accumulator_set<>` via the `accumulator` key, including the constructor. The accumulators within the set are constructed in an order determined by their interdependencies. As a result, it is safe for an accumulator to access one on which it depends during construction.

## Infix Notation and the Numeric Operators Sub-Library

Although not necessary, it can be a good idea to put your accumulator implementations in the `boost::accumulators::impl` namespace. This namespace pulls in any operators defined in the `boost::numeric::operators` namespace with a `using` directive. The Numeric Operators Sub-Library defines some additional overloads that will make your accumulators work with all sorts of data types.

Consider `mean_accumulator` defined above. It divides the sum of the samples by the count. The type of the count is `std::size_t`. What if the sample type doesn't define division by `std::size_t`? That's the case for `std::complex<>`. You might think that if the sample type is `std::complex<>`, the code would not work, but in fact it does. That's because Numeric Operators Sub-Library defines an overloaded operator `/` for `std::complex<>` and `std::size_t`. This operator is defined in the `boost::numeric::operators` namespace and will be found within the `boost::accumulators::impl` namespace. That's why it's a good idea to put your accumulators there.

## Droppable Accumulators

The term "droppable" refers to an accumulator that can be removed from the `accumulator_set<>`. You can request that an accumulator be made droppable by using the `droppable<>` class template.

```
// calculate sum and count, make sum droppable:
accumulator_set< double, features< tag::count, droppable<tag::sum> > > acc;

// add some data
acc(3.0);
acc(2.0);

// drop the sum (sum is 5 here)
acc.drop<tag::sum>();

// add more data
acc(1.0);

// This will display "3" and "5"
std::cout << count(acc) << ' ' << sum(acc);
```

Any accumulators that get added to an accumulator set in order to satisfy dependencies on droppable accumulators are themselves droppable. Consider the following accumulator:

```
// Sum is not droppable. Mean is droppable. Count, brought in to
// satisfy mean's dependencies, is implicitly droppable, too.
accumulator_set< double, features< tag::sum, droppable<tag::mean> > > acc;
```

mean depends on sum and count. Since mean is droppable, so too is count. However, we have explicitly requested that sum be not droppable, so it isn't. Had we left tag::sum out of the above declaration, the sum accumulator would have been implicitly droppable.

A droppable accumulator is reference counted, and is only really dropped after all the accumulators that depend on it have been dropped. This can lead to some surprising behavior in some situations.

```
// calculate sum and mean, make mean droppable.
accumulator_set< double, features< tag::sum, droppable<tag::mean> > > acc;

// add some data
acc(1.0);
acc(2.0);

// drop the mean. mean's reference count
// drops to 0, so it's really dropped. So
// too, count's reference count drops to 0
// and is really dropped.
acc.drop<tag::mean>();

// add more data. Sum continues to accumulate!
acc(3.0);

// This will display "6 2 3"
std::cout << sum(acc) << ' '
          << count(acc) << ' '
          << mean(acc);
```

Note that at the point at which mean is dropped, sum is 3, count is 2, and therefore mean is 1.5. But since sum continues to accumulate even after mean has been dropped, the value of mean continues to change. If you want to remember the value of mean at the point it is dropped, you should save its value into a local variable.

The following rules more precisely specify how droppable and non-droppable accumulators behave within an accumulator set.

- There are two types of accumulators: droppable and non-droppable. The default is non-droppable.
- For any feature *x*, both *x* and droppable<*x*> satisfy the *x* dependency.

- If feature `x` depends on `y` and `z`, then `droppable<x>` depends on `droppable<y>` and `droppable<z>`.
- All accumulators have `add_ref()` and `drop()` member functions.
- For non-droppable accumulators, `drop()` is a no-op, and `add_ref()` invokes `add_ref()` on all accumulators corresponding to the features upon which the current accumulator depends.
- Droppable accumulators have a reference count and define `add_ref()` and `drop()` to manipulate the reference count.
- For droppable accumulators, `add_ref()` increments the accumulator's reference count, and also `add_ref()`'s the accumulators corresponding to the features upon which the current accumulator depends.
- For droppable accumulators, `drop()` decrements the accumulator's reference count, and also `drop()`'s the accumulators corresponding to the features upon which the current accumulator depends.
- The `accumulator_set` constructor walks the list of **user-specified** features and `add_ref()`'s the accumulator that corresponds to each of them. (Note: that means that an accumulator that is not user-specified but in the set merely to satisfy a dependency will be dropped as soon as all its dependencies have been dropped. Ones that have been user specified are not dropped until their dependencies have been dropped **and** the user has explicitly dropped the accumulator.)
- Droppable accumulators check their reference count in their `accumulate` member function. If the reference count is 0, the function is a no-op.
- Users are not allowed to drop a feature that is not user-specified and marked as droppable.

And as an optimization:

- If the user specifies the non-droppable feature `x`, which depends on `y` and `z`, then the accumulators for `y` and `z` can be safely made non-droppable, as well as any accumulators on which they depend.

## Defining a New Feature

Once we have implemented an accumulator, we must define a feature for it so that users can specify the feature when declaring an `accumulator_set<>`. We typically put the features into a nested namespace, so that later we can define an extractor of the same name. All features must satisfy the [Feature Concept](#). Using `depends_on<>` makes satisfying the concept simple. Below is an example of a feature definition.

```
namespace boost { namespace accumulators { namespace tag {  
  
    struct mean                                // Features should inherit from  
    : depends_on< count, sum >                // depends_on<> to specify dependencies  
    {  
        // Define a nested typedef called 'impl' that specifies which  
        // accumulator implements this feature.  
        typedef accumulators::impl::mean_accumulator< mpl::_1 > impl;  
    };  
  
}}}
```

The only two things we must do to define the `mean` feature is to specify the dependencies with `depends_on<>` and define the nested `impl` typedef. Even features that have no dependencies should inherit from `depends_on<>`. The nested `impl` type must be an [MPL Lambda Expression](#). The result of `mpl::apply< impl, sample-type, weight-type >::type` must be the type of the accumulator that implements this feature. The use of [MPL](#) placeholders like `mpl::_1` make it especially easy to make a template such as `mean_accumulator<>` an [MPL Lambda Expression](#). Here, `mpl::_1` will be replaced with the sample type. Had we used `mpl::_2`, it would have been replaced with the weight type.

What about accumulator types that are not templates? If you have a `foo_accumulator` which is a plain struct and not a template, you could turn it into an [MPL Lambda Expression](#) using `mpl::always<>`, like this:

```
// An MPL lambda expression that always evaluates to
// foo_accumulator:
typedef mpl::always< foo_accumulator > impl;
```

If you are ever unsure, or if you are not comfortable with MPL lambda expressions, you could always define `impl` explicitly:

```
// Same as 'typedef mpl::always< foo_accumulator > impl;'
struct impl
{
    template< typename Sample, typename Weight >
    struct apply
    {
        typedef foo_accumulator type;
    };
};
```

Here, `impl` is a binary [MPL Metafunction Class](#), which is a kind of [MPL Lambda Expression](#). The nested `apply<>` template is part of the metafunction class protocol and tells MPL how to build the accumulator type given the sample and weight types.

All features must also provide a nested `is_weight_accumulator` typedef. It must be either `mpl::true_` or `mpl::false_`. [depends\\_on<>](#) provides a default of `mpl::false_` for all features that inherit from it, but that can be overridden (or hidden, technically speaking) in the derived type. When the feature represents an accumulation of information about the weights instead of the samples, we can mark this feature as such with `typedef mpl::true_ is_weight_accumulator;`. The weight accumulators are made external if the weight type is specified using the `external<>` template.

## Defining a New Extractor

Now that we have an accumulator and a feature, the only thing lacking is a way to get results from the accumulator set. The Accumulators Framework provides the `extractor<>` class template to make it simple to define an extractor for your feature. Here's an extractor for the mean feature we defined above:

```
namespace boost {
namespace accumulators {
namespace extract {
    // By convention, we put extractors
    // in the 'extract' namespace

    extractor< tag::mean > const mean = {}; // Simply define our extractor with
    // our feature tag, like this.
}
using extract::mean;
    // Pull the extractor into the
    // enclosing namespace.
}
```

Once defined, the mean extractor can be used to extract the result of the `tag::mean` feature from an `accumulator_set<>`.

Parameterized features complicate this simple picture. Consider the `moment` feature, for calculating the  $N$ -th moment, where  $N$  is specified as a template parameter:

```
// An accumulator set for calculating the N-th moment, for N == 2 ...
accumulator_set< double, features< tag::moment<2> > > acc;

// ... add some data ...

// Display the 2nd moment ...
std::cout << "2nd moment is " << accumulators::moment<2>(acc) << std::endl;
```

In the expression `accumulators::moment<2>(acc)`, what is `moment`? It cannot be an object -- the syntax of C++ will not allow it. Clearly, if we want to provide this syntax, we must make `moment` a function template. Here's what the definition of the `moment` extractor looks like:



```
namespace boost {
namespace accumulators {
namespace extract {
    // By convention, we put extractors
    // in the 'extract' namespace

    template<int N, typename AccumulatorSet>
    typename mpl::apply<AccumulatorSet, tag::moment<N> >::type::result_type
    moment(AccumulatorSet const &acc)
    {
        return extract_result<tag::moment<N> >(acc);
    }
}
using extract::moment;
// Pull the extractor into the
// enclosing namespace.
}
```

The return type deserves some explanation. Every `accumulator_set<>` type is actually a unary [MPL Metafunction Class](#). When you `mpl::apply<>` an `accumulator_set<>` and a feature, the result is the type of the accumulator within the set that implements that feature. And every accumulator provides a nested `result_type` typedef that tells what its return type is. The extractor simply delegates its work to the `extract_result()` function.

## Controlling Dependencies

The feature-based dependency resolution of the Accumulators Framework is designed to allow multiple different implementation strategies for each feature. For instance, two different accumulators may calculate the same quantity with different rounding modes, or using different algorithms with different size/speed tradeoffs. Other accumulators that depend on that quantity shouldn't care how it's calculated. The Accumulators Framework handles this by allowing several different accumulators satisfy the same feature.

### Aliasing feature dependencies with `feature_of<>`

Imagine that you would like to implement the hypothetical *fubar* statistic, and that you know two ways to calculate fubar on a bunch of samples: an accurate but slow calculation and an approximate but fast calculation. You might opt to make the accurate calculation the default, so you implement two accumulators and call them `impl::fubar_impl` and `impl::fast_fubar_impl`. You would also define the `tag::fubar` and `tag::fast_fubar` features as described [above](#). Now, you would like to inform the Accumulators Framework that these two features are the same from the point of view of dependency resolution. You can do that with `feature_of<>`, as follows:

```
namespace boost { namespace accumulators
{
    // For the purposes of feature-based dependency resolution,
    // fast_fubar provides the same feature as fubar
    template<>
    struct feature_of<tag::fast_fubar>
        : feature_of<tag::fubar>
    {
    };
}
```

The above code instructs the Accumulators Framework that, if another accumulator in the set depends on the `tag::fubar` feature, the `tag::fast_fubar` feature is an acceptable substitute.

### Registering feature variants with `as_feature<>`

You may have noticed that some feature variants in the Accumulators Framework can be specified with a nicer syntax. For instance, instead of `tag::mean` and `tag::immediate_mean` you can specify them with `tag::mean(lazy)` and `tag::mean(immediate)` respectively. These are merely aliases, but the syntax makes the relationship between the two clearer. You can create these feature aliases with the `as_feature<>` trait. Given the fubar example above, you might decide to alias `tag::fubar(accurate)` with `tag::fubar` and `tag::fubar(fast)` with `tag::fast_fubar`. You would do that as follows:

```
namespace boost { namespace accumulators
{
    struct fast {}; // OK to leave these tags empty
    struct accurate {};

    template<>
    struct as_feature<tag::fubar(accurate)>
    {
        typedef tag::fubar type;
    };

    template<>
    struct as_feature<tag::fubar(fast)>
    {
        typedef tag::fast_fubar type;
    };
}}
```

Once you have done this, users of your fubar accumulator can request the `tag::fubar(fast)` and `tag::fubar(accurate)` features when defining their `accumulator_sets` and get the correct accumulator.

## Specializing Numeric Operators

This section describes how to adapt third-party numeric types to work with the Accumulator Framework.

Rather than relying on the built-in operators, the Accumulators Framework relies on functions and operator overloads defined in the [Numeric Operators Sub-Library](#) for many of its numeric operations. This is so that it is possible to assign non-standard meanings to arithmetic operations. For instance, when calculating an average by dividing two integers, the standard integer division behavior would be mathematically incorrect for most statistical quantities. So rather than use `x / y`, the Accumulators Framework uses `numeric::fdiv(x, y)`, which does floating-point division even if both `x` and `y` are integers.

Another example where the Numeric Operators Sub-Library is useful is when a type does not define the operator overloads required to use it for some statistical calculations. For instance, `std::vector<>` does not overload any arithmetic operators, yet it may be useful to use `std::vector<>` as a sample or variate type. The Numeric Operators Sub-Library defines the necessary operator overloads in the `boost::numeric::operators` namespace, which is brought into scope by the Accumulators Framework with a `using` directive.

### Numeric Function Objects and Tag Dispatching

How are the numeric function object defined by the Numeric Operators Sub-Library made to work with types such as `std::vector<>`? The free functions in the `boost::numeric` namespace are implemented in terms of the function objects in the `boost::numeric::functional` namespace, so to make `boost::numeric::fdiv()` do something sensible with a `std::vector<>`, for instance, we'll need to partially specialize the `boost::numeric::functional::fdiv<>` function object.

The functional objects make use of a technique known as *tag dispatching* to select the proper implementation for the given operands. It works as follows:

```
namespace boost { namespace numeric { namespace functional
{
    // Metafunction for looking up the tag associated with
    // a given numeric type T.
    template<typename T>
    struct tag
    {
        // by default, all types have void as a tag type
        typedef void type;
    };

    // Forward declaration looks up the tag types of each operand
    template<
        typename Left
        , typename Right
        , typename LeftTag = typename tag<Left>::type
        , typename RightTag = typename tag<Right>::type
    >
    struct fdiv;
}}}
```

If you have some user-defined type `MyDouble` for which you would like to customize the behavior of `numeric::fdiv()`, you would specialize `numeric::functional::fdiv<>` by first defining a tag type, as shown below:

```
namespace boost { namespace numeric { namespace functional
{
    // Tag type for MyDouble
    struct MyDoubleTag {};

    // Specialize tag<> for MyDouble.
    // This only needs to be done once.
    template<>
    struct tag<MyDouble>
    {
        typedef MyDoubleTag type;
    };

    // Specify how to divide a MyDouble by an integral count
    template<typename Left, typename Right>
    struct fdiv<Left, Right, MyDoubleTag, void>
    {
        // Define the type of the result
        typedef ... result_type;

        result_type operator()(Left & left, Right & right) const
        {
            return ...;
        }
    };
}}}
```

Once you have done this, `numeric::fdiv()` will use your specialization of `numeric::functional::fdiv<>` when the first argument is a `MyDouble` object. All of the function objects in the Numeric Operators Sub-Library can be customized in a similar fashion.

## Concepts

### Accumulator Concept

In the following table, `Acc` is the type of an accumulator, `acc` and `acc2` are objects of type `Acc`, and `args` is the name of an argument pack from the [Boost.Parameter](#) library.

**Table 2. Accumulator Requirements**

Expression	Return type	Assertion / Note / Pre- / Post-condition
<code>Acc::result_type</code>	<i>implementation defined</i>	The type returned by <code>Acc::result()</code> .
<code>Acc acc(args)</code>	none	Construct from an argument pack.
<code>Acc acc(acc2)</code>	none	Post: <code>acc.result(args)</code> is equivalent to <code>acc2.result(args)</code>
<code>acc(args)</code>	<i>unspecified</i>	
<code>acc.on_drop(args)</code>	<i>unspecified</i>	
<code>acc.result(args)</code>	<code>Acc::result_type</code>	

### Feature Concept

In the following table, `F` is the type of a feature and `S` is some scalar type.

**Table 3. Feature Requirements**

Expression	Return type	Assertion / Note / Pre- / Post-condition
<code>F::dependencies</code>	<i>unspecified</i>	An MPL sequence of other features on which <code>F</code> depends.
<code>F::is_weight_accumulator</code>	<code>mpl::true_</code> or <code>mpl::false_</code>	<code>mpl::true_</code> if the accumulator for this feature should be made external when the weight type for the accumulator set is <code>external&lt;S&gt;</code> , <code>mpl::false_</code> otherwise.
<code>F::impl</code>	<i>unspecified</i>	An <a href="#">MPL Lambda Expression</a> that returns the type of the accumulator that implements this feature when passed a sample type and a weight type.

## The Statistical Accumulators Library

The Statistical Accumulators Library defines accumulators for incremental statistical computations. It is built on top of [The Accumulator Framework](#).

## count

The `count` feature is a simple counter that tracks the number of samples pushed into the accumulator set.

Result Type

```
std::size_t
```

Depends On

*none*

Variants

*none*

Initialization Parameters

*none*

Accumulator Parameters

*none*

Extractor Parameters

*none*

Accumulator Complexity

O(1)

Extractor Complexity

O(1)

### Header

```
#include <boost/accumulators/statistics/count.hpp>
```

### Example

```
accumulator_set<int, features<tag::count> > acc;  
acc(0);  
acc(0);  
acc(0);  
assert(3 == count(acc));
```

### See also

- [count\\_impl](#)

## covariance

The `covariance` feature is an iterative Monte Carlo estimator for the covariance. It is specified as `tag::covariance<variate-type, variate-tag>`.

Result Type

```
numeric::functional::outer_product<  
    numeric::functional::fdiv<sample-type, std::size_t>::result_type  
    , numeric::functional::fdiv<variate-type, std::size_t>::result_type  
>::result_type
```

Depends On

count  
mean  
mean\_of\_variates<variate-type, variate-tag>

Variants

abstract\_covariance

Initialization Parameters

*none*

Accumulator Parameters

*variate-tag*

Extractor Parameters	<i>none</i>
Accumulator Complexity	TODO
Extractor Complexity	O(1)

## Headers

```
#include <boost/accumulators/statistics/covariance.hpp>
#include <boost/accumulators/statistics/variates/covariate.hpp>
```

## Example

```
accumulator_set<double, stats<tag::covariance<double, tag::covariate1> > > acc;
acc(1., covariate1 = 2.);
acc(1., covariate1 = 4.);
acc(2., covariate1 = 3.);
acc(6., covariate1 = 1.);
assert(covariance(acc) == -1.75);
```

## See also

- [covariance\\_impl](#)
- [count](#)
- [mean](#)

## density

The `tag::density` feature returns a histogram of the sample distribution. For more implementation details, see [density\\_impl](#).

### Result Type

```
iterator_range<
    std::vector<
        std::pair<
            numeric::functional::fdiv<sample-
type, std::size_t>::result_type
            , numeric::functional::fdiv<sample-
type, std::size_t>::result_type
        >
    >::iterator
>
```

Depends On	count min max
Variants	<i>none</i>
Initialization Parameters	density::cache_size density::num_bins
Accumulator Parameters	<i>none</i>
Extractor Parameters	<i>none</i>
Accumulator Complexity	TODO

Extractor Complexity  $O(N)$ , when  $N$  is `density::num_bins`

### Header

```
#include <boost/accumulators/statistics/density.hpp>
```

### Note

Results from the `density` accumulator can only be extracted after the number of samples meets or exceeds the cache size.

### See also

- [density\\_impl](#)
- [count](#)
- [min](#)
- [max](#)

## error\_of<mean>

The `error_of<mean>` feature calculates the error of the mean feature. It is equal to `sqrt(variance / (count - 1))`.

### Result Type

```
numeric::functional::fdiv<sample-type, std::size_t>::result_type
```

Depends On	<code>count</code> <code>variance</code>
Variants	<code>error_of&lt;immediate_mean&gt;</code>
Initialization Parameters	<i>none</i>
Accumulator Parameters	<i>none</i>
Extractor Parameters	<i>none</i>
Accumulator Complexity	TODO
Extractor Complexity	$O(1)$

### Header

```
#include <boost/accumulators/statistics/error_of.hpp>
#include <boost/accumulators/statistics/error_of_mean.hpp>
```

### Example

```
accumulator_set<double, stats<tag::error_of<tag::mean> > > acc;
acc(1.1);
acc(1.2);
acc(1.3);
assert(0.057735 == error_of<tag::mean>(acc));
```

### See also

- [error\\_of\\_mean\\_impl](#)

- [count](#)
- [variance](#)

## extended\_p\_square

Multiple quantile estimation with the extended P<sup>2</sup> algorithm. For further details, see [extended\\_p\\_square\\_impl](#).

### Result Type

```
boost::iterator_range<
    implementation-defined
>
```

### Depends On

count

### Variants

*none*

### Initialization Parameters

tag::extended\_p\_square::probabilities

### Accumulator Parameters

*none*

### Extractor Parameters

*none*

### Accumulator Complexity

TODO

### Extractor Complexity

O(1)

### Header

```
#include <boost/accumulators/statistics/extended_p_square.hpp>
```

### Example

```
boost::array<double> probs = {0.001,0.01,0.1,0.25,0.5,0.75,0.9,0.99,0.999};
accumulator_set<double, stats<tag::extended_p_square> >
    acc(tag::extended_p_square::probabilities = probs);

boost::lagged_fibonacci607 rng; // a random number generator
for (int i=0; i<10000; ++i)
    acc(rng());

BOOST_CHECK_CLOSE(extended_p_square(acc)[0], probs[0], 25);
BOOST_CHECK_CLOSE(extended_p_square(acc)[1], probs[1], 10);
BOOST_CHECK_CLOSE(extended_p_square(acc)[2], probs[2], 5);

for (std::size_t i=3; i < probs.size(); ++i)
{
    BOOST_CHECK_CLOSE(extended_p_square(acc)[i], probs[i], 2);
}
```

### See also

- [extended\\_p\\_square\\_impl](#)
- [count](#)



## extended\_p\_square\_quantile *and variants*

Quantile estimation using the extended  $P^2$  algorithm for weighted and unweighted samples. By default, the calculation is linear and unweighted, but quadratic and weighted variants are also provided. For further implementation details, see [extended\\_p\\_square\\_quantile\\_impl](#).

All the variants share the `tag::quantile` feature and can be extracted using the `quantile()` extractor.

### Result Type

```
numeric::functional::fdiv<sample-type, std::size_t>::result_type
```

### Depends On

weighted variants depend on `weighted_extended_p_square`  
unweighted variants depend on `extended_p_square`

### Variants

`extended_p_square_quantile_quadratic`  
`weighted_extended_p_square_quantile`  
`weighted_extended_p_square_quantile_quadratic`

### Initialization Parameters

`tag::extended_p_square::probabilities`

### Accumulator Parameters

weight for the weighted variants

### Extractor Parameters

`quantile_probability`

### Accumulator Complexity

TODO

### Extractor Complexity

$O(N)$  where  $N$  is the count of probabilities.

### Header

```
#include <boost/accumulators/statistics/extended_p_square_quantile.hpp>
```

### Example

```
typedef accumulator_set<double, stats<tag::extended_p_square_quantile> >
    accumulator_t;
typedef accumulator_set<double, stats<tag::weighted_extended_p_square_quantile>, double >
    accumulator_t_weighted;
typedef accumulator_set<double, stats<tag::extended_p_square_quantile(quadratic)> >
    accumulator_t_quadratic;
typedef accumulator_set<double, stats<tag::weighted_extended_p_square_quantile(quadratic)
ic)>, double >
    accumulator_t_weighted_quadratic;

// tolerance
double epsilon = 1;

// a random number generator
boost::lagged_fibonacci607 rng;

boost::array<double> probs = { 0.990, 0.991, 0.992, 0.993, 0.994,
                              0.995, 0.996, 0.997, 0.998, 0.999 };
accumulator_t acc(extended_p_square_probabilities = probs);
accumulator_t_weighted acc_weighted(extended_p_square_probabilities = probs);
accumulator_t_quadratic acc2(extended_p_square_probabilities = probs);
accumulator_t_weighted_quadratic acc_weighted2(extended_p_square_probabilities = probs);

for (int i=0; i<10000; ++i)
{
    double sample = rng();
    acc(sample);
    acc2(sample);
    acc_weighted(sample, weight = 1.);
    acc_weighted2(sample, weight = 1.);
}

for (std::size_t i = 0; i < probs.size() - 1; ++i)
{
    BOOST_CHECK_CLOSE(
        quantile(acc, quantile_probability = 0.99025 + i*0.001)
        , 0.99025 + i*0.001
        , epsilon
    );
    BOOST_CHECK_CLOSE(
        quantile(acc2, quantile_probability = 0.99025 + i*0.001)
        , 0.99025 + i*0.001
        , epsilon
    );
    BOOST_CHECK_CLOSE(
        quantile(acc_weighted, quantile_probability = 0.99025 + i*0.001)
        , 0.99025 + i*0.001
        , epsilon
    );
    BOOST_CHECK_CLOSE(
        quantile(acc_weighted2, quantile_probability = 0.99025 + i*0.001)
        , 0.99025 + i*0.001
        , epsilon
    );
}
```

#### See also

- [extended\\_p\\_square\\_quantile\\_impl](#)
- [extended\\_p\\_square](#)
- [weighted\\_extended\\_p\\_square](#)

## kurtosis

The kurtosis of a sample distribution is defined as the ratio of the 4th central moment and the square of the 2nd central moment (the variance) of the samples, minus 3. The term -3 is added in order to ensure that the normal distribution has zero kurtosis. For more implementation details, see [kurtosis\\_impl](#)

### Result Type

```
numeric::functional::fdiv<sample-type, sample-type>::result_type
```

### Depends On

```
mean  
moment<2>  
moment<3>  
moment<4>
```

### Variants

*none*

### Initialization Parameters

*none*

### Accumulator Parameters

*none*

### Extractor Parameters

*none*

### Accumulator Complexity

O(1)

### Extractor Complexity

O(1)

### Header

```
#include <boost/accumulators/statistics/kurtosis.hpp>
```

### Example

```
accumulator_set<int, stats<tag::kurtosis > > acc;  
  
acc(2);  
acc(7);  
acc(4);  
acc(9);  
acc(3);  
  
BOOST_CHECK_EQUAL( mean(acc), 5 );  
BOOST_CHECK_EQUAL( accumulators::moment<2>(acc), 159./5. );  
BOOST_CHECK_EQUAL( accumulators::moment<3>(acc), 1171./5. );  
BOOST_CHECK_EQUAL( accumulators::moment<4>(acc), 1863 );  
BOOST_CHECK_CLOSE( kurtosis(acc), -1.39965397924, 1e-6 );
```

### See also

- [kurtosis\\_impl](#)
- [mean](#)
- [moment](#)

## max

Calculates the maximum value of all the samples.

Result Type

*sample-type*

Depends On

*none*

Variants

*none*

Initialization Parameters

*none*

Accumulator Parameters

*none*

Extractor Parameters

*none*

Accumulator Complexity

O(1)

Extractor Complexity

O(1)

### Header

```
#include <boost/accumulators/statistics/max.hpp>
```

### Example

```
accumulator_set<int, stats<tag::max> > acc;

acc(1);
BOOST_CHECK_EQUAL(1, (max)(acc));

acc(0);
BOOST_CHECK_EQUAL(1, (max)(acc));

acc(2);
BOOST_CHECK_EQUAL(2, (max)(acc));
```

### See also

- [max\\_impl](#)

## mean and variants

Calculates the mean of samples, weights or variates. The calculation is either lazy (in the result extractor), or immediate (in the accumulator). The lazy implementation is the default. For more implementation details, see [mean\\_impl](#) or [immediate\\_mean\\_impl](#)

Result Type

For samples, `numeric::functional::fdiv<sample-type, std::size_t>::result_type`  
For weights, `numeric::functional::fdiv<weight-type, std::size_t>::result_type`  
For variates, `numeric::functional::fdiv<variate-type, std::size_t>::result_type`

Depends On

count  
The lazy mean of samples depends on `sum`  
The lazy mean of weights depends on `sum_of_weights`  
The lazy mean of variates depends on `sum_of_variates<>`

Variants	<code>mean_of_weights</code> <code>mean_of_variates&lt;variate-type, variate-tag&gt;</code> <code>immediate_mean</code> <code>immediate_mean_of_weights</code> <code>immediate_mean_of_variates&lt;variate-type, variate-tag&gt;</code>
Initialization Parameters	<i>none</i>
Accumulator Parameters	<i>none</i>
Extractor Parameters	<i>none</i>
Accumulator Complexity	O(1)
Extractor Complexity	O(1)

**Header**

```
#include <boost/accumulators/statistics/mean.hpp>
```

**Example**

```
accumulator_set<
    int
    , stats<
        tag::mean
        , tag::mean_of_weights
        , tag::mean_of_variates<int, tag::covariate1>
    >
    , int
> acc;

acc(1, weight = 2, covariate1 = 3);
BOOST_CHECK_CLOSE(1., mean(acc), 1e-5);
BOOST_CHECK_EQUAL(1u, count(acc));
BOOST_CHECK_EQUAL(2, sum(acc));
BOOST_CHECK_CLOSE(2., mean_of_weights(acc), 1e-5);
BOOST_CHECK_CLOSE(3., (accumulators::mean_of_variates<int, tag::covariate1>(acc)), 1e-5);

acc(0, weight = 4, covariate1 = 4);
BOOST_CHECK_CLOSE(0.3333333333333333, mean(acc), 1e-5);
BOOST_CHECK_EQUAL(2u, count(acc));
BOOST_CHECK_EQUAL(2, sum(acc));
BOOST_CHECK_CLOSE(3., mean_of_weights(acc), 1e-5);
BOOST_CHECK_CLOSE(3.5, (accumulators::mean_of_variates<int, tag::covariate1>(acc)), 1e-5);

acc(2, weight = 9, covariate1 = 8);
BOOST_CHECK_CLOSE(1.3333333333333333, mean(acc), 1e-5);
BOOST_CHECK_EQUAL(3u, count(acc));
BOOST_CHECK_EQUAL(20, sum(acc));
BOOST_CHECK_CLOSE(5., mean_of_weights(acc), 1e-5);
BOOST_CHECK_CLOSE(5., (accumulators::mean_of_variates<int, tag::covariate1>(acc)), 1e-5);

accumulator_set<
    int
    , stats<
        tag::mean(immediate)
        , tag::mean_of_weights(immediate)
        , tag::mean_of_variates<int, tag::covariate1>(immediate)
    >
    , int
> acc2;

acc2(1, weight = 2, covariate1 = 3);
BOOST_CHECK_CLOSE(1., mean(acc2), 1e-5);
BOOST_CHECK_EQUAL(1u, count(acc2));
BOOST_CHECK_CLOSE(2., mean_of_weights(acc2), 1e-5);
BOOST_CHECK_CLOSE(3., (accumulators::mean_of_variates<int, tag::covariate1>(acc2)), 1e-5);

acc2(0, weight = 4, covariate1 = 4);
BOOST_CHECK_CLOSE(0.3333333333333333, mean(acc2), 1e-5);
BOOST_CHECK_EQUAL(2u, count(acc2));
BOOST_CHECK_CLOSE(3., mean_of_weights(acc2), 1e-5);
BOOST_CHECK_CLOSE(3.5, (accumulators::mean_of_variates<int, tag::covariate1>(acc2)), 1e-5);

acc2(2, weight = 9, covariate1 = 8);
BOOST_CHECK_CLOSE(1.3333333333333333, mean(acc2), 1e-5);
BOOST_CHECK_EQUAL(3u, count(acc2));
BOOST_CHECK_CLOSE(5., mean_of_weights(acc2), 1e-5);
BOOST_CHECK_CLOSE(5., (accumulators::mean_of_variates<int, tag::covariate1>(acc2)), 1e-5);
```

## See also

- [mean\\_impl](#)

- [immediate\\_mean\\_impl](#)
- [count](#)
- [sum](#)

## median and variants

Median estimation based on the  $P^2$  quantile estimator, the density estimator, or the  $P^2$  cumulative distribution estimator. For more implementation details, see [median\\_impl](#), [with\\_density\\_median\\_impl](#), and [with\\_p\\_square\\_cumulative\\_distribution\\_median\\_impl](#).

The three median accumulators all satisfy the `tag::median` feature, and can all be extracted with the `median()` extractor.

### Result Type

```
numeric::functional::fdiv<sample-type, std::size_t>::result_type
```

### Depends On

`median` depends on `p_square_quantile_for_median`  
`with_density_median` depends on `count` and `density`  
`with_p_square_cumulative_distribution_median` depends on `p_square_cumulative_distribution`

### Variants

`with_density_median`  
`with_p_square_cumulative_distribution_median`

### Initialization Parameters

`with_density_median` requires `tag::density::cache_size` and `tag::density::num_bins`  
`with_p_square_cumulative_distribution_median` requires `tag::p_square_cumulative_distribution::num_cells`

### Accumulator Parameters

*none*

### Extractor Parameters

*none*

### Accumulator Complexity

TODO

### Extractor Complexity

TODO

### Header

```
#include <boost/accumulators/statistics/median.hpp>
```

### Example

```
// two random number generators
double mu = 1.;
boost::lagged_fibonacci607 rng;
boost::normal_distribution<> mean_sigma(mu,1);
boost::variate_generator<boost::lagged_fibonacci607&, boost::normal_distribution<> >
    normal(rng, mean_sigma);

accumulator_set<double, stats<tag::median(with_p_square_quantile) > > acc;
accumulator_set<double, stats<tag::median(with_density) > >
    acc_dens( density_cache_size = 10000, density_num_bins = 1000 );
accumulator_set<double, stats<tag::median(with_p_square_cumulative_distribution) > >
    acc_cdist( p_square_cumulative_distribution_num_cells = 100 );

for (std::size_t i=0; i<100000; ++i)
{
    double sample = normal();
    acc(sample);
    acc_dens(sample);
    acc_cdist(sample);
}

BOOST_CHECK_CLOSE(1., median(acc), 1.);
BOOST_CHECK_CLOSE(1., median(acc_dens), 1.);
BOOST_CHECK_CLOSE(1., median(acc_cdist), 3.);
```

#### See also

- [median\\_impl](#)
- [with\\_density\\_median\\_impl](#)
- [with\\_p\\_square\\_cumulative\\_distribution\\_median\\_impl](#)
- [count](#)
- [p\\_square\\_quantile](#)
- [p\\_square\\_cumulative\\_distribution](#)

## min

Calculates the minimum value of all the samples.

Result Type

*sample-type*

Depends On *none*

Variants *none*

Initialization Parameters *none*

Accumulator Parameters *none*

Extractor Parameters *none*

Accumulator Complexity O(1)

Extractor Complexity O(1)

#### Header



```
#include <boost/accumulators/statistics/min.hpp>
```

### Example

```
accumulator_set<int, stats<tag::min> > acc;

acc(1);
BOOST_CHECK_EQUAL(1, (min)(acc));

acc(0);
BOOST_CHECK_EQUAL(0, (min)(acc));

acc(2);
BOOST_CHECK_EQUAL(0, (min)(acc));
```

### See also

- [min\\_impl](#)

## moment

Calculates the N-th moment of the samples, which is defined as the sum of the N-th power of the samples over the count of samples.

#### Result Type

```
numeric::functional::fdiv<sample-type, std::size_t>::result_type
```

Depends On                      *count*

Variants                        *none*

Initialization Parameters      *none*

Accumulator Parameters        *none*

Extractor Parameters          *none*

Accumulator Complexity        O(1)

Extractor Complexity          O(1)

### Header

```
#include <boost/accumulators/statistics/moment.hpp>
```

### Example

```
accumulator_set<int, stats<tag::moment<2> > > acc1;

acc1(2); //    4
acc1(4); //   16
acc1(5); // + 25
        // = 45 / 3 = 15

BOOST_CHECK_CLOSE(15., accumulators::moment<2>(acc1), 1e-5);

accumulator_set<int, stats<tag::moment<5> > > acc2;

acc2(2); //    32
acc2(3); //   243
acc2(4); //  1024
acc2(5); // + 3125
        // = 4424 / 4 = 1106

BOOST_CHECK_CLOSE(1106., accumulators::moment<5>(acc2), 1e-5);
```

### See also

- [moment\\_impl](#)
- [count](#)

## p\_square\_cumulative\_distribution

Histogram calculation of the cumulative distribution with the  $p^2$  algorithm. For more implementation details, see [p\\_square\\_cumulative\\_distribution\\_impl](#)

### Result Type

```
iterator_range<
    std::vector<
        std::pair<
            numeric::functional::fdiv<sample-
type, std::size_t>::result_type
            , numeric::functional::fdiv<sample-
type, std::size_t>::result_type
            >
        >::iterator
    >
```

Depends On	<code>count</code>
Variants	<i>none</i>
Initialization Parameters	<code>tag::p_square_cumulative_distribution::num_cells</code>
Accumulator Parameters	<i>none</i>
Extractor Parameters	<i>none</i>
Accumulator Complexity	TODO
Extractor Complexity	$O(N)$ where $N$ is <code>num_cells</code>

### Header

```
#include <boost/accumulators/statistics/p_square_cumul_dist.hpp>
```

## Example

```
// tolerance in %
double epsilon = 3;

typedef accumulator_set<double, stats<tag::p_square_cumulative_distribution> > accumulator_t;

accumulator_t acc(tag::p_square_cumulative_distribution::num_cells = 100);

// two random number generators
boost::lagged_fibonacci607 rng;
boost::normal_distribution<> mean_sigma(0,1);
boost::variate_generator<boost::lagged_fibonacci607&, boost::normal_distribution<> > normal(rng, mean_sigma);

for (std::size_t i=0; i<100000; ++i)
{
    acc(normal());
}

typedef iterator_range<std::vector<std::pair<double, double> >::iterator > histogram_type;
histogram_type histogram = p_square_cumulative_distribution(acc);

for (std::size_t i = 0; i < histogram.size(); ++i)
{
    // problem with small results: epsilon is relative (in percent), not absolute!
    if ( histogram[i].second > 0.001 )
        BOOST_CHECK_CLOSE( 0.5 * (1.0 + erf( histogram[i].first / sqrt(2.0) )), histogram[i].second, epsilon );
}
```

## See also

- [p\\_square\\_cumulative\\_distribution\\_impl](#)
- [count](#)

## p\_square\_quantile and variants

Single quantile estimation with the  $P^2$  algorithm. For more implementation details, see [p\\_square\\_quantile\\_impl](#)

### Result Type

```
numeric::functional::fdiv<sample-type, std::size_t>::result_type
```

### Depends On

count

### Variants

p\_square\_quantile\_for\_median

### Initialization Parameters

quantile\_probability, which defaults to 0.5. (Note: for p\_square\_quantile\_for\_median, the quantile\_probability parameter is ignored and is always 0.5.)

### Accumulator Parameters

none

### Extractor Parameters

none

### Accumulator Complexity

TODO

### Extractor Complexity

O(1)

## Header

```
#include <boost/accumulators/statistics/p_square_quantile.hpp>
```

### Example

```
typedef accumulator_set<double, stats<tag::p_square_quantile> > accumulator_t;

// tolerance in %
double epsilon = 1;

// a random number generator
boost::lagged_fibonacci607 rng;

accumulator_t acc0(quantile_probability = 0.001);
accumulator_t acc1(quantile_probability = 0.01 );
accumulator_t acc2(quantile_probability = 0.1  );
accumulator_t acc3(quantile_probability = 0.25 );
accumulator_t acc4(quantile_probability = 0.5  );
accumulator_t acc5(quantile_probability = 0.75 );
accumulator_t acc6(quantile_probability = 0.9  );
accumulator_t acc7(quantile_probability = 0.99 );
accumulator_t acc8(quantile_probability = 0.999);

for (int i=0; i<100000; ++i)
{
    double sample = rng();
    acc0(sample);
    acc1(sample);
    acc2(sample);
    acc3(sample);
    acc4(sample);
    acc5(sample);
    acc6(sample);
    acc7(sample);
    acc8(sample);
}

BOOST_CHECK_CLOSE( p_square_quantile(acc0), 0.001, 15*epsilon );
BOOST_CHECK_CLOSE( p_square_quantile(acc1), 0.01 , 5*epsilon );
BOOST_CHECK_CLOSE( p_square_quantile(acc2), 0.1  , epsilon );
BOOST_CHECK_CLOSE( p_square_quantile(acc3), 0.25 , epsilon );
BOOST_CHECK_CLOSE( p_square_quantile(acc4), 0.5  , epsilon );
BOOST_CHECK_CLOSE( p_square_quantile(acc5), 0.75 , epsilon );
BOOST_CHECK_CLOSE( p_square_quantile(acc6), 0.9  , epsilon );
BOOST_CHECK_CLOSE( p_square_quantile(acc7), 0.99 , epsilon );
BOOST_CHECK_CLOSE( p_square_quantile(acc8), 0.999, epsilon );
```

### See also

- [p\\_square\\_quantile\\_impl](#)
- [count](#)

## peaks\_over\_threshold and variants

Peaks Over Threshold method for quantile and tail mean estimation. For implementation details, see [peaks\\_over\\_threshold\\_impl](#) and [peaks\\_over\\_threshold\\_prob\\_impl](#).

Both `tag::peaks_over_threshold` and `tag::peaks_over_threshold_prob<>` satisfy the `tag::abstract_peaks_over_threshold` feature, and can be extracted with the `peaks_over_threshold()` extractor. The result is a 3-tuple representing the fit parameters `u_bar`, `beta_bar` and `xi_hat`.

**Result Type**

```
boost::tuple<
    numeric::functional::fdiv<sample-type, std::size_t>::result_type // u_bar
    , numeric::functional::fdiv<sample-type, std::size_t>::result_type // beta_bar
    , numeric::functional::fdiv<sample-type, std::size_t>::result_type // xi_hat
>
```

**Depends On**`count`

In addition, `tag::peaks_over_threshold_prob<>` depends on `tail<left-or-right>`

**Variants**`peaks_over_threshold_prob<left-or-right>`**Initialization Parameters**

```
tag::peaks_over_threshold::threshold_value
tag::peaks_over_threshold_prob::threshold_probability
tag::tail<left-or-right>::cache_size
```

**Accumulator Parameters***none***Extractor Parameters***none***Accumulator Complexity**

TODO

**Extractor Complexity**

TODO

**Header**

```
#include <boost/accumulators/statistics/peaks_over_threshold.hpp>
```

**Example**

See example for [pot\\_quantile](#).

**See also**

- [peaks\\_over\\_threshold\\_impl](#)
- [peaks\\_over\\_threshold\\_prob\\_impl](#)
- [count](#)
- [tail](#)
- [pot\\_quantile](#)
- [pot\\_tail\\_mean](#)

## **[pot\\_quantile and variants](#)**

Quantile estimation based on Peaks over Threshold method (for both left and right tails). For implementation details, see [pot\\_quantile\\_impl](#).

Both `tag::pot_quantile<left-or-right>` and `tag::pot_quantile_prob<left-or-right>` satisfy the `tag::quantile` feature and can be extracted using the `quantile()` extractor.

**Result Type**

```
numeric::functional::fdiv<sample-type, std::size_t>::result_type
```

**Depends On**

```
pot_quantile<left-or-right> depends on peaks_over_threshold<left-or-right>
pot_quantile_prob<left-or-right> depends on
peaks_over_threshold_prob<left-or-right>
```

**Variants**

```
pot_quantile_prob<left-or-right>
```

**Initialization Parameters**

```
tag::peaks_over_threshold::threshold_value
tag::peaks_over_threshold_prob::threshold_probability
tag::tail<left-or-right>::cache_size
```

**Accumulator Parameters**

```
none
```

**Extractor Parameters**

```
quantile_probability
```

**Accumulator Complexity**

```
TODO
```

**Extractor Complexity**

```
TODO
```

**Header**

```
#include <boost/accumulators/statistics/pot_quantile.hpp>
```

**Example**

```
// tolerance in %
double epsilon = 1.;

double alpha = 0.999;
double threshold_probability = 0.99;
double threshold = 3.;

// two random number generators
boost::lagged_fibonacci607 rng;
boost::normal_distribution<> mean_sigma(0,1);
boost::exponential_distribution<> lambda(1);
boost::variate_generator<boost::lagged_fibonacci607&, boost::normal_distribution<> > normal(rng, mean_sigma);
boost::variate_generator<boost::lagged_fibonacci607&, boost::exponential_distribution<> > exponential(rng, lambda);

accumulator_set<double, stats<tag::pot_quantile<right>(with_threshold_value)> > acc1(
    tag::peaks_over_threshold::threshold_value = threshold
);
accumulator_set<double, stats<tag::pot_quantile<right>(with_threshold_probability)> > acc2(
    tag::tail<right>::cache_size = 2000
    , tag::peaks_over_threshold_prob::threshold_probability = threshold_probability
);

threshold_probability = 0.995;
threshold = 5.;

accumulator_set<double, stats<tag::pot_quantile<right>(with_threshold_value)> > acc3(
    tag::peaks_over_threshold::threshold_value = threshold
);
accumulator_set<double, stats<tag::pot_quantile<right>(with_threshold_probability)> > acc4(
    tag::tail<right>::cache_size = 2000
    , tag::peaks_over_threshold_prob::threshold_probability = threshold_probability
);

for (std::size_t i = 0; i < 100000; ++i)
{
    double sample = normal();
    acc1(sample);
    acc2(sample);
}

for (std::size_t i = 0; i < 100000; ++i)
{
    double sample = exponential();
    acc3(sample);
    acc4(sample);
}

BOOST_CHECK_CLOSE( quantile(acc1, quantile_probability = alpha), 3.090232, epsilon );
BOOST_CHECK_CLOSE( quantile(acc2, quantile_probability = alpha), 3.090232, epsilon );

BOOST_CHECK_CLOSE( quantile(acc3, quantile_probability = alpha), 6.908, epsilon );
BOOST_CHECK_CLOSE( quantile(acc4, quantile_probability = alpha), 6.908, epsilon );
```

## See also

- [pot\\_quantile\\_impl](#)
- [peaks\\_over\\_threshold](#)

## pot\_tail\_mean

Estimation of the (coherent) tail mean based on the peaks over threshold method (for both left and right tails). For implementation details, see [pot\\_tail\\_mean\\_impl](#).

Both `tag::pot_tail_mean<left-or-right>` and `tag::pot_tail_mean_prob<left-or-right>` satisfy the `tag::tail_mean` feature and can be extracted using the `tail_mean()` extractor.

### Result Type

```
numeric::functional::fdiv<sample-type, std::size_t>::result_type
```

### Depends On

`pot_tail_mean<left-or-right>` depends on `peaks_over_threshold<left-or-right>` and `pot_quantile<left-or-right>`  
`pot_tail_mean_prob<left-or-right>` depends on `peaks_over_threshold_prob<left-or-right>` and `pot_quantile_prob<left-or-right>`

### Variants

`pot_tail_mean_prob<left-or-right>`

### Initialization Parameters

`tag::peaks_over_threshold::threshold_value`  
`tag::peaks_over_threshold_prob::threshold_probability`  
`tag::tail<left-or-right>::cache_size`

### Accumulator Parameters

*none*

### Extractor Parameters

`quantile_probability`

### Accumulator Complexity

TODO

### Extractor Complexity

TODO

### Header

```
#include <boost/accumulators/statistics/pot_tail_mean.hpp>
```

### Example

```
// TODO
```

### See also

- [pot\\_tail\\_mean\\_impl](#)
- [peaks\\_over\\_threshold](#)
- [pot\\_quantile](#)

## rolling\_count

The rolling count is the current number of elements in the rolling window.

### Result Type

```
std::size_t
```

### Depends On

`rolling_window_plus1`

### Variants

*none*



Initialization Parameters	<code>tag::rolling_window::window_size</code>
Accumulator Parameters	<i>none</i>
Extractor Parameters	<i>none</i>
Accumulator Complexity	O(1)
Extractor Complexity	O(1)

### Header

```
#include <boost/accumulators/statistics/rolling_count.hpp>
```

### Example

```
accumulator_set<int, stats<tag::rolling_count> > acc(tag::rolling_window::window_size = 3);

BOOST_CHECK_EQUAL(0u, rolling_count(acc));

acc(1);
BOOST_CHECK_EQUAL(1u, rolling_count(acc));

acc(1);
BOOST_CHECK_EQUAL(2u, rolling_count(acc));

acc(1);
BOOST_CHECK_EQUAL(3u, rolling_count(acc));

acc(1);
BOOST_CHECK_EQUAL(3u, rolling_count(acc));

acc(1);
BOOST_CHECK_EQUAL(3u, rolling_count(acc));
```

### See also

- [rolling\\_count\\_impl](#)

## rolling\_sum

The rolling sum is the sum of the last  $N$  samples.

Result Type

```
sample-type
```

Depends On	<code>rolling_window_plus1</code>
Variants	<i>none</i>
Initialization Parameters	<code>tag::rolling_window::window_size</code>
Accumulator Parameters	<i>none</i>
Extractor Parameters	<i>none</i>
Accumulator Complexity	O(1)
Extractor Complexity	O(1)

## Header

```
#include <boost/accumulators/statistics/rolling_sum.hpp>
```

## Example

```
accumulator_set<int, stats<tag::rolling_sum> > acc(tag::rolling_window::window_size = 3);

BOOST_CHECK_EQUAL(0, rolling_sum(acc));

acc(1);
BOOST_CHECK_EQUAL(1, rolling_sum(acc));

acc(2);
BOOST_CHECK_EQUAL(3, rolling_sum(acc));

acc(3);
BOOST_CHECK_EQUAL(6, rolling_sum(acc));

acc(4);
BOOST_CHECK_EQUAL(9, rolling_sum(acc));

acc(5);
BOOST_CHECK_EQUAL(12, rolling_sum(acc));
```

## See also

- [rolling\\_sum\\_impl](#)

## rolling\_mean

The rolling mean is the mean over the last  $N$  samples. It is computed by dividing the rolling sum by the rolling count.

Lazy or iterative calculation of the mean over the last  $N$  samples. The lazy calculation is associated with the `tag::lazy_rolling_mean` feature, and the iterative calculation (which is the default) with the `tag::immediate_rolling_mean` feature. Both can be extracted using the `tag::rolling_mean()` extractor. For more implementation details, see [lazy\\_rolling\\_mean\\_impl](#) and [immediate\\_rolling\\_mean\\_impl](#)

### Result Type

```
numeric::functional::fdiv<sample-type, std::size_t>::result_type
```

### Depends On

`lazy_rolling_mean` depends on `rolling_sum` and `rolling_count`  
`immediate_rolling_mean` depends on `rolling_count`

### Variants

`lazy_rolling_mean` (a.k.a. `rolling_mean(lazy)`)  
`immediate_rolling_mean` (a.k.a. `rolling_mean(immediate)`)

### Initialization Parameters

`tag::rolling_window::window_size`

### Accumulator Parameters

*none*

### Extractor Parameters

*none*

### Accumulator Complexity

O(1)

### Extractor Complexity

O(1)

## Header

```
#include <boost/accumulators/statistics/rolling_mean.hpp>
```

### Example

```
accumulator_set<int, stats<tag::rolling_mean> > acc(tag::rolling_window::window_size = 5);

acc(1);
acc(2);
acc(3);

BOOST_CHECK_CLOSE( rolling_mean(acc), 2.0, 1e-6 );

acc(4);
acc(5);
acc(6);
acc(7);

BOOST_CHECK_CLOSE( rolling_mean(acc), 5.0, 1e-6 );
```

### See also

- [lazy\\_rolling\\_mean\\_impl](#)
- [immediate\\_rolling\\_mean\\_impl](#)
- [rolling\\_count](#)
- [rolling\\_sum](#)

## rolling\_moment

`rolling_moment<M>` calculates the  $M$ -th moment of the samples, which is defined as the sum of the  $M$ -th power of the samples over the count of samples, over the last  $N$  samples.

#### Result Type

```
numeric::functional::fdiv<sample-type, std::size_t>::result_type
```

Depends On	<i>none</i>
Variants	<i>none</i>
Initialization Parameters	<code>tag::rolling_window::window_size</code>
Accumulator Parameters	<i>none</i>
Extractor Parameters	<i>none</i>
Accumulator Complexity	$O(1)$
Extractor Complexity	$O(1)$

### Header

```
#include <boost/accumulators/statistics/rolling_moment.hpp>
```

### Example

```
accumulator_set<int, stats<tag::rolling_moment<2> > > acc(tag::rolling_window::window_size = 3);

acc(2);
acc(4);

BOOST_CHECK_CLOSE( rolling_moment<2>(acc), (4.0 + 16.0)/2, 1e-5 );

acc(5);
acc(6);

BOOST_CHECK_CLOSE( rolling_moment<2>(acc), (16.0 + 25.0 + 36.0)/3, 1e-5 );
```

### See also

- [rolling\\_moment\\_impl](#)

## rolling\_variance

Lazy or iterative calculation of the variance over the last  $N$  samples. The lazy calculation is associated with the `tag::lazy_rolling_variance` feature, and the iterative calculation with the `tag::immediate_rolling_variance` feature. Both can be extracted using the `tag::rolling_variance()` extractor. For more implementation details, see [lazy\\_rolling\\_variance\\_impl](#) and [immediate\\_rolling\\_variance\\_impl](#)

### Result Type

```
numeric::functional::fdiv<sample-type, std::size_t>::result_type
```

### Depends On

`lazy_rolling_variance` depends on `rolling_moment<2>`, `rolling_count` and `rolling_mean`  
`immediate_rolling_variance` depends on `rolling_count` and `immediate_rolling_mean`

### Variants

`lazy_rolling_variance` (a.k.a. `rolling_variance(lazy)`)  
`immediate_rolling_variance` (a.k.a. `rolling_variance(immediate)`)

### Initialization Parameters

`tag::rolling_window::window_size`

### Accumulator Parameters

*none*

### Extractor Parameters

*none*

### Accumulator Complexity

O(1)

### Extractor Complexity

O(1)

### Header

```
#include <boost/accumulators/statistics/rolling_variance.hpp>
```

### Example

```
accumulator_set<double, stats<tag::rolling_variance> > acc(tag::rolling_window::window_size = 4);

acc(1.2);

BOOST_CHECK_CLOSE( rolling_variance(acc), 0.0, 1e-10 ); // variance is not defined for a single ↵
sample

acc(2.3);
acc(3.4);

BOOST_CHECK_CLOSE( rolling_variance(acc), 1.21, 1e-10 ); // variance over samples 1-3

acc(4.5);
acc(0.4);
acc(2.2);
acc(7.1);

BOOST_CHECK_CLOSE( rolling_variance(acc), 8.41666666666667, 1e-10 ); // variance over samples 4-
7
```

### See also

- [lazy\\_rolling\\_variance\\_impl](#)
- [immediate\\_rolling\\_variance\\_impl](#)
- [rolling\\_count](#)
- [rolling\\_mean](#)
- [immediate\\_rolling\\_mean](#)
- [rolling\\_moment](#)

## skewness

The skewness of a sample distribution is defined as the ratio of the 3rd central moment and the 3/2-th power of the 2nd central moment (the variance) of the samples. For implementation details, see [skewness\\_impl](#).

### Result Type

numeric::functional::fdiv<sample-type, sample-type>::result\_type

### Depends On

mean  
moment<2>  
moment<3>

### Variants

*none*

### Initialization Parameters

*none*

### Accumulator Parameters

*none*

### Extractor Parameters

*none*

### Accumulator Complexity

O(1)

### Extractor Complexity

O(1)

### Header

```
#include <boost/accumulators/statistics/skewness.hpp>
```

### Example

```
accumulator_set<int, stats<tag::skewness > > acc2;

acc2(2);
acc2(7);
acc2(4);
acc2(9);
acc2(3);

BOOST_CHECK_EQUAL( mean(acc2), 5 );
BOOST_CHECK_EQUAL( accumulators::moment<2>(acc2), 159./5. );
BOOST_CHECK_EQUAL( accumulators::moment<3>(acc2), 1171./5. );
BOOST_CHECK_CLOSE( skewness(acc2), 0.406040288214, 1e-6 );
```

### See also

- [skewness\\_impl](#)
- [mean](#)
- [moment](#)

## sum and variants

For summing the samples, weights or variates. The default implementation uses the standard sum operation, but variants using the Kahan summation algorithm are also provided.

Result Type	<i>sample-type</i> for summing samples <i>weight-type</i> for summing weights <i>variate-type</i> for summing variates
Depends On	<i>none</i>
Variants	<code>tag::sum</code> <code>tag::sum_of_weights</code> <code>tag::sum_of_variates&lt;variate-type, variate-tag&gt;</code> <code>tag::sum_kahan</code> (a.k.a. <code>tag::sum(kahan)</code> ) <code>tag::sum_of_weights_kahan</code> (a.k.a. <code>tag::sum_of_weights(kahan)</code> ) <code>tag::sum_of_variates_kahan&lt;variate-type, variate-tag&gt;</code>
Initialization Parameters	<i>none</i>
Accumulator Parameters	<i>weight</i> for summing weights <i>variate-tag</i> for summing variates
Extractor Parameters	<i>none</i>
Accumulator Complexity	O(1). Note that the Kahan sum performs four floating-point sum operations per accumulated value, whereas the naive sum performs only one.
Extractor Complexity	O(1)

### Header

```
#include <boost/accumulators/statistics/sum.hpp>
#include <boost/accumulators/statistics/sum_kahan.hpp>
```

## Example

```
accumulator_set<
    int
    , stats<
        tag::sum
        , tag::sum_of_weights
        , tag::sum_of_variates<int, tag::covariate1>
    >
    , int
> acc;

acc(1, weight = 2, covariate1 = 3);
BOOST_CHECK_EQUAL(2, sum(acc)); // weighted sample = 1 * 2
BOOST_CHECK_EQUAL(2, sum_of_weights(acc));
BOOST_CHECK_EQUAL(3, sum_of_variates(acc));

acc(2, weight = 4, covariate1 = 6);
BOOST_CHECK_EQUAL(10, sum(acc)); // weighted sample = 2 * 4
BOOST_CHECK_EQUAL(6, sum_of_weights(acc));
BOOST_CHECK_EQUAL(9, sum_of_variates(acc));

acc(3, weight = 6, covariate1 = 9);
BOOST_CHECK_EQUAL(28, sum(acc)); // weighted sample = 3 * 6
BOOST_CHECK_EQUAL(12, sum_of_weights(acc));
BOOST_CHECK_EQUAL(18, sum_of_variates(acc));

// demonstrate Kahan summation
accumulator_set<float, stats<tag::sum_kahan> > acc;
BOOST_CHECK_EQUAL(0.0f, sum_kahan(acc));
for (size_t i = 0; i < 1e6; ++i) {
    acc(1e-6f);
}
BOOST_CHECK_EQUAL(1.0f, sum_kahan(acc));
```

## See also

- [sum\\_impl](#)
- [sum\\_kahan\\_impl](#)

## tail

Tracks the largest or smallest *N* values. `tag::tail<right>` tracks the largest *N*, and `tag::tail<left>` tracks the smallest. The parameter *N* is specified with the `tag::tail<left-or-right>::cache_size` initialization parameter. For implementation details, see [tail\\_impl](#).

Both `tag::tail<left>` and `tag::tail<right>` satisfy the `tag::abstract_tail` feature and can be extracted with the `tail()` extractor.

## Result Type

```
boost::iterator_range<
    boost::reverse_iterator<
        boost::permutation_iterator<
            std::vector<sample-type>::const_iterator // samples
            , std::vector<std::size_t>::iterator      // indices
        >
    >
>
```

## Depends On

*none*

Variants	<code>abstract_tail</code>
Initialization Parameters	<code>tag::tail&lt;left-or-right&gt;::cache_size</code>
Accumulator Parameters	<i>none</i>
Extractor Parameters	<i>none</i>
Accumulator Complexity	$O(\log N)$ , where $N$ is the cache size
Extractor Complexity	$O(N \log N)$ , where $N$ is the cache size

**Header**

```
#include <boost/accumulators/statistics/tail.hpp>
```

**Example**

See the Example for [tail\\_variate](#).

**See also**

- [tail\\_impl](#)
- [tail\\_variate](#)

## coherent\_tail\_mean

Estimation of the coherent tail mean based on order statistics (for both left and right tails). The left coherent tail mean feature is `tag::coherent_tail_mean<left>`, and the right coherent tail mean feature is `tag::coherent_tail_mean<right>`. They both share the `tag::tail_mean` feature and can be extracted with the `tail_mean()` extractor. For more implementation details, see [coherent\\_tail\\_mean\\_impl](#)

**Result Type**

```
numeric::functional::fdiv<sample-type, std::size_t>::result_type
```

Depends On	<code>count</code> <code>quantile</code> <code>non_coherent_tail_mean&lt;left-or-right&gt;</code>
------------	---

Variants	<i>none</i>
Initialization Parameters	<code>tag::tail&lt;left-or-right&gt;::cache_size</code>
Accumulator Parameters	<i>none</i>
Extractor Parameters	<code>quantile_probability</code>
Accumulator Complexity	$O(\log N)$ , where $N$ is the cache size
Extractor Complexity	$O(N \log N)$ , where $N$ is the cache size

**Header**

```
#include <boost/accumulators/statistics/tail_mean.hpp>
```

**Example**

See the example for [non\\_coherent\\_tail\\_mean](#).



**See also**

- [coherent\\_tail\\_mean\\_impl](#)
- [count](#)
- [extended\\_p\\_square\\_quantile](#)
- [pot\\_quantile](#)
- [tail\\_quantile](#)
- [non\\_coherent\\_tail\\_mean](#)

**non\_coherent\_tail\_mean**

Estimation of the (non-coherent) tail mean based on order statistics (for both left and right tails). The left non-coherent tail mean feature is `tag::non_coherent_tail_mean<left>`, and the right non-coherent tail mean feature is `tag::non_coherent_tail_mean<right>`. They both share the `tag::abstract_non_coherent_tail_mean` feature and can be extracted with the `non_coherent_tail_mean()` extractor. For more implementation details, see [non\\_coherent\\_tail\\_mean\\_impl](#)

**Result Type**

```
numeric::functional::fdiv<sample-type, std::size_t>::result_type
```

**Depends On**

```
count  
tail<left-or-right>
```

**Variants**

```
abstract_non_coherent_tail_mean
```

**Initialization Parameters**

```
tag::tail<left-or-right>::cache_size
```

**Accumulator Parameters**

```
none
```

**Extractor Parameters**

```
quantile_probability
```

**Accumulator Complexity**

```
O(log N), where N is the cache size
```

**Extractor Complexity**

```
O(N log N), where N is the cache size
```

**Header**

```
#include <boost/accumulators/statistics/tail_mean.hpp>
```

**Example**

```
// tolerance in %
double epsilon = 1;

std::size_t n = 100000; // number of MC steps
std::size_t c = 10000; // cache size

typedef accumulator_set<double, stats<tag::non_coher,
ent_tail_mean<right>, tag::tail_quantile<right> > > accumulator_t_right1;
typedef accumulator_set<double, stats<tag::non_coher,
ent_tail_mean<left>, tag::tail_quantile<left> > > accumulator_t_left1;
typedef accumulator_set<double, stats<tag::coher,
ent_tail_mean<right>, tag::tail_quantile<right> > > accumulator_t_right2;
typedef accumulator_set<double, stats<tag::coher,
ent_tail_mean<left>, tag::tail_quantile<left> > > accumulator_t_left2;

accumulator_t_right1 acc0( right_tail_cache_size = c );
accumulator_t_left1 acc1( left_tail_cache_size = c );
accumulator_t_right2 acc2( right_tail_cache_size = c );
accumulator_t_left2 acc3( left_tail_cache_size = c );

// a random number generator
boost::lagged_fibonacci607 rng;

for (std::size_t i = 0; i < n; ++i)
{
    double sample = rng();
    acc0(sample);
    acc1(sample);
    acc2(sample);
    acc3(sample);
}

// check uniform distribution
BOOST_CHECK_CLOSE( non_coherent_tail_mean(acc0, quantile_probability = 0.95), 0.975, epsilon );
BOOST_CHECK_CLOSE( non_coherent_tail_mean(acc0, quantile_probability = 0.975), 0.9875, epsilon );
BOOST_CHECK_CLOSE( non_coherent_tail_mean(acc0, quantile_probability = 0.99), 0.995, epsilon );
BOOST_CHECK_CLOSE( non_coherent_tail_mean(acc0, quantile_probability = 0.999), 0.9995, epsilon );
BOOST_CHECK_CLOSE( non_coherent_tail_mean(acc1, quantile_probability = 0.05), 0.025, epsilon );
BOOST_CHECK_CLOSE( non_coherent_tail_mean(acc1, quantile_probability = 0.025), 0.0125, epsilon );
BOOST_CHECK_CLOSE( non_coherent_tail_mean(acc1, quantile_probability = 0.01), 0.005, 5 );
BOOST_CHECK_CLOSE( non_coherent_tail_mean(acc1, quantile_probability = 0.001), 0.0005, 10 );
BOOST_CHECK_CLOSE( tail_mean(acc2, quantile_probability = 0.95), 0.975, epsilon );
BOOST_CHECK_CLOSE( tail_mean(acc2, quantile_probability = 0.975), 0.9875, epsilon );
BOOST_CHECK_CLOSE( tail_mean(acc2, quantile_probability = 0.99), 0.995, epsilon );
BOOST_CHECK_CLOSE( tail_mean(acc2, quantile_probability = 0.999), 0.9995, epsilon );
BOOST_CHECK_CLOSE( tail_mean(acc3, quantile_probability = 0.05), 0.025, epsilon );
BOOST_CHECK_CLOSE( tail_mean(acc3, quantile_probability = 0.025), 0.0125, epsilon );
BOOST_CHECK_CLOSE( tail_mean(acc3, quantile_probability = 0.01), 0.005, 5 );
BOOST_CHECK_CLOSE( tail_mean(acc3, quantile_probability = 0.001), 0.0005, 10 );
```

### See also

- [non\\_coherent\\_tail\\_mean\\_impl](#)
- [count](#)
- [tail](#)

## tail\_quantile

Tail quantile estimation based on order statistics (for both left and right tails). The left tail quantile feature is `tag::tail_quantile<left>`, and the right tail quantile feature is `tag::tail_quantile<right>`. They both share the

`tag::quantile` feature and can be extracted with the `quantile()` extractor. For more implementation details, see [tail\\_quantile\\_impl](#)

**Result Type**

*sample-type*

**Depends On**

`count`  
`tail<left-or-right>`

**Variants**

*none*

**Initialization Parameters**

`tag::tail<left-or-right>::cache_size`

**Accumulator Parameters**

*none*

**Extractor Parameters**

`quantile_probability`

**Accumulator Complexity**

$O(\log N)$ , where  $N$  is the cache size

**Extractor Complexity**

$O(N \log N)$ , where  $N$  is the cache size

**Header**

```
#include <boost/accumulators/statistics/tail_quantile.hpp>
```

**Example**

```

// tolerance in %
double epsilon = 1;

std::size_t n = 100000; // number of MC steps
std::size_t c = 10000; // cache size

typedef accumulator_set<double, stats<tag::tail_quantile<right> > > accumulator_t_right;
typedef accumulator_set<double, stats<tag::tail_quantile<left> > > accumulator_t_left;

accumulator_t_right acc0( tag::tail<right>::cache_size = c );
accumulator_t_right acc1( tag::tail<right>::cache_size = c );
accumulator_t_left acc2( tag::tail<left>::cache_size = c );
accumulator_t_left acc3( tag::tail<left>::cache_size = c );

// two random number generators
boost::lagged_fibonacci607 rng;
boost::normal_distribution<> mean_sigma(0,1);
boost::variate_generator<boost::lagged_fibonacci607&, boost::normal_distribution<> > normal(
rng, mean_sigma);

for (std::size_t i = 0; i < n; ++i)
{
    double sample1 = rng();
    double sample2 = normal();
    acc0(sample1);
    acc1(sample2);
    acc2(sample1);
    acc3(sample2);
}

// check uniform distribution
BOOST_CHECK_CLOSE( quantile(acc0, quantile_probability = 0.95 ), 0.95, epsilon );
BOOST_CHECK_CLOSE( quantile(acc0, quantile_probability = 0.975), 0.975, epsilon );
BOOST_CHECK_CLOSE( quantile(acc0, quantile_probability = 0.99 ), 0.99, epsilon );
BOOST_CHECK_CLOSE( quantile(acc0, quantile_probability = 0.999), 0.999, epsilon );
BOOST_CHECK_CLOSE( quantile(acc2, quantile_probability = 0.05 ), 0.05, 2 );
BOOST_CHECK_CLOSE( quantile(acc2, quantile_probability = 0.025), 0.025, 2 );
BOOST_CHECK_CLOSE( quantile(acc2, quantile_probability = 0.01 ), 0.01, 3 );
BOOST_CHECK_CLOSE( quantile(acc2, quantile_probability = 0.001), 0.001, 20 );

// check standard normal distribution
BOOST_CHECK_CLOSE( quantile(acc1, quantile_probability = 0.975), 1.959963, epsilon );
BOOST_CHECK_CLOSE( quantile(acc1, quantile_probability = 0.999), 3.090232, epsilon );
BOOST_CHECK_CLOSE( quantile(acc3, quantile_probability = 0.025), -1.959963, epsilon );
BOOST_CHECK_CLOSE( quantile(acc3, quantile_probability = 0.001), -3.090232, epsilon );

```

**See also**

- [tail\\_quantile\\_impl](#)
- [count](#)
- [tail](#)

**tail\_variate**

Tracks the covariates of largest or smallest N samples. `tag::tail_variate<variate-type, variate-tag, right>` tracks the covariate associated with *variate-tag* for the largest N, and `tag::tail_variate<variate-type, variate-tag, left>` for the smallest. The parameter N is specified with the `tag::tail<left-or-right>::cache_size` initialization parameter. For implementation details, see [tail\\_variate\\_impl](#).

Both `tag::tail_variate<variate-type, variate-tag, right>` and `tag::tail_variate<variate-type, variate-tag, left>` satisfy the `tag::abstract_tail_variate` feature and can be extracted with the `tail_variate()` extractor.

### Result Type

```
boost::iterator_range<
    boost::reverse_iterator<
        boost::permutation_iterator<
            std::vector<variate-type>::const_iterator // variates
            , std::vector<std::size_t>::iterator       // indices
        >
    >
>
```

Depends On	<code>tail&lt;left-or-right&gt;</code>
Variants	<code>abstract_tail_variate</code>
Initialization Parameters	<code>tag::tail&lt;left-or-right&gt;::cache_size</code>
Accumulator Parameters	<i>none</i>
Extractor Parameters	<i>none</i>
Accumulator Complexity	$O(\log N)$ , where $N$ is the cache size
Extractor Complexity	$O(N \log N)$ , where $N$ is the cache size

### Header

```
#include <boost/accumulators/statistics/tail_variate.hpp>
```

### Example

```
accumulator_set<int, stats<tag::tail_variate<int, tag::covariate1, right> > > acc(
    tag::tail<right>::cache_size = 4
);

acc(8, covariate1 = 3);
CHECK_RANGE_EQUAL(tail(acc), {8});
CHECK_RANGE_EQUAL(tail_variate(acc), {3});

acc(16, covariate1 = 1);
CHECK_RANGE_EQUAL(tail(acc), {16, 8});
CHECK_RANGE_EQUAL(tail_variate(acc), {1, 3});

acc(12, covariate1 = 4);
CHECK_RANGE_EQUAL(tail(acc), {16, 12, 8});
CHECK_RANGE_EQUAL(tail_variate(acc), {1, 4, 3});

acc(24, covariate1 = 5);
CHECK_RANGE_EQUAL(tail(acc), {24, 16, 12, 8});
CHECK_RANGE_EQUAL(tail_variate(acc), {5, 1, 4, 3});

acc(1, covariate1 = 9);
CHECK_RANGE_EQUAL(tail(acc), {24, 16, 12, 8});
CHECK_RANGE_EQUAL(tail_variate(acc), {5, 1, 4, 3});

acc(9, covariate1 = 7);
CHECK_RANGE_EQUAL(tail(acc), {24, 16, 12, 9});
CHECK_RANGE_EQUAL(tail_variate(acc), {5, 1, 4, 7});
```

**See also**

- [tail\\_variate\\_impl](#)
- [tail](#)

**tail\_variate\_means and variants**

Estimation of the absolute and relative tail variate means (for both left and right tails). The absolute tail variate means has the feature `tag::absolute_tail_variate_means<left-or-right, variate-type, variate-tag>` and the relative tail variate mean has the feature `tag::relative_tail_variate_means<left-or-right, variate-type, variate-tag>`. All absolute tail variate mean features share the `tag::abstract_absolute_tail_variate_means` feature and can be extracted with the `tail_variate_means()` extractor. All the relative tail variate mean features share the `tag::abstract_relative_tail_variate_means` feature and can be extracted with the `relative_tail_variate_means()` extractor.

For more implementation details, see [tail\\_variate\\_means\\_impl](#)

**Result Type**

```
boost::iterator_range<
    std::vector<
        numeric::functional::fdiv<sample-type, std::size_t>::result_type
    >::iterator
>
```

**Depends On**

```
non_coherent_tail_mean<left-or-right>
tail_variate<variate-type, variate-tag, left-or-right>
```

**Variants**

```
tag::absolute_tail_variate_means<left-or-right, variate-type, variate-tag>
tag::relative_tail_variate_means<left-or-right, variate-type, variate-tag>
```

**Initialization Parameters**

```
tag::tail<left-or-right>::cache_size
```

**Accumulator Parameters**

```
none
```

**Extractor Parameters**

```
quantile_probability
```

**Accumulator Complexity**

```
O(log N), where N is the cache size
```

**Extractor Complexity**

```
O(N log N), where N is the cache size
```

**Header**

```
#include <boost/accumulators/statistics/tail_variate_means.hpp>
```

**Example**

```
std::size_t c = 5; // cache size

typedef double variate_type;
typedef std::vector<variate_type> variate_set_type;

typedef accumulator_set<double, stats<
    tag::tail_variate_means<right, variate_set_type, tag::covariatel>(relative)>, tag::tail<right> >
    accumulator_t1;

typedef accumulator_set<double, stats<
    tag::tail_variate_means<right, variate_set_type, tag::covariatel>(absolute)>, tag::tail<right> >
    accumulator_t2;

typedef accumulator_set<double, stats<
    tag::tail_variate_means<left, variate_set_type, tag::covariatel>(relative)>, tag::tail<left> >
    accumulator_t3;

typedef accumulator_set<double, stats<
    tag::tail_variate_means<left, variate_set_type, tag::covariatel>(absolute)>, tag::tail<left> >
    accumulator_t4;

accumulator_t1 acc1( right_tail_cache_size = c );
accumulator_t2 acc2( right_tail_cache_size = c );
accumulator_t3 acc3( left_tail_cache_size = c );
accumulator_t4 acc4( left_tail_cache_size = c );

variate_set_type cov1, cov2, cov3, cov4, cov5;
double c1[] = { 10., 20., 30., 40. }; // 100
double c2[] = { 26., 4., 17., 3. }; // 50
double c3[] = { 46., 64., 40., 50. }; // 200
double c4[] = { 1., 3., 70., 6. }; // 80
double c5[] = { 2., 2., 2., 14. }; // 20
cov1.assign(c1, c1 + sizeof(c1)/sizeof(variate_type));
cov2.assign(c2, c2 + sizeof(c2)/sizeof(variate_type));
cov3.assign(c3, c3 + sizeof(c3)/sizeof(variate_type));
cov4.assign(c4, c4 + sizeof(c4)/sizeof(variate_type));
cov5.assign(c5, c5 + sizeof(c5)/sizeof(variate_type));

acc1(100., covariatel = cov1);
acc1( 50., covariatel = cov2);
acc1(200., covariatel = cov3);
acc1( 80., covariatel = cov4);
acc1( 20., covariatel = cov5);

acc2(100., covariatel = cov1);
acc2( 50., covariatel = cov2);
acc2(200., covariatel = cov3);
acc2( 80., covariatel = cov4);
acc2( 20., covariatel = cov5);

acc3(100., covariatel = cov1);
acc3( 50., covariatel = cov2);
acc3(200., covariatel = cov3);
acc3( 80., covariatel = cov4);
acc3( 20., covariatel = cov5);

acc4(100., covariatel = cov1);
acc4( 50., covariatel = cov2);
acc4(200., covariatel = cov3);
acc4( 80., covariatel = cov4);
acc4( 20., covariatel = cov5);
```

```

// check relative risk contributions
BOOST_CHECK_EQUAL( *(relative_tail_variate_means(acc1, quantile_probability = 0.7).begin()
), 14./75. ); // (10 + 46) / 300 = 14/75
BOOST_CHECK_EQUAL( *(relative_tail_variate_means(acc1, quantile_probability = 0.7).begin() + 1),
7./25. ); // (20 + 64) / 300 = 7/25
BOOST_CHECK_EQUAL( *(relative_tail_variate_means(acc1, quantile_probability = 0.7).begin() + 2),
7./30. ); // (30 + 40) / 300 = 7/30
BOOST_CHECK_EQUAL( *(relative_tail_variate_means(acc1, quantile_probability = 0.7).begin() + 3),
3./10. ); // (40 + 50) / 300 = 3/10
BOOST_CHECK_EQUAL( *(relative_tail_variate_means(acc3, quantile_probability = 0.3).begin()
), 14./35. ); // (26 + 2) / 70 = 14/35
BOOST_CHECK_EQUAL( *(relative_tail_variate_means(acc3, quantile_probability = 0.3).begin() + 1),
3./35. ); // (4 + 2) / 70 = 3/35
BOOST_CHECK_EQUAL( *(relative_tail_variate_means(acc3, quantile_probability = 0.3).begin() + 2),
19./70. ); // (17 + 2) / 70 = 19/70
BOOST_CHECK_EQUAL( *(relative_tail_variate_means(acc3, quantile_probability = 0.3).begin() + 3),
17./70. ); // (3 + 14) / 70 = 17/70

// check absolute risk contributions
BOOST_CHECK_EQUAL( *(tail_variate_means(acc2, quantile_probability = 0.7).begin()
), 28 ); // (10 + 46) / 2 = 28
BOOST_CHECK_EQUAL( *(tail_variate_means(acc2, quantile_probability = 0.7).begin() + 1),
42 ); // (20 + 64) / 2 = 42
BOOST_CHECK_EQUAL( *(tail_variate_means(acc2, quantile_probability = 0.7).begin() + 2),
35 ); // (30 + 40) / 2 = 35
BOOST_CHECK_EQUAL( *(tail_variate_means(acc2, quantile_probability = 0.7).begin() + 3),
45 ); // (40 + 50) / 2 = 45
BOOST_CHECK_EQUAL( *(tail_variate_means(acc4, quantile_probability = 0.3).begin()
), 14 ); // (26 + 2) / 2 = 14
BOOST_CHECK_EQUAL( *(tail_variate_means(acc4, quantile_probability = 0.3).begin() + 1),
3 ); // (4 + 2) / 2 = 3
BOOST_CHECK_EQUAL( *(tail_variate_means(acc4, quantile_probability = 0.3).begin() + 2),
9.5 ); // (17 + 2) / 2 = 9.5
BOOST_CHECK_EQUAL( *(tail_variate_means(acc4, quantile_probability = 0.3).begin() + 3),
8.5 ); // (3 + 14) / 2 = 8.5

// check relative risk contributions
BOOST_CHECK_EQUAL( *(relative_tail_variate_means(acc1, quantile_probability = 0.9).begin()
), 23./100. ); // 46/200 = 23/100
BOOST_CHECK_EQUAL( *(relative_tail_variate_means(acc1, quantile_probability = 0.9).begin() + 1),
8./25. ); // 64/200 = 8/25
BOOST_CHECK_EQUAL( *(relative_tail_variate_means(acc1, quantile_probability = 0.9).begin() + 2),
1./5. ); // 40/200 = 1/5
BOOST_CHECK_EQUAL( *(relative_tail_variate_means(acc1, quantile_probability = 0.9).begin() + 3),
1./4. ); // 50/200 = 1/4
BOOST_CHECK_EQUAL( *(relative_tail_variate_means(acc3, quantile_probability = 0.1).begin()
), 1./10. ); // 2/ 20 = 1/10
BOOST_CHECK_EQUAL( *(relative_tail_variate_means(acc3, quantile_probability = 0.1).begin() + 1),
1./10. ); // 2/ 20 = 1/10
BOOST_CHECK_EQUAL( *(relative_tail_variate_means(acc3, quantile_probability = 0.1).begin() + 2),
1./10. ); // 2/ 20 = 1/10
BOOST_CHECK_EQUAL( *(relative_tail_variate_means(acc3, quantile_probability = 0.1).begin() + 3),
7./10. ); // 14/ 20 = 7/10

// check absolute risk contributions
BOOST_CHECK_EQUAL( *(tail_variate_means(acc2, quantile_probability = 0.9).begin()
), 46 ); // 46
BOOST_CHECK_EQUAL( *(tail_variate_means(acc2, quantile_probability = 0.9).begin() + 1),
64 ); // 64
BOOST_CHECK_EQUAL( *(tail_variate_means(acc2, quantile_probability = 0.9).begin() + 2),
40 ); // 40
BOOST_CHECK_EQUAL( *(tail_variate_means(acc2, quantile_probability = 0.9).begin() + 3),
50 ); // 50
BOOST_CHECK_EQUAL( *(tail_variate_means(acc4, quantile_probability = 0.1).begin()
), 2 ); // 2

```



```
BOOST_CHECK_EQUAL( *(tail_variate_means(acc4, quantile_probability = 0.1).begin() + 1), 2 ); // ↵
2
BOOST_CHECK_EQUAL( *(tail_variate_means(acc4, quantile_probability = 0.1).begin() + 2), 2 ); // ↵
2
BOOST_CHECK_EQUAL( *(tail_variate_means(acc4, quantile_probability = 0.1).begin() + 3), 14 ); // 14
```

### See also

- [tail\\_variate\\_means\\_impl](#)
- [non\\_coherent\\_tail\\_mean](#)
- [tail\\_variate](#)

## variance and variants

Lazy or iterative calculation of the variance. The lazy calculation is associated with the `tag::lazy_variance` feature, and the iterative calculation with the `tag::variance` feature. Both can be extracted using the `tag::variance()` extractor. For more implementation details, see [lazy\\_variance\\_impl](#) and [variance\\_impl](#)

### Result Type

```
numeric::functional::fdiv<sample-type, std::size_t>::result_type
```

### Depends On

`tag::lazy_variance` depends on `tag::moment<2>` and `tag::mean`  
`tag::variance` depends on `tag::count` and `tag::immediate_mean`

### Variants

`tag::lazy_variance` (a.k.a. `tag::variance(lazy)`)  
`tag::variance` (a.k.a. `tag::variance(immediate)`)

### Initialization Parameters

*none*

### Accumulator Parameters

*none*

### Extractor Parameters

*none*

### Accumulator Complexity

O(1)

### Extractor Complexity

O(1)

### Header

```
#include <boost/accumulators/statistics/variance.hpp>
```

### Example

```
// lazy variance
accumulator_set<int, stats<tag::variance(lazy)> > acc1;

acc1(1);
acc1(2);
acc1(3);
acc1(4);
acc1(5);

BOOST_CHECK_EQUAL(5u, count(acc1));
BOOST_CHECK_CLOSE(3., mean(acc1), 1e-5);
BOOST_CHECK_CLOSE(11., accumulators::moment<2>(acc1), 1e-5);
BOOST_CHECK_CLOSE(2., variance(acc1), 1e-5);

// immediate variance
accumulator_set<int, stats<tag::variance> > acc2;

acc2(1);
acc2(2);
acc2(3);
acc2(4);
acc2(5);

BOOST_CHECK_EQUAL(5u, count(acc2));
BOOST_CHECK_CLOSE(3., mean(acc2), 1e-5);
BOOST_CHECK_CLOSE(2., variance(acc2), 1e-5);
```

#### See also

- [lazy\\_variance\\_impl](#)
- [variance\\_impl](#)
- [count](#)
- [mean](#)
- [moment](#)

## weighted\_covariance

An iterative Monte Carlo estimator for the weighted covariance. The feature is specified as `tag::weighted_covariance<variate-type, variate-tag>` and is extracted with the `weighted_variate()` extractor. For more implementation details, see [weighted\\_covariance\\_impl](#)

#### Result Type

```
numeric::functional::outer_product<
    numeric::functional::multiplies<
        weight-type
    , numeric::functional::fdiv<sample-type, std::size_t>::result_type>::result_type
    , numeric::functional::multiplies<
        weight-type
    , numeric::functional::fdiv<variate-type, std::size_t>::result_type>::result_type
>
```

#### Depends On

`count`  
`sum_of_weights`

	<code>weighted_mean</code> <code>weighted_mean_of_variates&lt;variate-type, variate-tag&gt;</code>
Variants	<code>abstract_weighted_covariance</code>
Initialization Parameters	<i>none</i>
Accumulator Parameters	<code>weight</code> <code>variate-tag</code>
Extractor Parameters	<i>none</i>
Accumulator Complexity	$O(1)$
Extractor Complexity	$O(1)$

### Header

```
#include <boost/accumulators/statistics/weighted_covariance.hpp>
```

### Example

```
accumulator_set<double, stats<tag::weighted_covariance<double, tag::covariate1> >, double > acc;  
  
acc(1., weight = 1.1, covariate1 = 2.);  
acc(1., weight = 2.2, covariate1 = 4.);  
acc(2., weight = 3.3, covariate1 = 3.);  
acc(6., weight = 4.4, covariate1 = 1.);  
  
double epsilon = 1e-6;  
BOOST_CHECK_CLOSE(weighted_covariance(acc), -2.39, epsilon);
```

### See also

- [weighted\\_covariance\\_impl](#)
- [count](#)
- [sum](#)
- [weighted\\_mean](#)

## weighted\_density

The `tag::weighted_density` feature returns a histogram of the weighted sample distribution. For more implementation details, see [weighted\\_density\\_impl](#).

### Result Type

```
iterator_range<  
    std::vector<  
        std::pair<  
            numeric::functional::fdiv<weight-  
type, std::size_t>::result_type  
            , numeric::functional::fdiv<weight-  
type, std::size_t>::result_type  
        >  
    >::iterator  
>
```

Depends On `count`

	<code>sum_of_weights</code> <code>min</code> <code>max</code>
Variants	<i>none</i>
Initialization Parameters	<code>tag::weighted_density::cache_size</code> <code>tag::weighted_density::num_bins</code>
Accumulator Parameters	<code>weight</code>
Extractor Parameters	<i>none</i>
Accumulator Complexity	TODO
Extractor Complexity	O(N), when N is <code>weighted_density::num_bins</code>

**Header**

```
#include <boost/accumulators/statistics/weighted_density.hpp>
```

**See also**

- [weighted\\_density\\_impl](#)
- [count](#)
- [sum](#)
- [min](#)
- [max](#)

## weighted\_extended\_p\_square

Multiple quantile estimation with the extended  $P^2$  algorithm for weighted samples. For further details, see [weighted\\_extended\\_p\\_square\\_impl](#).

**Result Type**

```
boost::iterator_range<  
    implementation-defined  
>
```

Depends On	<code>count</code> <code>sum_of_weights</code>
Variants	<i>none</i>
Initialization Parameters	<code>tag::weighted_extended_p_square::probabilities</code>
Accumulator Parameters	<code>weight</code>
Extractor Parameters	<i>none</i>
Accumulator Complexity	TODO
Extractor Complexity	O(1)

**Header**

```
#include <boost/accumulators/statistics/weighted_extended_p_square.hpp>
```

## Example

```
typedef accumulator_set<double, stats<tag::weighted_extended_p_square>, double> accumulator_t;

// tolerance in %
double epsilon = 1;

// some random number generators
double mu1 = -1.0;
double mu2 = 1.0;
boost::lagged_fibonacci607 rng;
boost::normal_distribution<> mean_sigma1(mu1, 1);
boost::normal_distribution<> mean_sigma2(mu2, 1);
boost::variate_generator<boost::lagged_fibonacci607&, boost::normal_distribution<> > nor1(
    rng, mean_sigma1);
boost::variate_generator<boost::lagged_fibonacci607&, boost::normal_distribution<> > nor2(
    rng, mean_sigma2);

std::vector<double> probs_uniform, probs_normal1, probs_normal2, probs_normal_exact1, probs_normal_exact2;

double p1[] = { /*0.001,*/ 0.01, 0.1, 0.5, 0.9, 0.99, 0.999 };
probs_uniform.assign(p1, p1 + sizeof(p1) / sizeof(double));

double p2[] = { 0.001, 0.025 };
double p3[] = { 0.975, 0.999 };
probs_normal1.assign(p2, p2 + sizeof(p2) / sizeof(double));
probs_normal2.assign(p3, p3 + sizeof(p3) / sizeof(double));

double p4[] = { -3.090232, -1.959963 };
double p5[] = { 1.959963, 3.090232 };
probs_normal_exact1.assign(p4, p4 + sizeof(p4) / sizeof(double));
probs_normal_exact2.assign(p5, p5 + sizeof(p5) / sizeof(double));

accumulator_t acc_uniform(tag::weighted_extended_p_square::probabilities = probs_uniform);
accumulator_t acc_normal1(tag::weighted_extended_p_square::probabilities = probs_normal1);
accumulator_t acc_normal2(tag::weighted_extended_p_square::probabilities = probs_normal2);

for (std::size_t i = 0; i < 100000; ++i)
{
    acc_uniform(rng(), weight = 1.);

    double sample1 = normal1();
    double sample2 = normal2();
    acc_normal1(sample1, weight = std::exp(-mu1 * (sample1 - 0.5 * mu1)));
    acc_normal2(sample2, weight = std::exp(-mu2 * (sample2 - 0.5 * mu2)));
}

// check for uniform distribution
for (std::size_t i = 0; i < probs_uniform.size(); ++i)
{
    BOOST_CHECK_CLOSE(weighted_extended_p_square(acc_uniform)[i], probs_uniform[i], epsilon);
}

// check for standard normal distribution
for (std::size_t i = 0; i < probs_normal1.size(); ++i)
{
    BOOST_CHECK_CLOSE(weighted_extended_p_square(acc_normal1)[i], probs_normal_exact1[i], epsilon);
    BOOST_CHECK_CLOSE(weighted_extended_p_square(acc_normal2)[i], probs_normal_exact2[i], epsilon);
}
```

**See also**

- [weighted\\_extended\\_p\\_square\\_impl](#)
- [count](#)
- [sum](#)

## weighted\_kurtosis

The kurtosis of a sample distribution is defined as the ratio of the 4th central moment and the square of the 2nd central moment (the variance) of the samples, minus 3. The term -3 is added in order to ensure that the normal distribution has zero kurtosis. For more implementation details, see [weighted\\_kurtosis\\_impl](#)

**Result Type**

```
numeric::functional::fdiv<
    numeric::functional::multiplies<sample-type, weight-type>::result_type,
    numeric::functional::multiplies<sample-type, weight-type>::result_type>::result_type
```

**Depends On**

```
weighted_mean
weighted_moment<2>
weighted_moment<3>
weighted_moment<4>
```

**Variants**

*none*

**Initialization Parameters**

*none*

**Accumulator Parameters**

*none*

**Extractor Parameters**

*none*

**Accumulator Complexity**

O(1)

**Extractor Complexity**

O(1)

**Header**

```
#include <boost/accumulators/statistics/weighted_kurtosis.hpp>
```

**Example**

```
accumulator_set<int, stats<tag::weighted_kurtosis>, int > acc2;

acc2(2, weight = 4);
acc2(7, weight = 1);
acc2(4, weight = 3);
acc2(9, weight = 1);
acc2(3, weight = 2);

BOOST_CHECK_EQUAL( weighted_mean(acc2), 42./11. );
BOOST_CHECK_EQUAL( accumulators::weighted_moment<2>(acc2), 212./11. );
BOOST_CHECK_EQUAL( accumulators::weighted_moment<3>(acc2), 1350./11. );
BOOST_CHECK_EQUAL( accumulators::weighted_moment<4>(acc2), 9956./11. );
BOOST_CHECK_CLOSE( weighted_kurtosis(acc2), 0.58137026432, 1e-6 );
```

**See also**

- [weighted\\_kurtosis\\_impl](#)
- [weighted\\_mean](#)
- [weighted\\_moment](#)

## weighted\_mean and variants

Calculates the weighted mean of samples or variates. The calculation is either lazy (in the result extractor), or immediate (in the accumulator). The lazy implementation is the default. For more implementation details, see [weighted\\_mean\\_impl](#) or [immediate\\_weighted\\_mean\\_impl](#)

Result Type	For samples, <code>numeric::functional::fdiv&lt;numeric::functional::multiplies&lt;sample-type, weight-type&gt;::result_type, weight-type&gt;::result_type</code> For variates, <code>numeric::functional::fdiv&lt;numeric::functional::multiplies&lt;variate-type, weight-type&gt;::result_type, weight-type&gt;::result_type</code>
Depends On	<code>sum_of_weights</code> The lazy mean of samples depends on <code>weighted_sum</code> The lazy mean of variates depends on <code>weighted_sum_of_variates&lt;&gt;</code>
Variants	<code>weighted_mean_of_variates&lt;variate-type, variate-tag&gt;</code> <code>immediate_weighted_mean</code> <code>immediate_weighted_mean_of_variates&lt;variate-type, variate-tag&gt;</code>
Initialization Parameters	<i>none</i>
Accumulator Parameters	<i>none</i>
Extractor Parameters	<i>none</i>
Accumulator Complexity	O(1)
Extractor Complexity	O(1)

**Header**

```
#include <boost/accumulators/statistics/weighted_mean.hpp>
```

**Example**

```
accumulator_set<
    int
    , stats<
        tag::weighted_mean
        , tag::weighted_mean_of_variates<int, tag::covariate1>
    >
    , int
> acc;

acc(10, weight = 2, covariate1 = 7);           // 20
BOOST_CHECK_EQUAL(2, sum_of_weights(acc));    //
//
acc(6, weight = 3, covariate1 = 8);           // 18
BOOST_CHECK_EQUAL(5, sum_of_weights(acc));    //
//
acc(4, weight = 4, covariate1 = 9);           // 16
BOOST_CHECK_EQUAL(9, sum_of_weights(acc));    //
//
acc(6, weight = 5, covariate1 = 6);           //+ 30
BOOST_CHECK_EQUAL(14, sum_of_weights(acc));   //
// = 84 / 14 = 6

BOOST_CHECK_EQUAL(6., weighted_mean(acc));
BOOST_CHECK_EQUAL(52./7., (accumulators::weighted_mean_of_variates<int, tag::covariate1>(acc)));

accumulator_set<
    int
    , stats<
        tag::weighted_mean(immediate)
        , tag::weighted_mean_of_variates<int, tag::covariate1>(immediate)
    >
    , int
> acc2;

acc2(10, weight = 2, covariate1 = 7);         // 20
BOOST_CHECK_EQUAL(2, sum_of_weights(acc2));  //
//
acc2(6, weight = 3, covariate1 = 8);         // 18
BOOST_CHECK_EQUAL(5, sum_of_weights(acc2));  //
//
acc2(4, weight = 4, covariate1 = 9);         // 16
BOOST_CHECK_EQUAL(9, sum_of_weights(acc2));  //
//
acc2(6, weight = 5, covariate1 = 6);         //+ 30
BOOST_CHECK_EQUAL(14, sum_of_weights(acc2)); //
// = 84 / 14 = 6

BOOST_CHECK_EQUAL(6., weighted_mean(acc2));
BOOST_CHECK_EQUAL(52./7., (accumulators::weighted_mean_of_variates<int, tag::covariate1>(acc2)));
```

## See also

- [weighted\\_mean\\_impl](#)
- [immediate\\_weighted\\_mean\\_impl](#)
- [weighted\\_sum](#)
- [sum](#)



## weighted\_median and variants

Median estimation for weighted samples based on the  $P^2$  quantile estimator, the density estimator, or the  $P^2$  cumulative distribution estimator. For more implementation details, see [weighted\\_median\\_impl](#), [with\\_weighted\\_density\\_median\\_impl](#), and [with\\_weighted\\_p\\_square\\_cumulative\\_distribution\\_median\\_impl](#).

The three median accumulators all satisfy the `tag::weighted_median` feature, and can all be extracted with the `weighted_median()` extractor.

### Result Type

```
numeric::functional::fdiv<sample-type, std::size_t>::result_type
```

### Depends On

`weighted_median` depends on `weighted_p_square_quantile_for_median`  
`with_weighted_density_median` depends on `count` and `weighted_density`  
`with_weighted_p_square_cumulative_distribution_median` depends on `weighted_p_square_cumulative_distribution`

### Variants

`with_weighted_density_median` (a.k.a. `weighted_median(with_weighted_density)`)  
`with_weighted_p_square_cumulative_distribution_median` (a.k.a. `weighted_median(with_weighted_p_square_cumulative_distribution)`)

### Initialization Parameters

`with_weighted_density_median` requires `tag::weighted_density::cache_size` and `tag::weighted_density::num_bins`  
`with_weighted_p_square_cumulative_distribution_median` requires `tag::weighted_p_square_cumulative_distribution::num_cells`

### Accumulator Parameters

`weight`

### Extractor Parameters

*none*

### Accumulator Complexity

TODO

### Extractor Complexity

TODO

### Header

```
#include <boost/accumulators/statistics/weighted_median.hpp>
```

### Example

```
// Median estimation of normal distribution N(1,1) using samples from a narrow normal distribution N(1,0.01)
// The weights equal to the likelihood ratio of the corresponding samples

// two random number generators
double mu = 1.;
double sigma_narrow = 0.01;
double sigma = 1.;
boost::lagged_fibonacci607 rng;
boost::normal_distribution<> mean_sigma_narrow(mu, sigma_narrow);
boost::variate_generator<boost::lagged_fibonacci607&, boost::normal_distribution<> > normal_narrow(rng, mean_sigma_narrow);

accumulator_set<double, stats<tag::weighted_median>> acc(
    with_weighted_p_square_quantile) >, double > acc;
accumulator_set<double, stats<tag::weighted_median>> acc_dens(
    tag::weighted_density::cache_size = 10000, tag::weighted_density::num_bins = 1000) >, double >
accumulator_set<double, stats<tag::weighted_median>> acc_cdist(
    tag::weighted_p_square_cumulative_distribution::num_cells = 100) >, double >

for (std::size_t i=0; i<100000; ++i)
{
    double sample = normal_narrow();
    acc(sample, weight = std::exp(0.5 * (sample - mu) * (sample - mu) * (1./sigma_narrow/sigma_narrow - 1./sigma/sigma)));
    acc_dens(sample, weight = std::exp(0.5 * (sample - mu) * (sample - mu) * (1./sigma_narrow/sigma_narrow - 1./sigma/sigma)));
    acc_cdist(sample, weight = std::exp(0.5 * (sample - mu) * (sample - mu) * (1./sigma_narrow/sigma_narrow - 1./sigma/sigma)));
}

BOOST_CHECK_CLOSE(1., weighted_median(acc), 1e-1);
BOOST_CHECK_CLOSE(1., weighted_median(acc_dens), 1e-1);
BOOST_CHECK_CLOSE(1., weighted_median(acc_cdist), 1e-1);
```

### See also

- [weighted\\_median\\_impl](#)
- [with\\_weighted\\_density\\_median\\_impl](#)
- [with\\_weighted\\_p\\_square\\_cumulative\\_distribution\\_median\\_impl](#)
- [count](#)
- [weighted\\_p\\_square\\_quantile](#)
- [weighted\\_p\\_square\\_cumulative\\_distribution](#)

## weighted\_moment

Calculates the N-th moment of the weighted samples, which is defined as the sum of the weighted N-th power of the samples over the sum of the weights.

### Result Type

```
numeric::functional::fdiv<
    numeric::functional::multiplies<sample-type, weight-type>::result_type,
    weight_type
>::result_type
```

Depends On	count sum_of_weights
Variants	<i>none</i>
Initialization Parameters	<i>none</i>
Accumulator Parameters	weight
Extractor Parameters	<i>none</i>
Accumulator Complexity	O(1)
Extractor Complexity	O(1)

### Header

```
#include <boost/accumulators/statistics/weighted_moment.hpp>
```

### Example

```
accumulator_set<double, stats<tag::weighted_moment<2> >, double> acc2;  
accumulator_set<double, stats<tag::weighted_moment<7> >, double> acc7;  
  
acc2(2.1, weight = 0.7);  
acc2(2.7, weight = 1.4);  
acc2(1.8, weight = 0.9);  
  
acc7(2.1, weight = 0.7);  
acc7(2.7, weight = 1.4);  
acc7(1.8, weight = 0.9);  
  
BOOST_CHECK_CLOSE(5.403, accumulators::weighted_moment<2>(acc2), 1e-5);  
BOOST_CHECK_CLOSE(548.54182, accumulators::weighted_moment<7>(acc7), 1e-5);
```

### See also

- [weighted\\_moment\\_impl](#)
- [count](#)
- [sum](#)

## weighted\_p\_square\_cumulative\_distribution

Histogram calculation of the cumulative distribution with the P<sup>2</sup> algorithm for weighted samples. For more implementation details, see [weighted\\_p\\_square\\_cumulative\\_distribution\\_impl](#)

### Result Type

```
iterator_range<  
    std::vector<  
        std::pair<  
            numeric::function↓  
al::fdiv<weighted_sample, std::size_t>::result_type  
            , numeric::function↓  
al::fdiv<weighted_sample, std::size_t>::result_type  
        >  
    >::iterator  
>
```

where `weighted_sample` is `numeric::functional::multiplies<sample-type, weight-type>::result_type`

Depends On	<code>count</code> <code>sum_or_weights</code>
Variants	<i>none</i>
Initialization Parameters	<code>tag::weighted_p_square_cumulative_distribution::num_cells</code>
Accumulator Parameters	<code>weight</code>
Extractor Parameters	<i>none</i>
Accumulator Complexity	TODO
Extractor Complexity	$O(N)$ where <code>N</code> is <code>num_cells</code>

### Header

```
#include <boost/accumulators/statistics/weighted_p_square_cumul_dist.hpp>
```

### Example

```
// tolerance in %
double epsilon = 4;

typedef accumulator_set<double, stats<tag::weighted_p_square_cumulative_distribution>, double > accu-
accumulator_t;

accumulator_t acc_upper(tag::weighted_p_square_cumulative_distribution::num_cells = 100);
accumulator_t acc_lower(tag::weighted_p_square_cumulative_distribution::num_cells = 100);

// two random number generators
double mu_upper = 1.0;
double mu_lower = -1.0;
boost::lagged_fibonacci607 rng;
boost::normal_distribution<> mean_sigma_upper(mu_upper,1);
boost::normal_distribution<> mean_sigma_lower(mu_lower,1);
boost::variate_generator<boost::lagged_fibonacci607&, boost::normal_distribution<> > normal_up-
per(rng, mean_sigma_upper);
boost::variate_generator<boost::lagged_fibonacci607&, boost::normal_distribution<> > nor-
mal_lower(rng, mean_sigma_lower);

for (std::size_t i=0; i<100000; ++i)
{
    double sample = normal_upper();
    acc_upper(sample, weight = std::exp(-mu_upper * (sample - 0.5 * mu_upper)));
}

for (std::size_t i=0; i<100000; ++i)
{
    double sample = normal_lower();
    acc_lower(sample, weight = std::exp(-mu_lower * (sample - 0.5 * mu_lower)));
}

typedef iterator_range<std::vector<std::pair<double, double> >::iterator > histogram_type;
histogram_type histogram_upper = weighted_p_square_cumulative_distribution(acc_upper);
histogram_type histogram_lower = weighted_p_square_cumulative_distribution(acc_lower);

// Note that applying importance sampling results in a region of the distribution
// to be estimated more accurately and another region to be estimated less accurately
// than without importance sampling, i.e., with unweighted samples

for (std::size_t i = 0; i < histogram_upper.size(); ++i)
{
    // problem with small results: epsilon is relative (in percent), not absolute!

    // check upper region of distribution
    if ( histogram_upper[i].second > 0.1 )
        BOOST_CHECK_CLOSE( 0.5 * (1.0 + erf( histogram_upper[i].first / sqrt(2.0) )), histogram_up-
per[i].second, epsilon );
    // check lower region of distribution
    if ( histogram_lower[i].second < -0.1 )
        BOOST_CHECK_CLOSE( 0.5 * (1.0 + erf( histogram_lower[i].first / sqrt(2.0) )), histo-
gram_lower[i].second, epsilon );
}
```

## See also

- [weighted\\_p\\_square\\_cumulative\\_distribution\\_impl](#)
- [count](#)
- [sum](#)

## weighted\_p\_square\_quantile *and variants*

Single quantile estimation with the  $P^2$  algorithm. For more implementation details, see [weighted\\_p\\_square\\_quantile\\_impl](#)

### Result Type

```
numeric::functional::fdiv<
    numeric::functional::multiplies<sample-type, weight-type>::result_type,
    numeric::functional::multiplies<sample-type, weight-type>::result_type,
    std::size_t
>::result_type
```

### Depends On

count  
sum\_of\_weights

### Variants

weighted\_p\_square\_quantile\_for\_median

### Initialization Parameters

quantile\_probability, which defaults to 0.5. (Note: for weighted\_p\_square\_quantile\_for\_median, the quantile\_probability parameter is ignored and is always 0.5.)

### Accumulator Parameters

weight

### Extractor Parameters

*none*

### Accumulator Complexity

TODO

### Extractor Complexity

O(1)

### Header

```
#include <boost/accumulators/statistics/weighted_p_square_quantile.hpp>
```

### Example

```
typedef accumulator_set<double, stats<tag::weighted_p_square_quantile>, double> accumulator_t;

// tolerance in %
double epsilon = 1;

// some random number generators
double mu4 = -1.0;
double mu5 = -1.0;
double mu6 = 1.0;
double mu7 = 1.0;
boost::lagged_fibonacci607 rng;
boost::normal_distribution<> mean_sigma4(mu4, 1);
boost::normal_distribution<> mean_sigma5(mu5, 1);
boost::normal_distribution<> mean_sigma6(mu6, 1);
boost::normal_distribution<> mean_sigma7(mu7, 1);
boost::variate_generator<boost::lagged_fibonacci607&, boost::normal_distribution<> > norJ
mal4(rng, mean_sigma4);
boost::variate_generator<boost::lagged_fibonacci607&, boost::normal_distribution<> > norJ
mal5(rng, mean_sigma5);
boost::variate_generator<boost::lagged_fibonacci607&, boost::normal_distribution<> > norJ
mal6(rng, mean_sigma6);
boost::variate_generator<boost::lagged_fibonacci607&, boost::normal_distribution<> > norJ
mal7(rng, mean_sigma7);

accumulator_t acc0(quantile_probability = 0.001);
accumulator_t acc1(quantile_probability = 0.025);
accumulator_t acc2(quantile_probability = 0.975);
accumulator_t acc3(quantile_probability = 0.999);

accumulator_t acc4(quantile_probability = 0.001);
accumulator_t acc5(quantile_probability = 0.025);
accumulator_t acc6(quantile_probability = 0.975);
accumulator_t acc7(quantile_probability = 0.999);

for (std::size_t i=0; i<100000; ++i)
{
    double sample = rng();
    acc0(sample, weight = 1.);
    acc1(sample, weight = 1.);
    acc2(sample, weight = 1.);
    acc3(sample, weight = 1.);

    double sample4 = normal4();
    double sample5 = normal5();
    double sample6 = normal6();
    double sample7 = normal7();
    acc4(sample4, weight = std::exp(-mu4 * (sample4 - 0.5 * mu4)));
    acc5(sample5, weight = std::exp(-mu5 * (sample5 - 0.5 * mu5)));
    acc6(sample6, weight = std::exp(-mu6 * (sample6 - 0.5 * mu6)));
    acc7(sample7, weight = std::exp(-mu7 * (sample7 - 0.5 * mu7)));
}

// check for uniform distribution with weight = 1
BOOST_CHECK_CLOSE( weighted_p_square_quantile(acc0), 0.001, 15 );
BOOST_CHECK_CLOSE( weighted_p_square_quantile(acc1), 0.025, 5 );
BOOST_CHECK_CLOSE( weighted_p_square_quantile(acc2), 0.975, epsilon );
BOOST_CHECK_CLOSE( weighted_p_square_quantile(acc3), 0.999, epsilon );
```

```
// check for shifted standard normal distribution ("importance sampling")
BOOST_CHECK_CLOSE( weighted_p_square_quantile(acc4), -3.090232, epsilon );
BOOST_CHECK_CLOSE( weighted_p_square_quantile(acc5), -1.959963, epsilon );
BOOST_CHECK_CLOSE( weighted_p_square_quantile(acc6),  1.959963, epsilon );
BOOST_CHECK_CLOSE( weighted_p_square_quantile(acc7),  3.090232, epsilon );
```

**See also**

- [weighted\\_p\\_square\\_quantile\\_impl](#)
- [count](#)
- [sum](#)

**weighted\_peaks\_over\_threshold and variants**

Weighted peaks over threshold method for weighted quantile and weighted tail mean estimation. For more implementation details, see [weighted\\_peaks\\_over\\_threshold\\_impl](#) and [weighted\\_peaks\\_over\\_threshold\\_prob\\_impl](#).

Both `tag::weighted_peaks_over_threshold<left-or-right>` and `tag::weighted_peaks_over_threshold_prob<left-or-right>` satisfy the `tag::weighted_peaks_over_threshold<left-or-right>` feature and can be extracted using the `weighted_peaks_over_threshold()` extractor.

**Result Type** `tuple<float_type, float_type, float_type>` where `float_type` is

```
numeric::functional::fdiv<
    numeric::functional::multiplies<sample-type, weight-type>::result_type
    , std::size_t
>::result_type
```

**Depends On** `weighted_peaks_over_threshold<left-or-right>` depends on `sum_of_weights`  
`weighted_peaks_over_threshold_prob<left-or-right>` depends on `sum_of_weights` and `tail_weights<left-or-right>`

**Variants** `weighted_peaks_over_threshold_prob`

**Initialization Parameters**  
`tag::peaks_over_threshold::threshold_value`  
`tag::peaks_over_threshold_prob::threshold_probability`  
`tag::tail<left-or-right>::cache_size`

**Accumulator Parameters** `weight`

**Extractor Parameters** `none`

**Accumulator Complexity** `TODO`

**Extractor Complexity** `O(1)`

**Header**

```
#include <boost/accumulators/statistics/weighted_peaks_over_threshold.hpp>
```

**See also**

- [weighted\\_peaks\\_over\\_threshold\\_impl](#)



- [weighted\\_peaks\\_over\\_threshold\\_prob\\_impl](#)
- [sum](#)
- [tail](#)

## weighted\_skewness

The skewness of a sample distribution is defined as the ratio of the 3rd central moment and the  $3/2$ -th power of the 2nd central moment (the variance) of the samples <sup>3</sup>. The skewness estimator for weighted samples is formally identical to the estimator for un-weighted samples, except that the weighted counterparts of all measures it depends on are to be taken.

For implementation details, see [weighted\\_skewness\\_impl](#).

### Result Type

```
numeric::functional::fdiv<
    numeric::functional::multiplies<sample-type, weight-type>::result_type
    , numeric::functional::multiplies<sample-type, weight-type>::result_type
>::result_type
```

### Depends On

```
weighted_mean
weighted_moment<2>
weighted_moment<3>
```

### Variants

*none*

### Initialization Parameters

*none*

### Accumulator Parameters

*weight*

### Extractor Parameters

*none*

### Accumulator Complexity

$O(1)$

### Extractor Complexity

$O(1)$

### Header

```
#include <boost/accumulators/statistics/weighted_skewness.hpp>
```

### Example

```
accumulator_set<int, stats<tag::weighted_skewness>, int > acc2;

acc2(2, weight = 4);
acc2(7, weight = 1);
acc2(4, weight = 3);
acc2(9, weight = 1);
acc2(3, weight = 2);

BOOST_CHECK_EQUAL( weighted_mean(acc2), 42./11. );
BOOST_CHECK_EQUAL( accumulators::weighted_moment<2>(acc2), 212./11. );
BOOST_CHECK_EQUAL( accumulators::weighted_moment<3>(acc2), 1350./11. );
BOOST_CHECK_CLOSE( weighted_skewness(acc2), 1.30708406282, 1e-6 );
```

### See also

- [weighted\\_skewness\\_impl](#)
- [weighted\\_mean](#)
- [weighted\\_moment](#)

## weighted\_sum and variants

For summing the weighted samples or variates. All of the `tag::weighted_sum_of_variates<>` features can be extracted with the `weighted_sum_of_variates()` extractor. Variants that implement the Kahan summation algorithm are also provided.

Result Type	<code>numeric::functional::multiplies&lt;sample-type, weight-type&gt;::result_type</code> for summing weighted samples <code>numeric::functional::multiplies&lt;variate-type, weight-type&gt;::result_type</code> for summing weighted variates
Depends On	<i>none</i>
Variants	<code>tag::weighted_sum</code> <code>tag::weighted_sum_of_variates&lt;variate-type, variate-tag&gt;</code> <code>tag::weighted_sum_kahan</code> (a.k.a. <code>tag::weighted_sum(kahan)</code> ) <code>tag::weighted_sum_of_variates_kahan&lt;variate-type, variate-tag&gt;</code>
Initialization Parameters	<i>none</i>
Accumulator Parameters	<code>weight</code> <code>variate-tag</code> for summing variates
Extractor Parameters	<i>none</i>
Accumulator Complexity	O(1). Note that the Kahan sum performs four floating-point sum operations per accumulated value, whereas the naive sum performs only one.
Extractor Complexity	O(1)

### Header

```
#include <boost/accumulators/statistics/weighted_sum.hpp>
#include <boost/accumulators/statistics/weighted_sum_kahan.hpp>
```

### Example

```
accumulator_set<int, stats<tag::weighted_sum, tag::weighted_sum_of_variates<int, tag::covariate1> >, int> acc;

acc(1, weight = 2, covariate1 = 3);
BOOST_CHECK_EQUAL(2, weighted_sum(acc));
BOOST_CHECK_EQUAL(6, weighted_sum_of_variates(acc));

acc(2, weight = 3, covariate1 = 6);
BOOST_CHECK_EQUAL(8, weighted_sum(acc));
BOOST_CHECK_EQUAL(24, weighted_sum_of_variates(acc));

acc(4, weight = 6, covariate1 = 9);
BOOST_CHECK_EQUAL(32, weighted_sum(acc));
BOOST_CHECK_EQUAL(78, weighted_sum_of_variates(acc));

// demonstrate weighted Kahan summation
accumulator_set<float, stats<tag::weighted_sum_kahan>, float > acc;
BOOST_CHECK_EQUAL(0.0f, weighted_sum_kahan(acc));
for (size_t i = 0; i < 1e6; ++i) {
    acc(1.0f, weight = 1e-6f);
}
BOOST_CHECK_EQUAL(1.0f, weighted_sum_kahan(acc));
```

#### See also

- [weighted\\_sum\\_impl](#)
- [weighted\\_sum\\_kahan\\_impl](#)

## non\_coherent\_weighted\_tail\_mean

Estimation of the (non-coherent) weighted tail mean based on order statistics (for both left and right tails). The left non-coherent weighted tail mean feature is `tag::non_coherent_weighted_tail_mean<left>`, and the right non-coherent weighted tail mean feature is `tag::non_coherent_weighted_tail_mean<right>`. They both share the `tag::abstract_non_coherent_tail_mean` feature with the unweighted non-coherent tail mean accumulators and can be extracted with either the `non_coherent_tail_mean()` or the `non_coherent_weighted_tail_mean()` extractors. For more implementation details, see [non\\_coherent\\_weighted\\_tail\\_mean\\_impl](#).

#### Result Type

```
numeric::functional::fdiv<
    numeric::functional::multiplies<sample-type, weight-type>::result_type
    , std::size_t
>::result_type
```

#### Depends On

`sum_of_weights`  
`tail_weights<left-or-right>`

#### Variants

`abstract_non_coherent_tail_mean`

#### Initialization Parameters

`tag::tail<left-or-right>::cache_size`

#### Accumulator Parameters

*none*

#### Extractor Parameters

`quantile_probability`

#### Accumulator Complexity

$O(\log N)$ , where  $N$  is the cache size

#### Extractor Complexity

$O(N \log N)$ , where  $N$  is the cache size

## Header

```
#include <boost/accumulators/statistics/weighted_tail_mean.hpp>
```

## Example

```
// tolerance in %
double epsilon = 1;

std::size_t n = 100000; // number of MC steps
std::size_t c = 25000; // cache size

accumulator_set<double, stats<tag::non_coherent_weighted_tail_mean<right> >, double >
    acc0( right_tail_cache_size = c );
accumulator_set<double, stats<tag::non_coherent_weighted_tail_mean<left> >, double >
    acc1( left_tail_cache_size = c );

// random number generators
boost::lagged_fibonacci607 rng;

for (std::size_t i = 0; i < n; ++i)
{
    double smpl = std::sqrt(rng());
    acc0(smpl, weight = 1./smpl);
}

for (std::size_t i = 0; i < n; ++i)
{
    double smpl = rng();
    acc1(smpl*smpl, weight = smpl);
}

// check uniform distribution
BOOST_CHECK_CLOSE( non_coherent_weighted_tail_mean(acc0, quantile_probability = 0.95), 0.975, epsilon );
BOOST_CHECK_CLOSE( non_coherent_weighted_tail_mean(acc0, quantile_probability = 0.975), 0.9875, epsilon );
BOOST_CHECK_CLOSE( non_coherent_weighted_tail_mean(acc0, quantile_probability = 0.99), 0.995, epsilon );
BOOST_CHECK_CLOSE( non_coherent_weighted_tail_mean(acc0, quantile_probability = 0.999), 0.9995, epsilon );
BOOST_CHECK_CLOSE( non_coherent_weighted_tail_mean(acc1, quantile_probability = 0.05), 0.025, epsilon );
BOOST_CHECK_CLOSE( non_coherent_weighted_tail_mean(acc1, quantile_probability = 0.025), 0.0125, epsilon );
BOOST_CHECK_CLOSE( non_coherent_weighted_tail_mean(acc1, quantile_probability = 0.01), 0.005, epsilon );
BOOST_CHECK_CLOSE( non_coherent_weighted_tail_mean(acc1, quantile_probability = 0.001), 0.0005, 5*epsilon );
```

## See also

- [non\\_coherent\\_weighted\\_tail\\_mean\\_impl](#)
- [sum](#)
- [tail](#)

## weighted\_tail\_quantile

Tail quantile estimation based on order statistics of weighted samples (for both left and right tails). The left weighted tail quantile feature is `tag::weighted_tail_quantile<left>`, and the right weighted tail quantile feature is `tag::weighted_tail_quantile<right>`. They both share the `tag::quantile` feature with the unweighted tail quantile accumulators and can be extracted with either the `quantile()` or the `weighted_tail_quantile()` extractors. For more implementation details, see [weighted\\_tail\\_quantile\\_impl](#)

### Result Type

`sample-type`

### Depends On

`sum_of_weights`  
`tail_weights<left-or-right>`

### Variants

*none*

### Initialization Parameters

`tag::tail<left-or-right>::cache_size`

### Accumulator Parameters

*none*

### Extractor Parameters

`quantile_probability`

### Accumulator Complexity

$O(\log N)$ , where  $N$  is the cache size

### Extractor Complexity

$O(N \log N)$ , where  $N$  is the cache size

### Header

```
#include <boost/accumulators/statistics/weighted_tail_quantile.hpp>
```

### Example

```
// tolerance in %
double epsilon = 1;

std::size_t n = 100000; // number of MC steps
std::size_t c = 20000; // cache size

double mu1 = 1.0;
double mu2 = -1.0;
boost::lagged_fibonacci607 rng;
boost::normal_distribution<> mean_signal(mu1,1);
boost::normal_distribution<> mean_sigma2(mu2,1);
boost::variate_generator<boost::lagged_fibonacci607&, boost::normal_distribution<> > nor1(
    rng, mean_signal);
boost::variate_generator<boost::lagged_fibonacci607&, boost::normal_distribution<> > nor2(
    rng, mean_sigma2);

accumulator_set<double, stats<tag::weighted_tail_quantile<right> >, double>
    acc1(right_tail_cache_size = c);

accumulator_set<double, stats<tag::weighted_tail_quantile<left> >, double>
    acc2(left_tail_cache_size = c);

for (std::size_t i = 0; i < n; ++i)
{
    double sample1 = nor1();
    double sample2 = nor2();
    acc1(sample1, weight = std::exp(-mu1 * (sample1 - 0.5 * mu1)));
    acc2(sample2, weight = std::exp(-mu2 * (sample2 - 0.5 * mu2)));
}

// check standard normal distribution
BOOST_CHECK_CLOSE( quantile(acc1, quantile_probability = 0.975), 1.959963, epsilon );
BOOST_CHECK_CLOSE( quantile(acc1, quantile_probability = 0.999), 3.090232, epsilon );
BOOST_CHECK_CLOSE( quantile(acc2, quantile_probability = 0.025), -1.959963, epsilon );
BOOST_CHECK_CLOSE( quantile(acc2, quantile_probability = 0.001), -3.090232, epsilon );
```

### See also

- [weighted\\_tail\\_quantile\\_impl](#)
- [sum](#)
- [tail](#)

## weighted\_tail\_variate\_means and variants

Estimation of the absolute and relative weighted tail variate means (for both left and right tails) The absolute weighted tail variate means has the feature `tag::absolute_weighted_tail_variate_means<left-or-right, variate-type, variate-tag>` and the relative weighted tail variate mean has the feature `tag::relative_weighted_tail_variate_means<left-or-right, variate-type, variate-tag>`. All absolute weighted tail variate mean features share the `tag::abstract_absolute_tail_variate_means` feature with their unweighted variants and can be extracted with the `tail_variate_means()` and `weighted_tail_variate_means()` extractors. All the relative weighted tail variate mean features share the `tag::abstract_relative_tail_variate_means` feature with their unweighted variants and can be extracted with either the `relative_tail_variate_means()` or `relative_weighted_tail_variate_means()` extractors.

For more implementation details, see [weighted\\_tail\\_variate\\_means\\_impl](#)

**Result Type**

```
boost::iterator_range<
    numeric::functional::fdiv<
        numeric::functional::multiplies<variate-type, weight-
type>::result_type
        , weight-type
    >::result_type::iterator
>
```

**Depends On**

```
non_coherent_weighted_tail_mean<left-or-right>
tail_variate<variate-type, variate-tag, left-or-right>
tail_weights<left-or-right>
```

**Variants**

```
tag::absolute_weighted_tail_variate_means<left-or-right, variate-type,
variate-tag>
tag::relative_weighted_tail_variate_means<left-or-right, variate-type,
variate-tag>
```

**Initialization Parameters**

```
tag::tail<left-or-right>::cache_size
```

**Accumulator Parameters**

```
none
```

**Extractor Parameters**

```
quantile_probability
```

**Accumulator Complexity**

```
O(log N), where N is the cache size
```

**Extractor Complexity**

```
O(N log N), where N is the cache size
```

**Header**

```
#include <boost/accumulators/statistics/weighted_tail_variate_means.hpp>
```

**Example**

```

std::size_t c = 5; // cache size

typedef double variate_type;
typedef std::vector<variate_type> variate_set_type;

accumulator_set<double, stats<tag::weighted_tail_variate_means<right, variate_set_type, tag::covariatel>(relative)>, double >
    acc1( right_tail_cache_size = c );
accumulator_set<double, stats<tag::weighted_tail_variate_means<right, variate_set_type, tag::covariatel>(absolute)>, double >
    acc2( right_tail_cache_size = c );
accumulator_set<double, stats<tag::weighted_tail_variate_means<left, variate_set_type, tag::covariatel>(relative)>, double >
    acc3( left_tail_cache_size = c );
accumulator_set<double, stats<tag::weighted_tail_variate_means<left, variate_set_type, tag::covariatel>(absolute)>, double >
    acc4( left_tail_cache_size = c );

variate_set_type cov1, cov2, cov3, cov4, cov5;
double c1[] = { 10., 20., 30., 40. }; // 100
double c2[] = { 26., 4., 17., 3. }; // 50
double c3[] = { 46., 64., 40., 50. }; // 200
double c4[] = { 1., 3., 70., 6. }; // 80
double c5[] = { 2., 2., 2., 14. }; // 20
cov1.assign(c1, c1 + sizeof(c1)/sizeof(variate_type));
cov2.assign(c2, c2 + sizeof(c2)/sizeof(variate_type));
cov3.assign(c3, c3 + sizeof(c3)/sizeof(variate_type));
cov4.assign(c4, c4 + sizeof(c4)/sizeof(variate_type));
cov5.assign(c5, c5 + sizeof(c5)/sizeof(variate_type));

acc1(100., weight = 0.8, covariatel = cov1);
acc1( 50., weight = 0.9, covariatel = cov2);
acc1(200., weight = 1.0, covariatel = cov3);
acc1( 80., weight = 1.1, covariatel = cov4);
acc1( 20., weight = 1.2, covariatel = cov5);

acc2(100., weight = 0.8, covariatel = cov1);
acc2( 50., weight = 0.9, covariatel = cov2);
acc2(200., weight = 1.0, covariatel = cov3);
acc2( 80., weight = 1.1, covariatel = cov4);
acc2( 20., weight = 1.2, covariatel = cov5);

acc3(100., weight = 0.8, covariatel = cov1);
acc3( 50., weight = 0.9, covariatel = cov2);
acc3(200., weight = 1.0, covariatel = cov3);
acc3( 80., weight = 1.1, covariatel = cov4);
acc3( 20., weight = 1.2, covariatel = cov5);

acc4(100., weight = 0.8, covariatel = cov1);
acc4( 50., weight = 0.9, covariatel = cov2);
acc4(200., weight = 1.0, covariatel = cov3);
acc4( 80., weight = 1.1, covariatel = cov4);
acc4( 20., weight = 1.2, covariatel = cov5);

// check relative risk contributions
BOOST_CHECK_EQUAL( *(relative_weighted_tail_variate_means(acc1, quantile_probability = 0.7).begin()
    ), (0.8*10 + 1.0*46)/(0.8*100 + 1.0*200) );
BOOST_CHECK_EQUAL( *(relative_weighted_tail_variate_means(acc1, quantile_probability = 0.7).begin()
    + 1), (0.8*20 + 1.0*64)/(0.8*100 + 1.0*200) );
BOOST_CHECK_EQUAL( *(relative_weighted_tail_variate_means(acc1, quantile_probability = 0.7).begin()
    + 2), (0.8*30 + 1.0*40)/(0.8*100 + 1.0*200) );
BOOST_CHECK_EQUAL( *(relative_weighted_tail_variate_means(acc1, quantile_probability = 0.7).begin()
    + 3), (0.8*40 + 1.0*50)/(0.8*100 + 1.0*200) );

```



```

BOOST_CHECK_EQUAL( *(relative_weighted_tail_variate_means(acc3, quantile_probability = 0.3).begin()
gin()      ), (0.9*26 + 1.2*2)/(0.9*50 + 1.2*20) );
BOOST_CHECK_EQUAL( *(relative_weighted_tail_variate_means(acc3, quantile_probability = 0.3).begin()
gin() + 1), (0.9*4 + 1.2*2)/(0.9*50 + 1.2*20) );
BOOST_CHECK_EQUAL( *(relative_weighted_tail_variate_means(acc3, quantile_probability = 0.3).begin()
gin() + 2), (0.9*17 + 1.2*2)/(0.9*50 + 1.2*20) );
BOOST_CHECK_EQUAL( *(relative_weighted_tail_variate_means(acc3, quantile_probability = 0.3).begin()
gin() + 3), (0.9*3 + 1.2*14)/(0.9*50 + 1.2*20) );

// check absolute risk contributions
BOOST_CHECK_EQUAL( *(weighted_tail_variate_means(acc2, quantile_probability = 0.7).begin()
), (0.8*10 + 1.0*46)/1.8 );
BOOST_CHECK_EQUAL( *(weighted_tail_variate_means(acc2, quantile_probability = 0.7).begin()
gin() + 1), (0.8*20 + 1.0*64)/1.8 );
BOOST_CHECK_EQUAL( *(weighted_tail_variate_means(acc2, quantile_probability = 0.7).begin()
gin() + 2), (0.8*30 + 1.0*40)/1.8 );
BOOST_CHECK_EQUAL( *(weighted_tail_variate_means(acc2, quantile_probability = 0.7).begin()
gin() + 3), (0.8*40 + 1.0*50)/1.8 );
BOOST_CHECK_EQUAL( *(weighted_tail_variate_means(acc4, quantile_probability = 0.3).begin()
), (0.9*26 + 1.2*2)/2.1 );
BOOST_CHECK_EQUAL( *(weighted_tail_variate_means(acc4, quantile_probability = 0.3).begin()
gin() + 1), (0.9*4 + 1.2*2)/2.1 );
BOOST_CHECK_EQUAL( *(weighted_tail_variate_means(acc4, quantile_probability = 0.3).begin()
gin() + 2), (0.9*17 + 1.2*2)/2.1 );
BOOST_CHECK_EQUAL( *(weighted_tail_variate_means(acc4, quantile_probability = 0.3).begin()
gin() + 3), (0.9*3 + 1.2*14)/2.1 );

// check relative risk contributions
BOOST_CHECK_EQUAL( *(relative_weighted_tail_variate_means(acc1, quantile_probability = 0.9).begin()
gin()      ), 1.0*46/(1.0*200) );
BOOST_CHECK_EQUAL( *(relative_weighted_tail_variate_means(acc1, quantile_probability = 0.9).begin()
gin() + 1), 1.0*64/(1.0*200) );
BOOST_CHECK_EQUAL( *(relative_weighted_tail_variate_means(acc1, quantile_probability = 0.9).begin()
gin() + 2), 1.0*40/(1.0*200) );
BOOST_CHECK_EQUAL( *(relative_weighted_tail_variate_means(acc1, quantile_probability = 0.9).begin()
gin() + 3), 1.0*50/(1.0*200) );
BOOST_CHECK_EQUAL( *(relative_weighted_tail_variate_means(acc3, quantile_probability = 0.1).begin()
gin()      ), 1.2*2/(1.2*20) );
BOOST_CHECK_EQUAL( *(relative_weighted_tail_variate_means(acc3, quantile_probability = 0.1).begin()
gin() + 1), 1.2*2/(1.2*20) );
BOOST_CHECK_EQUAL( *(relative_weighted_tail_variate_means(acc3, quantile_probability = 0.1).begin()
gin() + 2), 1.2*2/(1.2*20) );
BOOST_CHECK_EQUAL( *(relative_weighted_tail_variate_means(acc3, quantile_probability = 0.1).begin()
gin() + 3), 1.2*14/(1.2*20) );

// check absolute risk contributions
BOOST_CHECK_EQUAL( *(weighted_tail_variate_means(acc2, quantile_probability = 0.9).begin()
), 1.0*46/1.0 );
BOOST_CHECK_EQUAL( *(weighted_tail_variate_means(acc2, quantile_probability = 0.9).begin()
gin() + 1), 1.0*64/1.0 );
BOOST_CHECK_EQUAL( *(weighted_tail_variate_means(acc2, quantile_probability = 0.9).begin()
gin() + 2), 1.0*40/1.0 );
BOOST_CHECK_EQUAL( *(weighted_tail_variate_means(acc2, quantile_probability = 0.9).begin()
gin() + 3), 1.0*50/1.0 );
BOOST_CHECK_EQUAL( *(weighted_tail_variate_means(acc4, quantile_probability = 0.1).begin()
), 1.2*2/1.2 );
BOOST_CHECK_EQUAL( *(weighted_tail_variate_means(acc4, quantile_probability = 0.1).begin()
gin() + 1), 1.2*2/1.2 );
BOOST_CHECK_EQUAL( *(weighted_tail_variate_means(acc4, quantile_probability = 0.1).begin()
gin() + 2), 1.2*2/1.2 );
BOOST_CHECK_EQUAL( *(weighted_tail_variate_means(acc4, quantile_probability = 0.1).begin()
gin() + 3), 1.2*14/1.2 );

```

**See also**

- [weighted\\_tail\\_variate\\_means\\_impl](#)
- [non\\_coherent\\_weighted\\_tail\\_mean](#)
- [tail\\_variate](#)
- [tail](#)

## weighted\_variance and variants

Lazy or iterative calculation of the weighted variance. The lazy calculation is associated with the `tag::lazy_weighted_variance` feature, and the iterative calculation with the `tag::weighted_variance` feature. Both can be extracted using the `tag::weighted_variance()` extractor. For more implementation details, see [lazy\\_weighted\\_variance\\_impl](#) and [weighted\\_variance\\_impl](#)

**Result Type**

```
numeric::functional::fdiv<
    numeric::functional::multiplies<sample-type, weight-type>::result_type
    , std::size_t
>::result_type
```

**Depends On**

`tag::lazy_weighted_variance` depends on `tag::weighted_moment<2>` and `tag::weighted_mean`  
`tag::weighted_variance` depends on `tag::count` and `tag::immediate_weighted_mean`

**Variants**

`tag::lazy_weighted_variance` (a.k.a. `tag::weighted_variance(lazy)`)  
`tag::weighted_variance` (a.k.a. `tag::weighted_variance(immediate)`)

**Initialization Parameters**

*none*

**Accumulator Parameters**

`weight`

**Extractor Parameters**

*none*

**Accumulator Complexity**

O(1)

**Extractor Complexity**

O(1)

**Header**

```
#include <boost/accumulators/statistics/weighted_variance.hpp>
```

**Example**

```
// lazy weighted_variance
accumulator_set<int, stats<tag::weighted_variance(lazy)>, int> acc1;

acc1(1, weight = 2);    // 2
acc1(2, weight = 3);    // 6
acc1(3, weight = 1);    // 3
acc1(4, weight = 4);    // 16
acc1(5, weight = 1);    // 5

// weighted_mean = (2+6+3+16+5) / (2+3+1+4+1) = 32 / 11 = 2.9090909090909090909090909090909
BOOST_CHECK_EQUAL(5u, count(acc1));
BOOST_CHECK_CLOSE(2.9090909, weighted_mean(acc1), 1e-5);
BOOST_CHECK_CLOSE(10.1818182, accumulators::weighted_moment<2>(acc1), 1e-5);
BOOST_CHECK_CLOSE(1.7190083, weighted_variance(acc1), 1e-5);

// immediate weighted_variance
accumulator_set<int, stats<tag::weighted_variance>, int> acc2;

acc2(1, weight = 2);
acc2(2, weight = 3);
acc2(3, weight = 1);
acc2(4, weight = 4);
acc2(5, weight = 1);

BOOST_CHECK_EQUAL(5u, count(acc2));
BOOST_CHECK_CLOSE(2.9090909, weighted_mean(acc2), 1e-5);
BOOST_CHECK_CLOSE(1.7190083, weighted_variance(acc2), 1e-5);

// check lazy and immediate variance with random numbers

// two random number generators
boost::lagged_fibonacci607 rng;
boost::normal_distribution<> mean_sigma(0,1);
boost::variate_generator<boost::lagged_fibonacci607&, boost::normal_distribution<> > normal(rng, mean_sigma);

accumulator_set<double, stats<tag::weighted_variance>, double> acc_lazy;
accumulator_set<double, stats<tag::weighted_variance(immediate)>, double> acc_immediate;

for (std::size_t i=0; i<10000; ++i)
{
    double value = normal();
    acc_lazy(value, weight = rng());
    acc_immediate(value, weight = rng());
}

BOOST_CHECK_CLOSE(1., weighted_variance(acc_lazy), 1.);
BOOST_CHECK_CLOSE(1., weighted_variance(acc_immediate), 1.);
```

## See also

- [lazy\\_weighted\\_variance\\_impl](#)
- [weighted\\_variance\\_impl](#)
- [count](#)
- [weighted\\_mean](#)
- [weighted\\_moment](#)

## Acknowledgements

Boost.Accumulators represents the efforts of many individuals. I would like to thank Daniel Egloff of [Zürcher Kantonalbank](#) for helping to conceive the library and realize its implementation. I would also like to thank David Abrahams and Matthias Troyer for their key contributions to the design of the library. Many thanks are due to Michael Gauckler and Olivier Gygi, who, along with Daniel Egloff, implemented many of the statistical accumulators.

I would also like to thank Simon West for all his assistance maintaining Boost.Accumulators.

Finally, I would like to thank [Zürcher Kantonalbank](#) for sponsoring the work on Boost.Accumulators and graciously donating it to the community.

# Reference

## Accumulators Framework Reference

### Header <boost/accumulators/accumulators.hpp>

Includes all of the Accumulators Framework

### Header <boost/accumulators/accumulators\_fwd.hpp>

```
BOOST_ACCUMULATORS_MAX_FEATURES
BOOST_ACCUMULATORS_MAX_ARGS
BOOST_ACCUMULATORS_PROTO_DISABLE_IF_IS_CONST(T)
BOOST_ACCUMULATORS_GCC_VERSION
BOOST_ACCUMULATORS_IGNORE_GLOBAL(X)
BOOST_PARAMETER_NESTED_KEYWORD(tag_namespace, name, alias)
```

```
namespace boost {
    namespace accumulators {
        template<typename Sample, typename Features, typename Weight = void>
            struct accumulator_set;
        template<typename Feature> struct as_feature;
        template<typename Feature> struct as_weighted_feature;
        template<BOOST_PP_ENUM_PARAMS_WITH_A_DEFAULT(BOOST_ACCUMULATORS_MAX_FEATURES, typename Feature, mpl::na) >
            struct depends_on;
        template<typename Accumulator> struct droppable_accumulator;
        template<typename Accumulator> struct droppable_accumulator_base;
        template<typename Feature> struct extractor;
        template<typename Feature> struct feature_of;
        template<BOOST_PP_ENUM_PARAMS_WITH_A_DEFAULT(BOOST_ACCUMULATORS_MAX_FEATURES, typename Feature, mpl::na) >
            struct features;
        template<typename Accumulator> struct with_cached_result;
        template<typename Feature, typename AccumulatorSet>
            mpl::apply< AccumulatorSet, Feature >::type::result_type
            extract_result(AccumulatorSet const & acc);
        template<typename Feature, typename AccumulatorSet, typename A1>
            mpl::apply< AccumulatorSet, Feature >::type::result_type
            extract_result(AccumulatorSet const & acc, A1 const & a1);
        namespace impl {
        }
        namespace tag {
            template<typename Feature> struct droppable;
            template<typename Type, typename Tag = void,
                    typename AccumulatorSet = void>
                struct external;
            template<typename Referent, typename Tag> struct reference;
            template<typename Tag> struct reference_tag;
            template<typename ValueType, typename Tag> struct value;
            template<typename Tag> struct value_tag;
        }
    }
}
```

## Struct template droppable

boost::accumulators::tag::droppable

## Synopsis

```
// In header: <boost/accumulators/accumulators_fwd.hpp>

template<typename Feature>
struct droppable : public as_feature::type< Feature > {
};
```

## Struct template external

boost::accumulators::tag::external

## Synopsis

```
// In header: <boost/accumulators/accumulators_fwd.hpp>

template<typename Type, typename Tag = void, typename AccumulatorSet = void>
struct external :
    public boost::accumulators::depends_on< reference< AccumulatorSet, Tag > >
{
};
```

## Struct template reference

boost::accumulators::tag::reference

## Synopsis

```
// In header: <boost/accumulators/accumulators_fwd.hpp>

template<typename Referent, typename Tag>
struct reference : public boost::accumulators::depends_on<> {
};
```

## Struct template reference\_tag

boost::accumulators::tag::reference\_tag

## Synopsis

```
// In header: <boost/accumulators/accumulators_fwd.hpp>

template<typename Tag>
struct reference_tag {
};
```

## Struct template value

boost::accumulators::tag::value

## Synopsis

```
// In header: <boost/accumulators/accumulators_fwd.hpp>

template<typename ValueType, typename Tag>
struct value : public boost::accumulators::depends_on<> {
};
```

### Struct template value\_tag

boost::accumulators::tag::value\_tag

## Synopsis

```
// In header: <boost/accumulators/accumulators_fwd.hpp>

template<typename Tag>
struct value_tag {
};
```

### Struct template accumulator\_set

boost::accumulators::accumulator\_set — A set of accumulators.

## Synopsis

```
// In header: <boost/accumulators/accumulators_fwd.hpp>

template<typename Sample, typename Features, typename Weight = void>
struct accumulator_set {
    // construct/copy/destruct
    template<typename A1> explicit accumulator_set(A1 const &);

    // public member functions
    template<typename UnaryFunction> void visit(UnaryFunction const &);
    template<typename FilterPred, typename UnaryFunction>
        void visit_if(UnaryFunction const &);
    void operator()();
    template<typename A1> void operator()(A1 const &);
    template<typename Feature> apply< Feature >::type & extract();
    template<typename Feature> apply< Feature >::type const & extract() const;
    template<typename Feature> void drop();
};
```

### Description

[accumulator\\_set](#) resolves the dependencies between features and ensures that the accumulators in the set are updated in the proper order.

`accumulator_set` provides a general mechanism to visit the accumulators in the set in order, with or without a filter. You can also fetch a reference to an accumulator that corresponds to a feature.

#### `accumulator_set` public construct/copy/destruct

1. 

```
template<typename A1> explicit accumulator_set(A1 const & a1);
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:      a1    Optional named parameter to be passed to all the accumulators

#### **accumulator\_set public member functions**

1. 

```
template<typename UnaryFunction> void visit(UnaryFunction const & func);
```

Visitation

Parameters:      func    UnaryFunction which is invoked with each accumulator in turn.

2. 

```
template<typename FilterPred, typename UnaryFunction>
void visit_if(UnaryFunction const & func);
```

Conditional visitation

Parameters:      func    UnaryFunction which is invoked with each accumulator in turn, provided the accumulator satisfies the MPL predicate FilterPred.

3. 

```
void operator()();
```

Accumulation

4. 

```
template<typename A1> void operator()(A1 const & a1);
```

5. 

```
template<typename Feature> apply< Feature >::type & extract();
```

Extraction.

6. 

```
template<typename Feature> apply< Feature >::type const & extract() const;
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

7. 

```
template<typename Feature> void drop();
```

Drop.

### **Struct template as\_feature**

boost::accumulators::as\_feature

## **Synopsis**

```
// In header: <boost/accumulators/accumulators_fwd.hpp>

template<typename Feature>
struct as_feature {
};
```



## Struct template `as_weighted_feature`

`boost::accumulators::as_weighted_feature`

## Synopsis

```
// In header: <boost/accumulators/accumulators_fwd.hpp>

template<typename Feature>
struct as_weighted_feature {
};
```

## Struct template `depends_on`

`boost::accumulators::depends_on` — [depends\\_on](#)

## Synopsis

```
// In header: <boost/accumulators/accumulators_fwd.hpp>

template<BOOST_PP_ENUM_PARAMS_WITH_A_DEFAULT(BOOST_ACCUMULATORS_MAX_FEATURES,
typename Feature, mpl::na) >
struct depends_on {
};
```

## Struct template `droppable_accumulator`

`boost::accumulators::droppable_accumulator`

## Synopsis

```
// In header: <boost/accumulators/accumulators_fwd.hpp>

template<typename Accumulator>
struct droppable_accumulator :
    public boost::accumulators::droppable_accumulator_base< Accumulator >
{
    // construct/copy/destroy
    template<typename Args> droppable_accumulator(Args const &);
    droppable_accumulator(droppable_accumulator const &);
};
```

## Description

**`droppable_accumulator` public construct/copy/destroy**

1. 

```
template<typename Args> droppable_accumulator(Args const & args);
```
2. 

```
droppable_accumulator(droppable_accumulator const & that);
```

## Struct template droppable\_accumulator\_base

boost::accumulators::droppable\_accumulator\_base

## Synopsis

```
// In header: <boost/accumulators/accumulators_fwd.hpp>

template<typename Accumulator>
struct droppable_accumulator_base : public Accumulator {
    // construct/copy/destruct
    template<typename Args> droppable_accumulator_base(Args const &);
    droppable_accumulator_base(droppable_accumulator_base const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    template<typename Args> void add_ref(Args const &);
    template<typename Args> void drop(Args const &);
    bool is_dropped() const;
};
```

### Description

**droppable\_accumulator\_base public construct/copy/destruct**

1. 

```
template<typename Args> droppable_accumulator_base(Args const & args);
```
2. 

```
droppable_accumulator_base(droppable_accumulator_base const & that);
```

**droppable\_accumulator\_base public member functions**

1. 

```
template<typename Args> void operator()(Args const & args);
```
2. 

```
template<typename Args> void add_ref(Args const &);
```
3. 

```
template<typename Args> void drop(Args const & args);
```
4. 

```
bool is_dropped() const;
```

## Struct template extractor

boost::accumulators::extractor — Extracts the result associated with Feature from the specified [accumulator\\_set](#).

## Synopsis

```
// In header: <boost/accumulators/accumulators_fwd.hpp>

template<typename Feature>
struct extractor {

    // public member functions
    template<typename Arg1> unspecified operator()(Arg1 const &) const;
    template<typename AccumulatorSet, typename A1>
        unspecified operator()(AccumulatorSet const &, A1 const &) const;
    template<typename AccumulatorSet, typename A1, typename A2, ... >
        unspecified operator()(AccumulatorSet const &, A1 const &, A2 const &,
                                ...);
};
```

### Description

#### extractor public member functions

1. 

```
template<typename Arg1> unspecified operator()(Arg1 const & arg1) const;
```

Extract the result associated with Feature from the accumulator set

2. 

```
template<typename AccumulatorSet, typename A1>
    unspecified operator()(AccumulatorSet const & acc, A1 const & a1) const;
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:      a1    Optional named parameter to be passed to the accumulator's result() function.

3. 

```
template<typename AccumulatorSet, typename A1, typename A2, ... >
    unspecified operator()(AccumulatorSet const & acc, A1 const & a1,
                          A2 const & a2, ...);
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

### Struct template feature\_of

boost::accumulators::feature\_of

## Synopsis

```
// In header: <boost/accumulators/accumulators_fwd.hpp>

template<typename Feature>
struct feature_of {
};
```

### Struct template features

boost::accumulators::features

## Synopsis

```
// In header: <boost/accumulators/accumulators_fwd.hpp>

template<BOOST_PP_ENUM_PARAMS_WITH_A_DEFAULT(BOOST_ACCUMULATORS_MAX_FEATURES, typename Feature, mpl::na) >
struct features : public mpl::vector< Feature1, Feature2,...> {
};
```

### Struct template with\_cached\_result

boost::accumulators::with\_cached\_result

## Synopsis

```
// In header: <boost/accumulators/accumulators_fwd.hpp>

template<typename Accumulator>
struct with_cached_result : public Accumulator {
    // construct/copy/destruct
    template<typename Args> with_cached_result(Args const &);
    with_cached_result(with_cached_result const &);
    with_cached_result & operator=(with_cached_result const &);
    ~with_cached_result();

    // public member functions
    template<typename Args> void on_drop(Args const &);
    template<typename Args> result_type result(Args const &) const;

    // private member functions
    void set(result_type const &);
    result_type const & get() const;
    bool has_result() const;
};
```

### Description

#### with\_cached\_result public construct/copy/destruct

1. 

```
template<typename Args> with_cached_result(Args const & args);
```
2. 

```
with_cached_result(with_cached_result const & that);
```
3. 

```
with_cached_result & operator=(with_cached_result const &);
```
4. 

```
~with_cached_result();
```

#### with\_cached\_result public member functions

1. 

```
template<typename Args> void on_drop(Args const & args);
```

```
2. template<typename Args> result_type result(Args const & args) const;
```

**with\_cached\_result private member functions**

```
1. void set(result_type const & r);
```

```
2. result_type const & get() const;
```

```
3. bool has_result() const;
```

## Macro BOOST\_ACCUMULATORS\_MAX\_FEATURES

BOOST\_ACCUMULATORS\_MAX\_FEATURES

## Synopsis

```
// In header: <boost/accumulators/accumulators_fwd.hpp>

BOOST_ACCUMULATORS_MAX_FEATURES
```

### Description

The maximum number of accumulators that may be put in an accumulator\_set. Defaults to BOOST\_MPL\_LIMIT\_VECTOR\_SIZE (which defaults to 20).

## Macro BOOST\_ACCUMULATORS\_MAX\_ARGS

BOOST\_ACCUMULATORS\_MAX\_ARGS

## Synopsis

```
// In header: <boost/accumulators/accumulators_fwd.hpp>

BOOST_ACCUMULATORS_MAX_ARGS
```

### Description

The maximum number of arguments that may be specified to an accumulator\_set's accumulation function. Defaults to 15.

## Macro BOOST\_ACCUMULATORS\_PROTO\_DISABLE\_IF\_IS\_CONST

BOOST\_ACCUMULATORS\_PROTO\_DISABLE\_IF\_IS\_CONST

## Synopsis

```
// In header: <boost/accumulators/accumulators_fwd.hpp>

BOOST_ACCUMULATORS_PROTO_DISABLE_IF_IS_CONST(T)
```

**Macro BOOST\_ACCUMULATORS\_GCC\_VERSION**

BOOST\_ACCUMULATORS\_GCC\_VERSION

**Synopsis**

```
// In header: <boost/accumulators/accumulators_fwd.hpp>

BOOST_ACCUMULATORS_GCC_VERSION
```

**Macro BOOST\_ACCUMULATORS\_IGNORE\_GLOBAL**

BOOST\_ACCUMULATORS\_IGNORE\_GLOBAL

**Synopsis**

```
// In header: <boost/accumulators/accumulators_fwd.hpp>

BOOST_ACCUMULATORS_IGNORE_GLOBAL(X)
```

**Macro BOOST\_PARAMETER\_NESTED\_KEYWORD**

BOOST\_PARAMETER\_NESTED\_KEYWORD

**Synopsis**

```
// In header: <boost/accumulators/accumulators_fwd.hpp>

BOOST_PARAMETER_NESTED_KEYWORD(tag_namespace, name, alias)
```

**Header <boost/accumulators/framework/accumulator\_base.hpp>**

```
namespace boost {
    namespace accumulators {
        struct accumulator_base;
        struct dont_care;
    }
}
```

**Struct accumulator\_base**

boost::accumulators::accumulator\_base

## Synopsis

```
// In header: <boost/accumulators/framework/accumulator_base.hpp>

struct accumulator_base {
    // types
    typedef mpl::false_ is_droppable;

    // public member functions
    unspecified operator()(dont_care);
    unspecified add_ref(dont_care);
    unspecified drop(dont_care);
    unspecified on_drop(dont_care);
};
```

### Description

**accumulator\_base public member functions**

1. `unspecified operator()(dont_care);`
2. `unspecified add_ref(dont_care);`
3. `unspecified drop(dont_care);`
4. `unspecified on_drop(dont_care);`

### Struct dont\_care

boost::accumulators::dont\_care

## Synopsis

```
// In header: <boost/accumulators/framework/accumulator_base.hpp>

struct dont_care {
    // construct/copy/destruct
    template<typename Args> dont_care(Args const &);
};
```

### Description

**dont\_care public construct/copy/destruct**

1. `template<typename Args> dont_care(Args const &);`

## Header <boost/accumulators/framework/accumulator\_concept.hpp>

```
namespace boost {
    namespace accumulators {
        template<typename Stat> struct accumulator_concept;
    }
}
```

### Struct template accumulator\_concept

boost::accumulators::accumulator\_concept

## Synopsis

```
// In header: <boost/accumulators/framework/accumulator_concept.hpp>

template<typename Stat>
struct accumulator_concept {

    // public member functions
    void constraints();

    // public data members
    Stat stat;
};
```

### Description

**accumulator\_concept** public member functions

1. `void constraints();`

## Header <boost/accumulators/framework/accumulator\_set.hpp>

```
namespace boost {
    namespace accumulators {
        template<typename Feature, typename AccumulatorSet>
        mpl::apply< AccumulatorSet, Feature >::type &
        find_accumulator(AccumulatorSet &acc BOOST_ACCUMULATORS_PROTO_DISABLE_IF_IS_CONST);
    }
}
```



## Header `<boost/accumulators/framework/accumulators/droppable_accumulator.hpp>`

```
namespace boost {
  namespace accumulators {
    template<typename Feature> struct as_feature<tag::droppable< Feature >>;
    template<typename Feature>
      struct as_weighted_feature<tag::droppable< Feature >>;
    template<typename Feature> struct feature_of<tag::droppable< Feature >>;
    namespace tag {
      template<typename Feature> struct as_droppable;

      template<typename Feature> struct as_droppable<droppable< Feature >>;
    }
  }
}
```

### Struct template `as_droppable`

`boost::accumulators::tag::as_droppable`

## Synopsis

```
// In header: <boost/accumulators/framework/accumulators/droppable_accumulator.hpp>

template<typename Feature>
struct as_droppable {
  // types
  typedef droppable< Feature > type;
};
```

### Struct template `as_droppable<droppable< Feature >>`

`boost::accumulators::tag::as_droppable<droppable< Feature >>`

## Synopsis

```
// In header: <boost/accumulators/framework/accumulators/droppable_accumulator.hpp>

template<typename Feature>
struct as_droppable<droppable< Feature >> {
  // types
  typedef droppable< Feature > type;
};
```

### Struct template `as_feature<tag::droppable< Feature >>`

`boost::accumulators::as_feature<tag::droppable< Feature >>`

## Synopsis

```
// In header: <boost/accumulators/framework/accumulators/droppable_accumulator.hpp>

template<typename Feature>
struct as_feature<tag::droppable< Feature >> {
    // types
    typedef tag::droppable< typename as_feature< Feature >::type > type;
};
```

### Struct template `as_weighted_feature<tag::droppable< Feature >>`

`boost::accumulators::as_weighted_feature<tag::droppable< Feature >>`

## Synopsis

```
// In header: <boost/accumulators/framework/accumulators/droppable_accumulator.hpp>

template<typename Feature>
struct as_weighted_feature<tag::droppable< Feature >> {
    // types
    typedef tag::droppable< typename as_weighted_feature< Feature >::type > type;
};
```

### Struct template `feature_of<tag::droppable< Feature >>`

`boost::accumulators::feature_of<tag::droppable< Feature >>`

## Synopsis

```
// In header: <boost/accumulators/framework/accumulators/droppable_accumulator.hpp>

template<typename Feature>
struct feature_of<tag::droppable< Feature >> : public boost::accumulators::feature_of< Feature > {
};
```

### Header `<boost/accumulators/framework/accumulators/external_accumulator.hpp>`

```
namespace boost {
    namespace accumulators {
        template<typename Feature, typename Tag, typename AccumulatorSet>
        struct feature_of<tag::external< Feature, Tag, AccumulatorSet >>;
        namespace impl {
        }
        namespace tag {
            template<typename Feature, typename Tag>
            struct external<Feature, Tag, void>;
        }
    }
}
```

## Struct template `external<Feature, Tag, void>`

`boost::accumulators::tag::external<Feature, Tag, void>`

## Synopsis

```
// In header: <boost/accumulators/framework/accumulators/external_accumulator.hpp>

template<typename Feature, typename Tag>
struct external<Feature, Tag, void> : public boost::accumulators::depends_on<> {
    // types
    typedef unspecified impl;
};
```

## Struct template `feature_of<tag::external< Feature, Tag, AccumulatorSet >>`

`boost::accumulators::feature_of<tag::external< Feature, Tag, AccumulatorSet >>`

## Synopsis

```
// In header: <boost/accumulators/framework/accumulators/external_accumulator.hpp>

template<typename Feature, typename Tag, typename AccumulatorSet>
struct feature_of<tag::external< Feature, Tag, AccumulatorSet >> : public boost::accumulators::feature_of< Feature > {
};
```

## Header `<boost/accumulators/framework/accumulators/reference_accumulator.hpp>`

```
namespace boost {
    namespace accumulators {
        template<typename ValueType, typename Tag>
        struct feature_of<tag::reference< ValueType, Tag >>;
        namespace extract {
        }
        namespace impl {
            template<typename Referent, typename Tag> struct reference_accumulator_impl;
        }
        namespace tag {
        }
    }
}
```

## Struct template `reference_accumulator_impl`

`boost::accumulators::impl::reference_accumulator_impl`

## Synopsis

```
// In header: <boost/accumulators/framework/accumulators/reference_accumulator.hpp>

template<typename Referent, typename Tag>
struct reference_accumulator_impl :
    public boost::accumulators::accumulator_base
{
    // types
    typedef Referent & result_type;

    // construct/copy/destruct
    template<typename Args> reference_accumulator_impl(Args const &);

    // public member functions
    result_type result(dont_care) const;
};
```

### Description

#### **reference\_accumulator\_impl public construct/copy/destruct**

```
1. template<typename Args> reference_accumulator_impl(Args const & args);
```

#### **reference\_accumulator\_impl public member functions**

```
1. result_type result(dont_care) const;
```

## Struct template **feature\_of<tag::reference< ValueType, Tag >>**

boost::accumulators::feature\_of<tag::reference< ValueType, Tag >>

## Synopsis

```
// In header: <boost/accumulators/framework/accumulators/reference_accumulator.hpp>

template<typename ValueType, typename Tag>
struct feature_of<tag::reference< ValueType, Tag >> :
    public boost::accumulators::feature_of< tag::reference_tag< Tag > >
{
};
```

**Header** <[boost/accumulators/framework/accumulators/value\\_accumulator.hpp](#)>

```
namespace boost {
  namespace accumulators {
    template<typename ValueType, typename Tag>
      struct feature_of<tag::value< ValueType, Tag >>;
    namespace extract {
    }
    namespace impl {
      template<typename ValueType, typename Tag> struct value_accumulator_impl;
    }
    namespace tag {
    }
  }
}
```

**Struct template value\_accumulator\_impl**

boost::accumulators::impl::value\_accumulator\_impl

**Synopsis**

```
// In header: <boost/accumulators/framework/accumulators/value_accumulator.hpp>

template<typename ValueType, typename Tag>
struct value_accumulator_impl : public boost::accumulators::accumulator_base {
  // types
  typedef ValueType result_type;

  // construct/copy/destruct
  template<typename Args> value_accumulator_impl(Args const &);

  // public member functions
  result_type result(dont_care) const;
};
```

**Description****value\_accumulator\_impl public construct/copy/destruct**

1. `template<typename Args> value_accumulator_impl(Args const & args);`

**value\_accumulator\_impl public member functions**

1. `result_type result(dont_care) const;`

**Struct template feature\_of<tag::value< ValueType, Tag >>**

boost::accumulators::feature\_of&lt;tag::value&lt; ValueType, Tag &gt;&gt;

## Synopsis

```
// In header: <boost/accumulators/framework/accumulators/value_accumulator.hpp>

template<typename ValueType, typename Tag>
struct feature_of<tag::value< ValueType, Tag >> :
    public boost::accumulators::feature_of< tag::value_tag< Tag > >
{
};
```

Header <[boost/accumulators/framework/depends\\_on.hpp](#)>

Header <[boost/accumulators/framework/extractor.hpp](#)>

```
BOOST_ACCUMULATORS_DEFINE_EXTRACTOR(Tag, Feature, ParamSeq)
```

Macro **BOOST\_ACCUMULATORS\_DEFINE\_EXTRACTOR**

BOOST\_ACCUMULATORS\_DEFINE\_EXTRACTOR

## Synopsis

```
// In header: <boost/accumulators/framework/extractor.hpp>

BOOST_ACCUMULATORS_DEFINE_EXTRACTOR(Tag, Feature, ParamSeq)
```

Header <[boost/accumulators/framework/features.hpp](#)>

Header <[boost/accumulators/framework/parameters/accumulator.hpp](#)>

```
namespace boost {
    namespace accumulators {
        boost::parameter::keyword< tag::accumulator > const accumulator;
        namespace tag {
            struct accumulator;
        }
    }
}
```

**Struct accumulator**

boost::accumulators::tag::accumulator

## Synopsis

```
// In header: <boost/accumulators/framework/parameters/accumulator.hpp>

struct accumulator {
};
```

## Global accumulator

boost::accumulators::accumulator

## Synopsis

```
// In header: <boost/accumulators/framework/parameters/accumulator.hpp>

boost::parameter::keyword< tag::accumulator > const accumulator;
```

## Header <boost/accumulators/framework/parameters/sample.hpp>

```
namespace boost {
    namespace accumulators {
        boost::parameter::keyword< tag::sample > const sample;
        namespace tag {
            struct sample;
        }
    }
}
```

## Struct sample

boost::accumulators::tag::sample

## Synopsis

```
// In header: <boost/accumulators/framework/parameters/sample.hpp>

struct sample {
};
```

## Global sample

boost::accumulators::sample

## Synopsis

```
// In header: <boost/accumulators/framework/parameters/sample.hpp>

boost::parameter::keyword< tag::sample > const sample;
```

## Header <boost/accumulators/framework/parameters/weight.hpp>

```
namespace boost {
    namespace accumulators {
        boost::parameter::keyword< tag::weight > const weight;
        namespace tag {
            struct weight;
        }
    }
}
```

## Struct weight

boost::accumulators::tag::weight

## Synopsis

```
// In header: <boost/accumulators/framework/parameters/weight.hpp>

struct weight {
};
```

## Global weight

boost::accumulators::weight

## Synopsis

```
// In header: <boost/accumulators/framework/parameters/weight.hpp>

boost::parameter::keyword< tag::weight > const weight;
```

## Header <boost/accumulators/framework/parameters/weights.hpp>

```
namespace boost {
  namespace accumulators {
    boost::parameter::keyword< tag::weights > const weights;
    namespace tag {
      struct weights;
    }
  }
}
```

## Struct weights

boost::accumulators::tag::weights

## Synopsis

```
// In header: <boost/accumulators/framework/parameters/weights.hpp>

struct weights {
};
```

## Global weights

boost::accumulators::weights



## Synopsis

```
// In header: <boost/accumulators/framework/parameters/weights.hpp>

boost::parameter::keyword< tag::weights > const weights;
```

## Statistics Library Reference

### Header <boost/accumulators/statistics.hpp>

Includes all of the Statistical Accumulators Library

### Header <boost/accumulators/statistics/count.hpp>

```
namespace boost {
  namespace accumulators {
    namespace extract {
      extractor< tag::count > const count;
    }
    namespace impl {
      struct count_impl;
    }
    namespace tag {
      struct count;
    }
  }
}
```

### Global count

boost::accumulators::extract::count

## Synopsis

```
// In header: <boost/accumulators/statistics/count.hpp>

extractor< tag::count > const count;
```

### Struct count\_impl

boost::accumulators::impl::count\_impl

## Synopsis

```
// In header: <boost/accumulators/statistics/count.hpp>

struct count_impl : public accumulator_base {
    // types
    typedef std::size_t result_type;

    // construct/copy/destroy
    count_impl(dont_care);

    // public member functions
    void operator()(dont_care);
    result_type result(dont_care) const;
};
```

### Description

**count\_impl public construct/copy/destroy**

1. `count_impl(dont_care);`

**count\_impl public member functions**

1. `void operator()(dont_care);`
2. `result_type result(dont_care) const;`

### Struct count

`boost::accumulators::tag::count`

## Synopsis

```
// In header: <boost/accumulators/statistics/count.hpp>

struct count : public boost::accumulators::depends_on<> {
};
```

## Header `<boost/accumulators/statistics/covariance.hpp>`

```
namespace boost {
  namespace accumulators {
    template<typename VariateType, typename VariateTag>
      struct as_weighted_feature<tag::covariance< VariateType, VariateTag >>;
    template<typename VariateType, typename VariateTag>
      struct feature_of<tag::covariance< VariateType, VariateTag >>;
    template<typename VariateType, typename VariateTag>
      struct feature_of<tag::weighted_covariance< VariateType, VariateTag >>;
    namespace extract {
      extractor< tag::abstract_covariance > const covariance;
    }
    namespace impl {
    }
    namespace tag {
      struct abstract_covariance;
    }
  }
  namespace numeric {
    namespace functional {
      template<typename Left, typename Right,
               typename LeftTag = typename tag<Left>::type,
               typename RightTag = typename tag<Right>::type>
        struct outer_product;

      template<typename Left, typename Right>
        struct outer_product<Left, Right, std_vector_tag, std_vector_tag>;

      template<typename Left, typename Right, typename EnableIf = void>
        struct outer_product_base;
    }
    namespace op {
      struct outer_product;
    }
  }
}
```

### Global covariance

`boost::accumulators::extract::covariance`

## Synopsis

```
// In header: <boost/accumulators/statistics/covariance.hpp>

extractor< tag::abstract_covariance > const covariance;
```

### Struct `abstract_covariance`

`boost::accumulators::tag::abstract_covariance`

## Synopsis

```
// In header: <boost/accumulators/statistics/covariance.hpp>

struct abstract_covariance : public boost::accumulators::depends_on<> {
};
```

### Struct template `as_weighted_feature<tag::covariance< VariateType, VariateTag >>`

`boost::accumulators::as_weighted_feature<tag::covariance< VariateType, VariateTag >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/covariance.hpp>

template<typename VariateType, typename VariateTag>
struct as_weighted_feature<tag::covariance< VariateType, VariateTag >> {
    // types
    typedef tag::weighted_covariance< VariateType, VariateTag > type;
};
```

### Struct template `feature_of<tag::covariance< VariateType, VariateTag >>`

`boost::accumulators::feature_of<tag::covariance< VariateType, VariateTag >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/covariance.hpp>

template<typename VariateType, typename VariateTag>
struct feature_of<tag::covariance< VariateType, VariateTag >> :
    public boost::accumulators::feature_of< tag::abstract_covariance >
{
};
```

### Struct template `feature_of<tag::weighted_covariance< VariateType, VariateTag >>`

`boost::accumulators::feature_of<tag::weighted_covariance< VariateType, VariateTag >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/covariance.hpp>

template<typename VariateType, typename VariateTag>
struct feature_of<tag::weighted_covariance< VariateType, VariateTag >> : public boost::accumulators::feature_of< tag::covariance< VariateType, VariateTag >>
{
};
```

### Struct template `outer_product`

`boost::numeric::functional::outer_product`

## Synopsis

```
// In header: <boost/accumulators/statistics/covariance.hpp>

template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
struct outer_product :
    public boost::numeric::functional::outer_product_base< Left, Right, void >
{
};
```

### Struct template outer\_product<Left, Right, std\_vector\_tag, std\_vector\_tag>

boost::numeric::functional::outer\_product<Left, Right, std\_vector\_tag, std\_vector\_tag>

## Synopsis

```
// In header: <boost/accumulators/statistics/covariance.hpp>

template<typename Left, typename Right>
struct outer_product<Left, Right, std_vector_tag, std_vector_tag> : public std::binary_function< Left, Right,
    ublas::matrix< functional::multiplies< Left::value_type, Right::value_type >::result_type > >
{
    // types
    typedef ublas::matrix< typename functional::multiplies< typename Left::value_type, typename
    Right::value_type >::result_type > result_type;

    // public member functions
    result_type operator()(Left &, Right &) const;
};
```

### Description

**outer\_product** public member functions

1. `result_type operator()(Left & left, Right & right) const;`

### Struct template outer\_product\_base

boost::numeric::functional::outer\_product\_base

## Synopsis

```
// In header: <boost/accumulators/statistics/covariance.hpp>

template<typename Left, typename Right, typename EnableIf = void>
struct outer_product_base : public functional::multiplies< Left, Right > {
};
```

### Struct outer\_product

boost::numeric::op::outer\_product

## Synopsis

```
// In header: <boost/accumulators/statistics/covariance.hpp>

struct outer_product {
};
```

## Header <boost/accumulators/statistics/density.hpp>

```
namespace boost {
  namespace accumulators {
    template<> struct as_weighted_feature<tag::density>;
    template<> struct feature_of<tag::weighted_density>;
    namespace extract {
      extractor< tag::density > const density;
    }
    namespace impl {
    }
    namespace tag {
      struct density;
    }
  }
}
```

### Global density

boost::accumulators::extract::density

## Synopsis

```
// In header: <boost/accumulators/statistics/density.hpp>

extractor< tag::density > const density;
```

### Struct density

boost::accumulators::tag::density

## Synopsis

```
// In header: <boost/accumulators/statistics/density.hpp>

struct density : public boost::accumulators::depends_on< count, min, max >,
                public density_cache_size,
                public density_num_bins
{
  // public data members
  static boost::parameter::keyword< density_cache_size > const cache_size;
  static boost::parameter::keyword< density_num_bins > const num_bins;
};
```

## Description

**density** public public data members

```
1. static boost::parameter::keyword< density_cache_size > const cache_size;
```

tag::density::cache\_size named parameter tag::density::num\_bins named parameter

## Struct as\_weighted\_feature<tag::density>

boost::accumulators::as\_weighted\_feature<tag::density>

## Synopsis

```
// In header: <boost/accumulators/statistics/density.hpp>

struct as_weighted_feature<tag::density> {
    // types
    typedef tag::weighted_density type;
};
```

## Struct feature\_of<tag::weighted\_density>

boost::accumulators::feature\_of<tag::weighted\_density>

## Synopsis

```
// In header: <boost/accumulators/statistics/density.hpp>

struct feature_of<tag::weighted_density> : public boost::accumulators::feature_of< tag::density > {
};
```

## Header <boost/accumulators/statistics/error\_of.hpp>

```
namespace boost {
    namespace accumulators {
        template<typename Feature> struct as_feature<tag::error_of< Feature >>;
        template<typename Feature>
            struct as_weighted_feature<tag::error_of< Feature >>;
        namespace extract {
        }
        namespace impl {
        }
        namespace tag {
        }
    }
}
```

## Struct template as\_feature<tag::error\_of< Feature >>

boost::accumulators::as\_feature<tag::error\_of< Feature >>

## Synopsis

```
// In header: <boost/accumulators/statistics/error_of.hpp>

template<typename Feature>
struct as_feature<tag::error_of< Feature >> {
    // types
    typedef tag::error_of< typename as_feature< Feature >::type > type;
};
```

### Struct template `as_weighted_feature<tag::error_of< Feature >>`

`boost::accumulators::as_weighted_feature<tag::error_of< Feature >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/error_of.hpp>

template<typename Feature>
struct as_weighted_feature<tag::error_of< Feature >> {
    // types
    typedef tag::error_of< typename as_weighted_feature< Feature >::type > type;
};
```

### Header `<boost/accumulators/statistics/error_of_mean.hpp>`

#### Struct `error_of<immediate_mean>`

`boost::accumulators::tag::error_of<immediate_mean>`

## Synopsis

```
// In header: <boost/accumulators/statistics/error_of_mean.hpp>

struct error_of<immediate_mean> : public boost::accumulators::depends_on< variance, count > {
};
```

#### Struct `error_of<mean>`

`boost::accumulators::tag::error_of<mean>`

## Synopsis

```
// In header: <boost/accumulators/statistics/error_of_mean.hpp>

struct error_of<mean> :
    public boost::accumulators::depends_on< lazy_variance, count >
{
};
```



## Header `<boost/accumulators/statistics/extended_p_square.hpp>`

```
namespace boost {
    namespace accumulators {
        template<> struct as_weighted_feature<tag::extended_p_square>;
        template<> struct feature_of<tag::weighted_extended_p_square>;
        namespace extract {
            extractor< tag::extended_p_square > const extended_p_square;
        }
        namespace impl {
        }
        namespace tag {
            struct extended_p_square;
        }
    }
}
```

### Global `extended_p_square`

`boost::accumulators::extract::extended_p_square`

## Synopsis

```
// In header: <boost/accumulators/statistics/extended_p_square.hpp>

extractor< tag::extended_p_square > const extended_p_square;
```

### Struct `extended_p_square`

`boost::accumulators::tag::extended_p_square`

## Synopsis

```
// In header: <boost/accumulators/statistics/extended_p_square.hpp>

struct extended_p_square : public boost::accumulators::depends_on< count >,
                           public extended_p_square_probabilities
{
    // types
    typedef accumulators::impl::extended_p_square_impl< mpl::_1 > impl;

    // public data members
    static boost::parameter::keyword< tag::probabilities > const probabilities; // tag::extended_p_square::probabilities named parameter
};
```

### Struct `as_weighted_feature<tag::extended_p_square>`

`boost::accumulators::as_weighted_feature<tag::extended_p_square>`

## Synopsis

```
// In header: <boost/accumulators/statistics/extended_p_square.hpp>

struct as_weighted_feature<tag::extended_p_square> {
    // types
    typedef tag::weighted_extended_p_square type;
};
```

### Struct feature\_of<tag::weighted\_extended\_p\_square>

boost::accumulators::feature\_of<tag::weighted\_extended\_p\_square>

## Synopsis

```
// In header: <boost/accumulators/statistics/extended_p_square.hpp>

struct feature_of<tag::weighted_extended_p_square> :
    public boost::accumulators::feature_of< tag::extended_p_square >
{
};
```

**Header <boost/accumulators/statistics/extended\_p\_square\_quantile.hpp>**

```

namespace boost {
    namespace accumulators {
        template<> struct as_feature<tag::extended_p_square_quantile(linear)>;
        template<> struct as_feature<tag::extended_p_square_quantile(quadratic)>;
        template<>
            struct as_feature<tag::weighted_extended_p_square_quantile(linear)>;
        template<>
            struct as_feature<tag::weighted_extended_p_square_quantile(quadratic)>;
        template<> struct as_weighted_feature<tag::extended_p_square_quantile>;
        template<>
            struct as_weighted_feature<tag::extended_p_square_quantile_quadratic>;
        template<> struct feature_of<tag::extended_p_square_quantile>;
        template<> struct feature_of<tag::extended_p_square_quantile_quadratic>;
        template<> struct feature_of<tag::weighted_extended_p_square_quantile>;
        template<>
            struct feature_of<tag::weighted_extended_p_square_quantile_quadratic>;
        namespace extract {
            extractor< tag::extended_p_square_quantile > const extended_p_square_quantile;
            extractor< tag::extended_p_square_quantile_quadratic > const extended_p_square_quantile_quad
            ratic;
            extractor< tag::weighted_extended_p_square_quantile > const weighted_exten
            ded_p_square_quantile;
            extractor< tag::weighted_extended_p_square_quantile_quadratic > const weighted_exten
            ded_p_square_quantile_quadratic;
        }
        namespace impl {
        }
        namespace tag {
            struct extended_p_square_quantile;
            struct extended_p_square_quantile_quadratic;
            struct weighted_extended_p_square_quantile;
            struct weighted_extended_p_square_quantile_quadratic;
        }
    }
}

```

**Global extended\_p\_square\_quantile**

boost::accumulators::extract::extended\_p\_square\_quantile

**Synopsis**

```

// In header: <boost/accumulators/statistics/extended_p_square_quantile.hpp>

extractor< tag::extended_p_square_quantile > const extended_p_square_quantile;

```

**Global extended\_p\_square\_quantile\_quadratic**

boost::accumulators::extract::extended\_p\_square\_quantile\_quadratic

**Synopsis**

```

// In header: <boost/accumulators/statistics/extended_p_square_quantile.hpp>

extractor< tag::extended_p_square_quantile_quadratic > const extended_p_square_quantile_quadratic;

```

## Global `weighted_extended_p_square_quantile`

`boost::accumulators::extract::weighted_extended_p_square_quantile`

## Synopsis

```
// In header: <boost/accumulators/statistics/extended_p_square_quantile.hpp>

extractor< tag::weighted_extended_p_square_quantile > const weighted_extended_p_square_quantile;
```

## Global `weighted_extended_p_square_quantile_quadratic`

`boost::accumulators::extract::weighted_extended_p_square_quantile_quadratic`

## Synopsis

```
// In header: <boost/accumulators/statistics/extended_p_square_quantile.hpp>

extractor< tag::weighted_extended_p_square_quantile_quadratic > const weighted_extended_p_square_quantile_quadratic;
```

## Struct `extended_p_square_quantile`

`boost::accumulators::tag::extended_p_square_quantile`

## Synopsis

```
// In header: <boost/accumulators/statistics/extended_p_square_quantile.hpp>

struct extended_p_square_quantile :
    public boost::accumulators::depends_on< extended_p_square >
{
    // types
    typedef accumulators::impl::extended_p_square_quantile_impl< mpl::_1, unweighted, linear > impl;
};
```

## Struct `extended_p_square_quantile_quadratic`

`boost::accumulators::tag::extended_p_square_quantile_quadratic`

## Synopsis

```
// In header: <boost/accumulators/statistics/extended_p_square_quantile.hpp>

struct extended_p_square_quantile_quadratic :
    public boost::accumulators::depends_on< extended_p_square >
{
    // types
    typedef accumulators::impl::extended_p_square_quantile_impl< mpl::_1, unweighted, quadratic > impl;
};
```

## Struct `weighted_extended_p_square_quantile`

`boost::accumulators::tag::weighted_extended_p_square_quantile`

## Synopsis

```
// In header: <boost/accumulators/statistics/extended_p_square_quantile.hpp>

struct weighted_extended_p_square_quantile :
    public boost::accumulators::depends_on< weighted_extended_p_square >
{
    // types
    typedef accumulators::impl::extended_p_square_quantile_impl< mpl::_1, weighted, linear > impl;
};
```

## Struct `weighted_extended_p_square_quantile_quadratic`

`boost::accumulators::tag::weighted_extended_p_square_quantile_quadratic`

## Synopsis

```
// In header: <boost/accumulators/statistics/extended_p_square_quantile.hpp>

struct weighted_extended_p_square_quantile_quadratic :
    public boost::accumulators::depends_on< weighted_extended_p_square >
{
    // types
    typedef accumulators::impl::extended_p_square_quantile_impl< mpl::_1, weighted, quadratic > impl;
};
```

## Struct `as_feature<tag::extended_p_square_quantile(linear)>`

`boost::accumulators::as_feature<tag::extended_p_square_quantile(linear)>`

## Synopsis

```
// In header: <boost/accumulators/statistics/extended_p_square_quantile.hpp>

struct as_feature<tag::extended_p_square_quantile(linear)> {
    // types
    typedef tag::extended_p_square_quantile type;
};
```

## Struct `as_feature<tag::extended_p_square_quantile(quadratic)>`

`boost::accumulators::as_feature<tag::extended_p_square_quantile(quadratic)>`

## Synopsis

```
// In header: <boost/accumulators/statistics/extended_p_square_quantile.hpp>

struct as_feature<tag::extended_p_square_quantile(quadratic)> {
    // types
    typedef tag::extended_p_square_quantile_quadratic type;
};
```

### Struct `as_feature<tag::weighted_extended_p_square_quantile(linear)>`

`boost::accumulators::as_feature<tag::weighted_extended_p_square_quantile(linear)>`

## Synopsis

```
// In header: <boost/accumulators/statistics/extended_p_square_quantile.hpp>

struct as_feature<tag::weighted_extended_p_square_quantile(linear)> {
    // types
    typedef tag::weighted_extended_p_square_quantile type;
};
```

### Struct `as_feature<tag::weighted_extended_p_square_quantile(quadratic)>`

`boost::accumulators::as_feature<tag::weighted_extended_p_square_quantile(quadratic)>`

## Synopsis

```
// In header: <boost/accumulators/statistics/extended_p_square_quantile.hpp>

struct as_feature<tag::weighted_extended_p_square_quantile(quadratic)> {
    // types
    typedef tag::weighted_extended_p_square_quantile_quadratic type;
};
```

### Struct `as_weighted_feature<tag::extended_p_square_quantile>`

`boost::accumulators::as_weighted_feature<tag::extended_p_square_quantile>`

## Synopsis

```
// In header: <boost/accumulators/statistics/extended_p_square_quantile.hpp>

struct as_weighted_feature<tag::extended_p_square_quantile> {
    // types
    typedef tag::weighted_extended_p_square_quantile type;
};
```

**Struct `as_weighted_feature<tag::extended_p_square_quantile_quadratic>`**`boost::accumulators::as_weighted_feature<tag::extended_p_square_quantile_quadratic>`**Synopsis**

```
// In header: <boost/accumulators/statistics/extended_p_square_quantile.hpp>

struct as_weighted_feature<tag::extended_p_square_quantile_quadratic> {
    // types
    typedef tag::weighted_extended_p_square_quantile_quadratic type;
};
```

**Struct `feature_of<tag::extended_p_square_quantile>`**`boost::accumulators::feature_of<tag::extended_p_square_quantile>`**Synopsis**

```
// In header: <boost/accumulators/statistics/extended_p_square_quantile.hpp>

struct feature_of<tag::extended_p_square_quantile> : public boost::accumulators::feature_of< tag::quantile > {
};
```

**Struct `feature_of<tag::extended_p_square_quantile_quadratic>`**`boost::accumulators::feature_of<tag::extended_p_square_quantile_quadratic>`**Synopsis**

```
// In header: <boost/accumulators/statistics/extended_p_square_quantile.hpp>

struct feature_of<tag::extended_p_square_quantile_quadratic> : public boost::accumulators::feature_of< tag::quantile > {
};
```

**Struct `feature_of<tag::weighted_extended_p_square_quantile>`**`boost::accumulators::feature_of<tag::weighted_extended_p_square_quantile>`**Synopsis**

```
// In header: <boost/accumulators/statistics/extended_p_square_quantile.hpp>

struct feature_of<tag::weighted_extended_p_square_quantile> :
    public boost::accumulators::feature_of< tag::extended_p_square_quantile >
{
};
```

## Struct `feature_of<tag::weighted_extended_p_square_quantile_quadratic>`

`boost::accumulators::feature_of<tag::weighted_extended_p_square_quantile_quadratic>`

## Synopsis

```
// In header: <boost/accumulators/statistics/extended_p_square_quantile.hpp>

struct feature_of<tag::weighted_extended_p_square_quantile_quadratic> : public boost::accumulators::feature_of< tag::extended_p_square_quantile_quadratic >
{
};
```

## Header `<boost/accumulators/statistics/kurtosis.hpp>`

```
namespace boost {
  namespace accumulators {
    template<> struct as_weighted_feature<tag::kurtosis>;
    template<> struct feature_of<tag::weighted_kurtosis>;
    namespace extract {
      extractor< tag::kurtosis > const kurtosis;
    }
    namespace impl {
    }
    namespace tag {
      struct kurtosis;
    }
  }
}
```

### Global kurtosis

`boost::accumulators::extract::kurtosis`

## Synopsis

```
// In header: <boost/accumulators/statistics/kurtosis.hpp>

extractor< tag::kurtosis > const kurtosis;
```

### Struct kurtosis

`boost::accumulators::tag::kurtosis`

## Synopsis

```
// In header: <boost/accumulators/statistics/kurtosis.hpp>

struct kurtosis : public boost::accumulators::depends_on< mean, moment< 2 >, moment< 3 >, moment< 4 > >
{
};
```



## Struct `as_weighted_feature<tag::kurtosis>`

`boost::accumulators::as_weighted_feature<tag::kurtosis>`

## Synopsis

```
// In header: <boost/accumulators/statistics/kurtosis.hpp>

struct as_weighted_feature<tag::kurtosis> {
    // types
    typedef tag::weighted_kurtosis type;
};
```

## Struct `feature_of<tag::weighted_kurtosis>`

`boost::accumulators::feature_of<tag::weighted_kurtosis>`

## Synopsis

```
// In header: <boost/accumulators/statistics/kurtosis.hpp>

struct feature_of<tag::weighted_kurtosis> : public boost::accumulators::feature_of< tag::kurtosis,
is > {
};
```

## Header `<boost/accumulators/statistics/max.hpp>`

```
namespace boost {
    namespace accumulators {
        namespace extract {
            extractor< tag::max > const max;
        }
        namespace impl {
        }
        namespace tag {
            struct max;
        }
    }
}
```

## Global `max`

`boost::accumulators::extract::max`

## Synopsis

```
// In header: <boost/accumulators/statistics/max.hpp>

extractor< tag::max > const max;
```

## Struct max

boost::accumulators::tag::max

## Synopsis

```
// In header: <boost/accumulators/statistics/max.hpp>

struct max : public boost::accumulators::depends_on<> {
};
```

## Header <boost/accumulators/statistics/mean.hpp>

```
namespace boost {
namespace accumulators {
template<> struct as_feature<tag::mean(immediate)>;
template<> struct as_feature<tag::mean(lazy)>;
template<typename VariateType, typename VariateTag>
    struct as_feature<tag::mean_of_variates< VariateType, VariateTag >(immediate)>;
template<typename VariateType, typename VariateTag>
    struct as_feature<tag::mean_of_variates< VariateType, VariateTag >(lazy)>;
template<> struct as_feature<tag::mean_of_weights(immediate)>;
template<> struct as_feature<tag::mean_of_weights(lazy)>;
template<> struct as_weighted_feature<tag::immediate_mean>;
template<typename VariateType, typename VariateTag>
    struct as_weighted_feature<tag::immediate_mean_of_variates< VariateType, VariateTag >>;
template<> struct as_weighted_feature<tag::mean>;
template<typename VariateType, typename VariateTag>
    struct as_weighted_feature<tag::mean_of_variates< VariateType, VariateTag >>;
template<> struct feature_of<tag::immediate_mean>;
template<typename VariateType, typename VariateTag>
    struct feature_of<tag::immediate_mean_of_variates< VariateType, VariateTag >>;
template<> struct feature_of<tag::immediate_mean_of_weights>;
template<> struct feature_of<tag::immediate_weighted_mean>;
template<typename VariateType, typename VariateTag>
    struct feature_of<tag::immediate_weighted_mean_of_variates< VariateType, VariateTag >>;
template<> struct feature_of<tag::weighted_mean>;
template<typename VariateType, typename VariateTag>
    struct feature_of<tag::weighted_mean_of_variates< VariateType, VariateTag >>;
namespace extract {
    extractor< tag::mean > const mean;
    extractor< tag::mean_of_weights > const mean_of_weights;
}
namespace impl {
}
namespace tag {
    struct immediate_mean;
    struct immediate_mean_of_weights;
    struct mean;
    struct mean_of_weights;
}
}
```

## Global mean

boost::accumulators::extract::mean

## Synopsis

```
// In header: <boost/accumulators/statistics/mean.hpp>

extractor< tag::mean > const mean;
```

### Global mean\_of\_weights

boost::accumulators::extract::mean\_of\_weights

## Synopsis

```
// In header: <boost/accumulators/statistics/mean.hpp>

extractor< tag::mean_of_weights > const mean_of_weights;
```

### Struct immediate\_mean

boost::accumulators::tag::immediate\_mean

## Synopsis

```
// In header: <boost/accumulators/statistics/mean.hpp>

struct immediate_mean : public boost::accumulators::depends_on< count > {
};
```

### Struct immediate\_mean\_of\_weights

boost::accumulators::tag::immediate\_mean\_of\_weights

## Synopsis

```
// In header: <boost/accumulators/statistics/mean.hpp>

struct immediate_mean_of_weights :
    public boost::accumulators::depends_on< count >
{
    // types
    typedef mpl::true_ is_weight_accumulator;
};
```

### Struct mean

boost::accumulators::tag::mean

## Synopsis

```
// In header: <boost/accumulators/statistics/mean.hpp>

struct mean : public boost::accumulators::depends_on< count, sum > {
};
```

### Struct mean\_of\_weights

boost::accumulators::tag::mean\_of\_weights

## Synopsis

```
// In header: <boost/accumulators/statistics/mean.hpp>

struct mean_of_weights :
    public boost::accumulators::depends_on< count, sum_of_weights >
{
    // types
    typedef mpl::true_ is_weight_accumulator;
};
```

### Struct as\_feature<tag::mean(immediate)>

boost::accumulators::as\_feature<tag::mean(immediate)>

## Synopsis

```
// In header: <boost/accumulators/statistics/mean.hpp>

struct as_feature<tag::mean(immediate)> {
    // types
    typedef tag::immediate_mean type;
};
```

### Struct as\_feature<tag::mean(lazy)>

boost::accumulators::as\_feature<tag::mean(lazy)>

## Synopsis

```
// In header: <boost/accumulators/statistics/mean.hpp>

struct as_feature<tag::mean(lazy)> {
    // types
    typedef tag::mean type;
};
```

**Struct template `as_feature<tag::mean_of_variates< VariateType, VariateTag >(immediate)>`**

`boost::accumulators::as_feature<tag::mean_of_variates< VariateType, VariateTag >(immediate)>`

## Synopsis

```
// In header: <boost/accumulators/statistics/mean.hpp>

template<typename VariateType, typename VariateTag>
struct as_feature<tag::mean_of_variates< VariateType, VariateTag >(immediate)> {
    // types
    typedef tag::immediate_mean_of_variates< VariateType, VariateTag > type;
};
```

**Struct template `as_feature<tag::mean_of_variates< VariateType, VariateTag >(lazy)>`**

`boost::accumulators::as_feature<tag::mean_of_variates< VariateType, VariateTag >(lazy)>`

## Synopsis

```
// In header: <boost/accumulators/statistics/mean.hpp>

template<typename VariateType, typename VariateTag>
struct as_feature<tag::mean_of_variates< VariateType, VariateTag >(lazy)> {
    // types
    typedef tag::mean_of_variates< VariateType, VariateTag > type;
};
```

**Struct `as_feature<tag::mean_of_weights(immediate)>`**

`boost::accumulators::as_feature<tag::mean_of_weights(immediate)>`

## Synopsis

```
// In header: <boost/accumulators/statistics/mean.hpp>

struct as_feature<tag::mean_of_weights(immediate)> {
    // types
    typedef tag::immediate_mean_of_weights type;
};
```

**Struct `as_feature<tag::mean_of_weights(lazy)>`**

`boost::accumulators::as_feature<tag::mean_of_weights(lazy)>`

## Synopsis

```
// In header: <boost/accumulators/statistics/mean.hpp>

struct as_feature<tag::mean_of_weights(lazy)> {
    // types
    typedef tag::mean_of_weights type;
};
```

### Struct `as_weighted_feature<tag::immediate_mean>`

`boost::accumulators::as_weighted_feature<tag::immediate_mean>`

## Synopsis

```
// In header: <boost/accumulators/statistics/mean.hpp>

struct as_weighted_feature<tag::immediate_mean> {
    // types
    typedef tag::immediate_weighted_mean type;
};
```

### Struct template `as_weighted_feature<tag::immediate_mean_of_variates< VariateType, VariateTag >>`

`boost::accumulators::as_weighted_feature<tag::immediate_mean_of_variates< VariateType, VariateTag >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/mean.hpp>

template<typename VariateType, typename VariateTag>
struct as_weighted_feature<tag::immediate_mean_of_variates< VariateType, VariateTag >> {
    // types
    typedef tag::immediate_weighted_mean_of_variates< VariateType, VariateTag > type;
};
```

### Struct `as_weighted_feature<tag::mean>`

`boost::accumulators::as_weighted_feature<tag::mean>`

## Synopsis

```
// In header: <boost/accumulators/statistics/mean.hpp>

struct as_weighted_feature<tag::mean> {
    // types
    typedef tag::weighted_mean type;
};
```

**Struct template `as_weighted_feature<tag::mean_of_variates< VariateType, VariateTag >>`**

`boost::accumulators::as_weighted_feature<tag::mean_of_variates< VariateType, VariateTag >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/mean.hpp>

template<typename VariateType, typename VariateTag>
struct as_weighted_feature<tag::mean_of_variates< VariateType, VariateTag >> {
    // types
    typedef tag::weighted_mean_of_variates< VariateType, VariateTag > type;
};
```

**Struct `feature_of<tag::immediate_mean>`**

`boost::accumulators::feature_of<tag::immediate_mean>`

## Synopsis

```
// In header: <boost/accumulators/statistics/mean.hpp>

struct feature_of<tag::immediate_mean> : public boost::accumulators::feature_of< tag::mean > {
};
```

**Struct template `feature_of<tag::immediate_mean_of_variates< VariateType, VariateTag >>`**

`boost::accumulators::feature_of<tag::immediate_mean_of_variates< VariateType, VariateTag >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/mean.hpp>

template<typename VariateType, typename VariateTag>
struct feature_of<tag::immediate_mean_of_variates< VariateType, VariateTag >> : public boost::ac-
cumulators::feature_of< tag::mean_of_variates< VariateType, VariateTag > >
{
};
```

**Struct `feature_of<tag::immediate_mean_of_weights>`**

`boost::accumulators::feature_of<tag::immediate_mean_of_weights>`

## Synopsis

```
// In header: <boost/accumulators/statistics/mean.hpp>

struct feature_of<tag::immediate_mean_of_weights> :
    public boost::accumulators::feature_of< tag::mean_of_weights >
{
};
```

**Struct feature\_of<tag::immediate\_weighted\_mean>**

boost::accumulators::feature\_of<tag::immediate\_weighted\_mean>

## Synopsis

```
// In header: <boost/accumulators/statistics/mean.hpp>

struct feature_of<tag::immediate_weighted_mean> :
    public boost::accumulators::feature_of< tag::immediate_mean >
{
};
```

**Struct template feature\_of<tag::immediate\_weighted\_mean\_of\_variates< VariateType, VariateTag >>**

boost::accumulators::feature\_of<tag::immediate\_weighted\_mean\_of\_variates< VariateType, VariateTag >>

## Synopsis

```
// In header: <boost/accumulators/statistics/mean.hpp>

template<typename VariateType, typename VariateTag>
struct feature_of<tag::immediate_weighted_mean_of_variates< VariateType, VariateTag >> : public boost::accumulators::feature_of< tag::immediate_mean_of_variates< VariateType, VariateTag >>
{
};
```

**Struct feature\_of<tag::weighted\_mean>**

boost::accumulators::feature\_of<tag::weighted\_mean>

## Synopsis

```
// In header: <boost/accumulators/statistics/mean.hpp>

struct feature_of<tag::weighted_mean> : public boost::accumulators::feature_of< tag::mean > {
};
```

**Struct template feature\_of<tag::weighted\_mean\_of\_variates< VariateType, VariateTag >>**

boost::accumulators::feature\_of<tag::weighted\_mean\_of\_variates< VariateType, VariateTag >>



## Synopsis

```
// In header: <boost/accumulators/statistics/mean.hpp>

template<typename VariateType, typename VariateTag>
struct feature_of<tag::weighted_mean_of_variates< VariateType, VariateTag >> : public boost::ac-
cumulators::feature_of< tag::mean_of_variates< VariateType, VariateTag > >
{
};
```

## Header <boost/accumulators/statistics/median.hpp>

```
namespace boost {
  namespace accumulators {
    template<> struct as_feature<tag::median(with_density)>;
    template<>
      struct as_feature<tag::median(with_p_square_cumulative_distribution)>;
    template<> struct as_feature<tag::median(with_p_square_quantile)>;
    template<> struct as_weighted_feature<tag::median>;
    template<> struct as_weighted_feature<tag::with_density_median>;
    template<>
      struct as_weighted_feature<tag::with_p_square_cumulative_distribution_median>;
    template<> struct feature_of<tag::weighted_median>;
    template<> struct feature_of<tag::with_density_median>;
    template<> struct feature_of<tag::with_density_weighted_median>;
    template<>
      struct feature_of<tag::with_p_square_cumulative_distribution_median>;
    template<>
      struct feature_of<tag::with_p_square_cumulative_distribution_weighted_median>;
    namespace extract {
      extractor< tag::median > const median;
      extractor< tag::with_density_median > const with_density_median;
      extractor< tag::with_p_square_cumulative_distribution_median > const with_p_square_cumulat-
ive_distribution_median;
    }
    namespace impl {
    }
    namespace tag {
      struct median;
      struct with_density_median;
      struct with_p_square_cumulative_distribution_median;
    }
  }
}
```

### Global median

boost::accumulators::extract::median

## Synopsis

```
// In header: <boost/accumulators/statistics/median.hpp>

extractor< tag::median > const median;
```

## Global with\_density\_median

boost::accumulators::extract::with\_density\_median

## Synopsis

```
// In header: <boost/accumulators/statistics/median.hpp>

extractor< tag::with_density_median > const with_density_median;
```

## Global with\_p\_square\_cumulative\_distribution\_median

boost::accumulators::extract::with\_p\_square\_cumulative\_distribution\_median

## Synopsis

```
// In header: <boost/accumulators/statistics/median.hpp>

extractor< tag::with_p_square_cumulative_distribution_median > const with_p_square_cumulative_distribution_median;
```

## Struct median

boost::accumulators::tag::median

## Synopsis

```
// In header: <boost/accumulators/statistics/median.hpp>

struct median :
    public boost::accumulators::depends_on< p_square_quantile_for_median >
{
};
```

## Struct with\_density\_median

boost::accumulators::tag::with\_density\_median

## Synopsis

```
// In header: <boost/accumulators/statistics/median.hpp>

struct with_density_median :
    public boost::accumulators::depends_on< count, density >
{
};
```

## Struct with\_p\_square\_cumulative\_distribution\_median

boost::accumulators::tag::with\_p\_square\_cumulative\_distribution\_median

## Synopsis

```
// In header: <boost/accumulators/statistics/median.hpp>

struct with_p_square_cumulative_distribution_median :
    public boost::accumulators::depends_on< p_square_cumulative_distribution >
{
};
```

### Struct `as_feature<tag::median(with_density)>`

`boost::accumulators::as_feature<tag::median(with_density)>`

## Synopsis

```
// In header: <boost/accumulators/statistics/median.hpp>

struct as_feature<tag::median(with_density)> {
    // types
    typedef tag::with_density_median type;
};
```

### Struct `as_feature<tag::median(with_p_square_cumulative_distribution)>`

`boost::accumulators::as_feature<tag::median(with_p_square_cumulative_distribution)>`

## Synopsis

```
// In header: <boost/accumulators/statistics/median.hpp>

struct as_feature<tag::median(with_p_square_cumulative_distribution)> {
    // types
    typedef tag::with_p_square_cumulative_distribution_median type;
};
```

### Struct `as_feature<tag::median(with_p_square_quantile)>`

`boost::accumulators::as_feature<tag::median(with_p_square_quantile)>`

## Synopsis

```
// In header: <boost/accumulators/statistics/median.hpp>

struct as_feature<tag::median(with_p_square_quantile)> {
    // types
    typedef tag::median type;
};
```

**Struct `as_weighted_feature<tag::median>`**`boost::accumulators::as_weighted_feature<tag::median>`**Synopsis**

```
// In header: <boost/accumulators/statistics/median.hpp>

struct as_weighted_feature<tag::median> {
    // types
    typedef tag::weighted_median type;
};
```

**Struct `as_weighted_feature<tag::with_density_median>`**`boost::accumulators::as_weighted_feature<tag::with_density_median>`**Synopsis**

```
// In header: <boost/accumulators/statistics/median.hpp>

struct as_weighted_feature<tag::with_density_median> {
    // types
    typedef tag::with_density_weighted_median type;
};
```

**Struct `as_weighted_feature<tag::with_p_square_cumulative_distribution_median>`**`boost::accumulators::as_weighted_feature<tag::with_p_square_cumulative_distribution_median>`**Synopsis**

```
// In header: <boost/accumulators/statistics/median.hpp>

struct as_weighted_feature<tag::with_p_square_cumulative_distribution_median> {
    // types
    typedef tag::with_p_square_cumulative_distribution_weighted_median type;
};
```

**Struct `feature_of<tag::weighted_median>`**`boost::accumulators::feature_of<tag::weighted_median>`**Synopsis**

```
// In header: <boost/accumulators/statistics/median.hpp>

struct feature_of<tag::weighted_median> : public boost::accumulators::feature_of< tag::median > {
};
```

**Struct feature\_of<tag::with\_density\_median>**

boost::accumulators::feature\_of<tag::with\_density\_median>

**Synopsis**

```
// In header: <boost/accumulators/statistics/median.hpp>

struct feature_of<tag::with_density_median> : public boost::accumulators::feature_of< tag::medi-
an > {
};
```

**Struct feature\_of<tag::with\_density\_weighted\_median>**

boost::accumulators::feature\_of<tag::with\_density\_weighted\_median>

**Synopsis**

```
// In header: <boost/accumulators/statistics/median.hpp>

struct feature_of<tag::with_density_weighted_median> :
    public boost::accumulators::feature_of< tag::with_density_median >
{
};
```

**Struct feature\_of<tag::with\_p\_square\_cumulative\_distribution\_median>**

boost::accumulators::feature\_of<tag::with\_p\_square\_cumulative\_distribution\_median>

**Synopsis**

```
// In header: <boost/accumulators/statistics/median.hpp>

struct feature_of<tag::with_p_square_cumulative_distribution_median> : public boost::accumulat-
ors::feature_of< tag::median > {
};
```

**Struct feature\_of<tag::with\_p\_square\_cumulative\_distribution\_weighted\_median>**

boost::accumulators::feature\_of<tag::with\_p\_square\_cumulative\_distribution\_weighted\_median>

**Synopsis**

```
// In header: <boost/accumulators/statistics/median.hpp>

struct feature_of<tag::with_p_square_cumulative_distribution_weighted_median> : public boost::ac-
cumulators::feature_of< tag::with_p_square_cumulative_distribution_median >
{
};
```

## Header <boost/accumulators/statistics/min.hpp>

```
namespace boost {
  namespace accumulators {
    namespace extract {
      extractor< tag::min > const min;
    }
    namespace impl {
    }
    namespace tag {
      struct min;
    }
  }
}
```

### Global min

boost::accumulators::extract::min

## Synopsis

```
// In header: <boost/accumulators/statistics/min.hpp>

extractor< tag::min > const min;
```

### Struct min

boost::accumulators::tag::min

## Synopsis

```
// In header: <boost/accumulators/statistics/min.hpp>

struct min : public boost::accumulators::depends_on<> {
};
```

## Header <boost/accumulators/statistics/moment.hpp>

```
namespace boost {
  namespace accumulators {
    template<int N> struct as_weighted_feature<tag::moment< N >>;
    template<int N> struct feature_of<tag::weighted_moment< N >>;
    namespace extract {
    }
    namespace impl {
    }
    namespace tag {
    }
  }
  namespace numeric {
  }
}
```

## Struct template `as_weighted_feature<tag::moment< N >>`

`boost::accumulators::as_weighted_feature<tag::moment< N >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/moment.hpp>

template<int N>
struct as_weighted_feature<tag::moment< N >> {
    // types
    typedef tag::weighted_moment< N > type;
};
```

## Struct template `feature_of<tag::weighted_moment< N >>`

`boost::accumulators::feature_of<tag::weighted_moment< N >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/moment.hpp>

template<int N>
struct feature_of<tag::weighted_moment< N >> :
    public boost::accumulators::feature_of< tag::moment< N > >
{
};
```

## Header `<boost/accumulators/statistics/p_square_cumul_dist.hpp>`

```
namespace boost {
    namespace accumulators {
        template<>
            struct as_weighted_feature<tag::p_square_cumulative_distribution>;
        template<>
            struct feature_of<tag::weighted_p_square_cumulative_distribution>;
        namespace extract {
            extractor< tag::p_square_cumulative_distribution > const p_square_cumulative_distribution;
        }
        namespace impl {
        }
        namespace tag {
            struct p_square_cumulative_distribution;
        }
    }
}
```

## Global `p_square_cumulative_distribution`

`boost::accumulators::extract::p_square_cumulative_distribution`

## Synopsis

```
// In header: <boost/accumulators/statistics/p_square_cumul_dist.hpp>

extractor< tag::p_square_cumulative_distribution > const p_square_cumulative_distribution;
```

### Struct p\_square\_cumulative\_distribution

boost::accumulators::tag::p\_square\_cumulative\_distribution

## Synopsis

```
// In header: <boost/accumulators/statistics/p_square_cumul_dist.hpp>

struct p_square_cumulative_distribution : public boost::accumulators::depends_on< count >,
                                         public p_square_cumulative_distribution_num_cells
{
};
```

### Struct as\_weighted\_feature<tag::p\_square\_cumulative\_distribution>

boost::accumulators::as\_weighted\_feature<tag::p\_square\_cumulative\_distribution>

## Synopsis

```
// In header: <boost/accumulators/statistics/p_square_cumul_dist.hpp>

struct as_weighted_feature<tag::p_square_cumulative_distribution> {
    // types
    typedef tag::weighted_p_square_cumulative_distribution type;
};
```

### Struct feature\_of<tag::weighted\_p\_square\_cumulative\_distribution>

boost::accumulators::feature\_of<tag::weighted\_p\_square\_cumulative\_distribution>

## Synopsis

```
// In header: <boost/accumulators/statistics/p_square_cumul_dist.hpp>

struct feature_of<tag::weighted_p_square_cumulative_distribution> : public boost::accumulators::feature_of< tag::p_square_cumulative_distribution >
{
};
```



## Header `<boost/accumulators/statistics/p_square_quantile.hpp>`

```
namespace boost {
  namespace accumulators {
    template<> struct as_weighted_feature<tag::p_square_quantile>;
    template<> struct feature_of<tag::weighted_p_square_quantile>;
    namespace extract {
      extractor< tag::p_square_quantile > const p_square_quantile;
      extractor< tag::p_square_quantile_for_median > const p_square_quantile_for_median;
    }
    namespace impl {
    }
    namespace tag {
      struct p_square_quantile;
      struct p_square_quantile_for_median;
    }
  }
}
```

### Global `p_square_quantile`

`boost::accumulators::extract::p_square_quantile`

## Synopsis

```
// In header: <boost/accumulators/statistics/p_square_quantile.hpp>

extractor< tag::p_square_quantile > const p_square_quantile;
```

### Global `p_square_quantile_for_median`

`boost::accumulators::extract::p_square_quantile_for_median`

## Synopsis

```
// In header: <boost/accumulators/statistics/p_square_quantile.hpp>

extractor< tag::p_square_quantile_for_median > const p_square_quantile_for_median;
```

### Struct `p_square_quantile`

`boost::accumulators::tag::p_square_quantile`

## Synopsis

```
// In header: <boost/accumulators/statistics/p_square_quantile.hpp>

struct p_square_quantile : public boost::accumulators::depends_on< count > {
};
```

## Struct `p_square_quantile_for_median`

`boost::accumulators::tag::p_square_quantile_for_median`

## Synopsis

```
// In header: <boost/accumulators/statistics/p_square_quantile.hpp>

struct p_square_quantile_for_median :
    public boost::accumulators::depends_on< count >
{
};
```

## Struct `as_weighted_feature<tag::p_square_quantile>`

`boost::accumulators::as_weighted_feature<tag::p_square_quantile>`

## Synopsis

```
// In header: <boost/accumulators/statistics/p_square_quantile.hpp>

struct as_weighted_feature<tag::p_square_quantile> {
    // types
    typedef tag::weighted_p_square_quantile type;
};
```

## Struct `feature_of<tag::weighted_p_square_quantile>`

`boost::accumulators::feature_of<tag::weighted_p_square_quantile>`

## Synopsis

```
// In header: <boost/accumulators/statistics/p_square_quantile.hpp>

struct feature_of<tag::weighted_p_square_quantile> :
    public boost::accumulators::feature_of< tag::p_square_quantile >
{
};
```

## Header `<boost/accumulators/statistics/peaks_over_threshold.hpp>`

```
namespace boost {
    namespace accumulators {
        template<typename LeftRight>
            struct as_feature<tag::peaks_over_threshold< LeftRight >(with_threshold_probability)>;
        template<typename LeftRight>
            struct as_feature<tag::peaks_over_threshold< LeftRight >(with_threshold_value)>;
        template<typename LeftRight>
            struct as_weighted_feature<tag::peaks_over_threshold< LeftRight >>;
        template<typename LeftRight>
            struct as_weighted_feature<tag::peaks_over_threshold_prob< LeftRight >>;
        template<typename LeftRight>
            struct feature_of<tag::peaks_over_threshold< LeftRight >>;
        template<typename LeftRight>
            struct feature_of<tag::peaks_over_threshold_prob< LeftRight >>;
        template<typename LeftRight>
            struct feature_of<tag::weighted_peaks_over_threshold< LeftRight >>;
        template<typename LeftRight>
            struct feature_of<tag::weighted_peaks_over_threshold_prob< LeftRight >>;
        namespace extract {
            extractor< tag::abstract_peaks_over_threshold > const peaks_over_threshold;
        }
        namespace impl {
            template<typename Sample, typename LeftRight>
                struct peaks_over_threshold_impl;
        }
        namespace tag {
            struct abstract_peaks_over_threshold;
        }
    }
}
```

### Global `peaks_over_threshold`

`boost::accumulators::extract::peaks_over_threshold`

## Synopsis

```
// In header: <boost/accumulators/statistics/peaks_over_threshold.hpp>

extractor< tag::abstract_peaks_over_threshold > const peaks_over_threshold;
```

### Struct template `peaks_over_threshold_impl`

`boost::accumulators::impl::peaks_over_threshold_impl` — Peaks over Threshold Method for Quantile and Tail Mean Estimation.

## Synopsis

```
// In header: <boost/accumulators/statistics/peaks_over_threshold.hpp>

template<typename Sample, typename LeftRight>
struct peaks_over_threshold_impl : public accumulator_base {
    // types
    typedef numeric::functional::fddiv< Sample, std::size_t >::result_type float_type;
    typedef boost::tuple< float_type, float_type, float_type > result_type;
    typedef mpl::int_< is_same< LeftRight, left >::value?-1:1 > sign;

    // construct/copy/destruct
    template<typename Args> peaks_over_threshold_impl(Args const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    template<typename Args> result_type result(Args const &) const;
};
```

### Description

According to the theorem of Pickands-Balkema-de Haan, the distribution function  $F_u(x)$  of the excesses  $x$  over some sufficiently high threshold  $u$  of a distribution function  $F(x)$  may be approximated by a generalized Pareto distribution

#### Equation 1.

$$G_{\xi, \beta}(x) = \begin{cases} \beta^{-1} \left(1 + \frac{\xi x}{\beta}\right)^{-1/\xi-1} & \text{if } \xi \neq 0 \\ \beta^{-1} \exp\left(-\frac{x}{\beta}\right) & \text{if } \xi = 0, \end{cases}$$

with suitable parameters  $\xi$  and  $\beta$  that can be estimated, e.g., with the method of moments, cf. Hosking and Wallis (1987),

#### Equation 2.

$$\begin{aligned} \hat{\xi} &= \frac{1}{2} \left[ 1 - \frac{(\hat{\mu} - u)^2}{\hat{\sigma}^2} \right] \\ \hat{\beta} &= \frac{\hat{\mu} - u}{2} \left[ \frac{(\hat{\mu} - u)^2}{\hat{\sigma}^2} + 1 \right], \end{aligned}$$

$\hat{\mu}$  and  $\hat{\sigma}^2$  being the empirical mean and variance of the samples over the threshold  $u$ . Equivalently, the distribution function  $F_u(x - u)$  of the exceedances  $x - u$  can be approximated by  $G_{\xi, \beta}(x - u) = G_{\xi, \beta, u}(x)$ . Since for  $x \geq u$  the distribution function  $F(x)$  can be written as

#### Equation 3.

$$F(x) = [1 - \mathbb{P}(X \leq u)]F_u(x - u) + \mathbb{P}(X \leq u)$$

and the probability  $\mathbb{P}(X \leq u)$  can be approximated by the empirical distribution function  $F_n(u)$  evaluated at  $u$ , an estimator of  $F(x)$  is given by

#### Equation 4.

$$\hat{F}(x) = [1 - F_n(u)]G_{\xi, \beta, u}(x) + F_n(u).$$

It can be shown that  $\hat{F}(x)$  is a generalized Pareto distribution  $G_{\xi, \beta, u}(x)$  with  $\bar{\beta} = \beta[1 - F_n(u)]^\xi$  and  $\bar{u} = u - \bar{\beta} \{[1 - F_n(u)]^{-\xi} - 1\} / \xi$ . By inverting  $\hat{F}(x)$ , one obtains an estimator for the  $\alpha$ -quantile,

**Equation 5.**

$$\hat{q}_\alpha = \hat{a} + \frac{\hat{\beta}}{\xi} [(1 - \alpha)^{-\xi} - 1],$$

and similarly an estimator for the (coherent) tail mean,

**Equation 6.**

$$\widehat{CTM}_\alpha = \hat{q}_\alpha - \frac{\hat{\beta}}{\xi - 1} (1 - \alpha)^{-\xi},$$

cf. McNeil and Frey (2000).

Note that in case extreme values of the left tail are fitted, the distribution is mirrored with respect to the  $y$  axis such that the left tail can be treated as a right tail. The computed fit parameters thus define the Pareto distribution that fits the mirrored left tail. When quantities like a quantile or a tail mean are computed using the fit parameters obtained from the mirrored data, the result is mirrored back, yielding the correct result.

For further details, see

J. R. M. Hosking and J. R. Wallis, Parameter and quantile estimation for the generalized Pareto distribution, *Technometrics*, Volume 29, 1987, p. 339-349

A. J. McNeil and R. Frey, Estimation of Tail-Related Risk Measures for Heteroscedastic Financial Time Series: an Extreme Value Approach, *Journal of Empirical Finance*, Volume 7, 2000, p. 271-300

**peaks\_over\_threshold\_impl public construct/copy/destruct**

```
1. template<typename Args> peaks_over_threshold_impl(Args const & args);
```

**peaks\_over\_threshold\_impl public member functions**

```
1. template<typename Args> void operator()(Args const & args);
```

```
2. template<typename Args> result_type result(Args const & args) const;
```

**Struct abstract\_peaks\_over\_threshold**

boost::accumulators::tag::abstract\_peaks\_over\_threshold

**Synopsis**

```
// In header: <boost/accumulators/statistics/peaks_over_threshold.hpp>

struct abstract_peaks_over_threshold :
    public boost::accumulators::depends_on<>
{
};
```

**Struct template as\_feature<tag::peaks\_over\_threshold< LeftRight >(with\_threshold\_probability)>**

boost::accumulators::as\_feature<tag::peaks\_over\_threshold< LeftRight >(with\_threshold\_probability)>

## Synopsis

```
// In header: <boost/accumulators/statistics/peaks_over_threshold.hpp>

template<typename LeftRight>
struct as_feature<tag::peaks_over_threshold< LeftRight >(with_threshold_probability)> {
    // types
    typedef tag::peaks_over_threshold_prob< LeftRight > type;
};
```

**Struct template as\_feature<tag::peaks\_over\_threshold< LeftRight >(with\_threshold\_value)>**

boost::accumulators::as\_feature<tag::peaks\_over\_threshold< LeftRight >(with\_threshold\_value)>

## Synopsis

```
// In header: <boost/accumulators/statistics/peaks_over_threshold.hpp>

template<typename LeftRight>
struct as_feature<tag::peaks_over_threshold< LeftRight >(with_threshold_value)> {
    // types
    typedef tag::peaks_over_threshold< LeftRight > type;
};
```

**Struct template as\_weighted\_feature<tag::peaks\_over\_threshold< LeftRight >>**

boost::accumulators::as\_weighted\_feature<tag::peaks\_over\_threshold< LeftRight >>

## Synopsis

```
// In header: <boost/accumulators/statistics/peaks_over_threshold.hpp>

template<typename LeftRight>
struct as_weighted_feature<tag::peaks_over_threshold< LeftRight >> {
    // types
    typedef tag::weighted_peaks_over_threshold< LeftRight > type;
};
```

**Struct template as\_weighted\_feature<tag::peaks\_over\_threshold\_prob< LeftRight >>**

boost::accumulators::as\_weighted\_feature<tag::peaks\_over\_threshold\_prob< LeftRight >>

## Synopsis

```
// In header: <boost/accumulators/statistics/peaks_over_threshold.hpp>

template<typename LeftRight>
struct as_weighted_feature<tag::peaks_over_threshold_prob< LeftRight >> {
    // types
    typedef tag::weighted_peaks_over_threshold_prob< LeftRight > type;
};
```

### Struct template feature\_of<tag::peaks\_over\_threshold< LeftRight >>

boost::accumulators::feature\_of<tag::peaks\_over\_threshold< LeftRight >>

## Synopsis

```
// In header: <boost/accumulators/statistics/peaks_over_threshold.hpp>

template<typename LeftRight>
struct feature_of<tag::peaks_over_threshold< LeftRight >> : public boost::accumulators::feature_of< tag::abstract_peaks_over_threshold >
{
};
```

### Struct template feature\_of<tag::peaks\_over\_threshold\_prob< LeftRight >>

boost::accumulators::feature\_of<tag::peaks\_over\_threshold\_prob< LeftRight >>

## Synopsis

```
// In header: <boost/accumulators/statistics/peaks_over_threshold.hpp>

template<typename LeftRight>
struct feature_of<tag::peaks_over_threshold_prob< LeftRight >> : public boost::accumulators::feature_of< tag::abstract_peaks_over_threshold >
{
};
```

### Struct template feature\_of<tag::weighted\_peaks\_over\_threshold< LeftRight >>

boost::accumulators::feature\_of<tag::weighted\_peaks\_over\_threshold< LeftRight >>

## Synopsis

```
// In header: <boost/accumulators/statistics/peaks_over_threshold.hpp>

template<typename LeftRight>
struct feature_of<tag::weighted_peaks_over_threshold< LeftRight >> : public boost::accumulators::feature_of< tag::peaks_over_threshold< LeftRight >>
{
};
```

**Struct template `feature_of<tag::weighted_peaks_over_threshold_prob< LeftRight >>`**

`boost::accumulators::feature_of<tag::weighted_peaks_over_threshold_prob< LeftRight >>`

**Synopsis**

```
// In header: <boost/accumulators/statistics/peaks_over_threshold.hpp>

template<typename LeftRight>
struct feature_of<tag::weighted_peaks_over_threshold_prob< LeftRight >> : public boost::accumulators::feature_of< tag::peaks_over_threshold_prob< LeftRight > >
{
};
```

**Header `<boost/accumulators/statistics/pot_quantile.hpp>`**

```
namespace boost {
  namespace accumulators {
    template<typename LeftRight>
      struct as_feature<tag::pot_quantile< LeftRight >(with_threshold_probability)>;
    template<typename LeftRight>
      struct as_feature<tag::pot_quantile< LeftRight >(with_threshold_value)>;
    template<typename LeftRight>
      struct as_feature<tag::weighted_pot_quantile< LeftRight >(with_threshold_probability)>;
    template<typename LeftRight>
      struct as_feature<tag::weighted_pot_quantile< LeftRight >(with_threshold_value)>;
    template<typename LeftRight>
      struct as_weighted_feature<tag::pot_quantile< LeftRight >>;
    template<typename LeftRight>
      struct as_weighted_feature<tag::pot_quantile_prob< LeftRight >>;
    template<typename LeftRight>
      struct feature_of<tag::pot_quantile< LeftRight >>;
    template<typename LeftRight>
      struct feature_of<tag::pot_quantile_prob< LeftRight >>;
    template<typename LeftRight>
      struct feature_of<tag::weighted_pot_quantile< LeftRight >>;
    template<typename LeftRight>
      struct feature_of<tag::weighted_pot_quantile_prob< LeftRight >>;
    namespace impl {
    }
    namespace tag {
    }
  }
}
```

**Struct template `as_feature<tag::pot_quantile< LeftRight >(with_threshold_probability)>`**

`boost::accumulators::as_feature<tag::pot_quantile< LeftRight >(with_threshold_probability)>`

**Synopsis**

```
// In header: <boost/accumulators/statistics/pot_quantile.hpp>

template<typename LeftRight>
struct as_feature<tag::pot_quantile< LeftRight >(with_threshold_probability)> {
  // types
  typedef tag::pot_quantile_prob< LeftRight > type;
};
```



**Struct template `as_feature<tag::pot_quantile< LeftRight >(with_threshold_value)>`**`boost::accumulators::as_feature<tag::pot_quantile< LeftRight >(with_threshold_value)>`

## Synopsis

```
// In header: <boost/accumulators/statistics/pot_quantile.hpp>

template<typename LeftRight>
struct as_feature<tag::pot_quantile< LeftRight >(with_threshold_value)> {
    // types
    typedef tag::pot_quantile< LeftRight > type;
};
```

**Struct `template` `as_feature<tag::weighted_pot_quantile< LeftRight >(with_threshold_probability)>`**`boost::accumulators::as_feature<tag::weighted_pot_quantile< LeftRight >(with_threshold_probability)>`

## Synopsis

```
// In header: <boost/accumulators/statistics/pot_quantile.hpp>

template<typename LeftRight>
struct as_feature<tag::weighted_pot_quantile< LeftRight >(with_threshold_probability)> {
    // types
    typedef tag::weighted_pot_quantile_prob< LeftRight > type;
};
```

**Struct template `as_feature<tag::weighted_pot_quantile< LeftRight >(with_threshold_value)>`**`boost::accumulators::as_feature<tag::weighted_pot_quantile< LeftRight >(with_threshold_value)>`

## Synopsis

```
// In header: <boost/accumulators/statistics/pot_quantile.hpp>

template<typename LeftRight>
struct as_feature<tag::weighted_pot_quantile< LeftRight >(with_threshold_value)> {
    // types
    typedef tag::weighted_pot_quantile< LeftRight > type;
};
```

**Struct template `as_weighted_feature<tag::pot_quantile< LeftRight >>`**`boost::accumulators::as_weighted_feature<tag::pot_quantile< LeftRight >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/pot_quantile.hpp>

template<typename LeftRight>
struct as_weighted_feature<tag::pot_quantile< LeftRight >> {
    // types
    typedef tag::weighted_pot_quantile< LeftRight > type;
};
```

### Struct template `as_weighted_feature<tag::pot_quantile_prob< LeftRight >>`

`boost::accumulators::as_weighted_feature<tag::pot_quantile_prob< LeftRight >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/pot_quantile.hpp>

template<typename LeftRight>
struct as_weighted_feature<tag::pot_quantile_prob< LeftRight >> {
    // types
    typedef tag::weighted_pot_quantile_prob< LeftRight > type;
};
```

### Struct template `feature_of<tag::pot_quantile< LeftRight >>`

`boost::accumulators::feature_of<tag::pot_quantile< LeftRight >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/pot_quantile.hpp>

template<typename LeftRight>
struct feature_of<tag::pot_quantile< LeftRight >> : public boost::accumulators::feature_of< tag::quantile > {
};
```

### Struct template `feature_of<tag::pot_quantile_prob< LeftRight >>`

`boost::accumulators::feature_of<tag::pot_quantile_prob< LeftRight >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/pot_quantile.hpp>

template<typename LeftRight>
struct feature_of<tag::pot_quantile_prob< LeftRight >> : public boost::accumulators::feature_of< tag::quantile > {
};
```

### Struct template `feature_of<tag::weighted_pot_quantile< LeftRight >>`

`boost::accumulators::feature_of<tag::weighted_pot_quantile< LeftRight >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/pot_quantile.hpp>

template<typename LeftRight>
struct feature_of<tag::weighted_pot_quantile< LeftRight >> :
    public boost::accumulators::feature_of< tag::pot_quantile< LeftRight > >
{
};
```

### Struct template feature\_of<tag::weighted\_pot\_quantile\_prob< LeftRight >>

boost::accumulators::feature\_of<tag::weighted\_pot\_quantile\_prob< LeftRight >>

## Synopsis

```
// In header: <boost/accumulators/statistics/pot_quantile.hpp>

template<typename LeftRight>
struct feature_of<tag::weighted_pot_quantile_prob< LeftRight >> : public boost::accumulators::feature_of< tag::pot_quantile_prob< LeftRight > >
{
};
```

### Header <boost/accumulators/statistics/pot\_tail\_mean.hpp>

```
namespace boost {
    namespace accumulators {
        template<typename LeftRight>
            struct as_feature<tag::pot_tail_mean< LeftRight >(with_threshold_probability)>;
        template<typename LeftRight>
            struct as_feature<tag::pot_tail_mean< LeftRight >(with_threshold_value)>;
        template<typename LeftRight>
            struct as_feature<tag::weighted_pot_tail_mean< LeftRight >(with_threshold_probability)>;
        template<typename LeftRight>
            struct as_feature<tag::weighted_pot_tail_mean< LeftRight >(with_threshold_value)>;
        template<typename LeftRight>
            struct as_weighted_feature<tag::pot_tail_mean< LeftRight >>;
        template<typename LeftRight>
            struct as_weighted_feature<tag::pot_tail_mean_prob< LeftRight >>;
        template<typename LeftRight>
            struct feature_of<tag::pot_tail_mean< LeftRight >>;
        template<typename LeftRight>
            struct feature_of<tag::pot_tail_mean_prob< LeftRight >>;
        template<typename LeftRight>
            struct feature_of<tag::weighted_pot_tail_mean< LeftRight >>;
        template<typename LeftRight>
            struct feature_of<tag::weighted_pot_tail_mean_prob< LeftRight >>;
        namespace impl {
        }
        namespace tag {
        }
    }
}
```

**Struct template as\_feature<tag::pot\_tail\_mean< LeftRight >(with\_threshold\_probability)>**

boost::accumulators::as\_feature<tag::pot\_tail\_mean< LeftRight >(with\_threshold\_probability)>

## Synopsis

```
// In header: <boost/accumulators/statistics/pot_tail_mean.hpp>

template<typename LeftRight>
struct as_feature<tag::pot_tail_mean< LeftRight >(with_threshold_probability)> {
    // types
    typedef tag::pot_tail_mean_prob< LeftRight > type;
};
```

**Struct template as\_feature<tag::pot\_tail\_mean< LeftRight >(with\_threshold\_value)>**

boost::accumulators::as\_feature<tag::pot\_tail\_mean< LeftRight >(with\_threshold\_value)>

## Synopsis

```
// In header: <boost/accumulators/statistics/pot_tail_mean.hpp>

template<typename LeftRight>
struct as_feature<tag::pot_tail_mean< LeftRight >(with_threshold_value)> {
    // types
    typedef tag::pot_tail_mean< LeftRight > type;
};
```

**Struct            template            as\_feature<tag::weighted\_pot\_tail\_mean<            LeftRight >(with\_threshold\_probability)>**

boost::accumulators::as\_feature<tag::weighted\_pot\_tail\_mean< LeftRight >(with\_threshold\_probability)>

## Synopsis

```
// In header: <boost/accumulators/statistics/pot_tail_mean.hpp>

template<typename LeftRight>
struct as_feature<tag::weighted_pot_tail_mean< LeftRight >(with_threshold_probability)> {
    // types
    typedef tag::weighted_pot_tail_mean_prob< LeftRight > type;
};
```

**Struct template as\_feature<tag::weighted\_pot\_tail\_mean< LeftRight >(with\_threshold\_value)>**

boost::accumulators::as\_feature<tag::weighted\_pot\_tail\_mean< LeftRight >(with\_threshold\_value)>

## Synopsis

```
// In header: <boost/accumulators/statistics/pot_tail_mean.hpp>

template<typename LeftRight>
struct as_feature<tag::weighted_pot_tail_mean< LeftRight >(with_threshold_value)> {
    // types
    typedef tag::weighted_pot_tail_mean< LeftRight > type;
};
```

### Struct template `as_weighted_feature<tag::pot_tail_mean< LeftRight >>`

`boost::accumulators::as_weighted_feature<tag::pot_tail_mean< LeftRight >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/pot_tail_mean.hpp>

template<typename LeftRight>
struct as_weighted_feature<tag::pot_tail_mean< LeftRight >> {
    // types
    typedef tag::weighted_pot_tail_mean< LeftRight > type;
};
```

### Struct template `as_weighted_feature<tag::pot_tail_mean_prob< LeftRight >>`

`boost::accumulators::as_weighted_feature<tag::pot_tail_mean_prob< LeftRight >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/pot_tail_mean.hpp>

template<typename LeftRight>
struct as_weighted_feature<tag::pot_tail_mean_prob< LeftRight >> {
    // types
    typedef tag::weighted_pot_tail_mean_prob< LeftRight > type;
};
```

### Struct template `feature_of<tag::pot_tail_mean< LeftRight >>`

`boost::accumulators::feature_of<tag::pot_tail_mean< LeftRight >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/pot_tail_mean.hpp>

template<typename LeftRight>
struct feature_of<tag::pot_tail_mean< LeftRight >> : public boost::accumulators::feature_of< tag::tail_mean > {
};
```

**Struct template feature\_of<tag::pot\_tail\_mean\_prob< LeftRight >>**

boost::accumulators::feature\_of<tag::pot\_tail\_mean\_prob< LeftRight >>

**Synopsis**

```
// In header: <boost/accumulators/statistics/pot_tail_mean.hpp>

template<typename LeftRight>
struct feature_of<tag::pot_tail_mean_prob< LeftRight >> : public boost::accumulators::fea-
ture_of< tag::tail_mean > {
};
```

**Struct template feature\_of<tag::weighted\_pot\_tail\_mean< LeftRight >>**

boost::accumulators::feature\_of<tag::weighted\_pot\_tail\_mean< LeftRight >>

**Synopsis**

```
// In header: <boost/accumulators/statistics/pot_tail_mean.hpp>

template<typename LeftRight>
struct feature_of<tag::weighted_pot_tail_mean< LeftRight >> :
    public boost::accumulators::feature_of< tag::pot_tail_mean< LeftRight > >
{
};
```

**Struct template feature\_of<tag::weighted\_pot\_tail\_mean\_prob< LeftRight >>**

boost::accumulators::feature\_of<tag::weighted\_pot\_tail\_mean\_prob< LeftRight >>

**Synopsis**

```
// In header: <boost/accumulators/statistics/pot_tail_mean.hpp>

template<typename LeftRight>
struct feature_of<tag::weighted_pot_tail_mean_prob< LeftRight >> : public boost::accumulators::fea-
ture_of< tag::pot_tail_mean_prob< LeftRight > >
{
};
```

**Header <boost/accumulators/statistics/rolling\_count.hpp>**

```
namespace boost {
    namespace accumulators {
        namespace extract {
            extractor< tag::rolling_count > const rolling_count;
        }
        namespace impl {
        }
        namespace tag {
            struct rolling_count;
        }
    }
}
```

## Global rolling\_count

boost::accumulators::extract::rolling\_count

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_count.hpp>

extractor< tag::rolling_count > const rolling_count;
```

## Struct rolling\_count

boost::accumulators::tag::rolling\_count

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_count.hpp>

struct rolling_count :
    public boost::accumulators::depends_on< rolling_window_plus1 >
{
    // public data members
    static boost::parameter::keyword< tag::rolling_window_size > const window_size; // ↵
    tag::rolling_window::window_size named parameter
};
```

## Header <boost/accumulators/statistics/rolling\_mean.hpp>

```
namespace boost {
    namespace accumulators {
        template<> struct as_feature<tag::rolling_mean(immediate)>;
        template<> struct as_feature<tag::rolling_mean(lazy)>;
        template<> struct feature_of<tag::immediate_rolling_mean>;
        template<> struct feature_of<tag::lazy_rolling_mean>;
        namespace extract {
            extractor< tag::lazy_rolling_mean > const lazy_rolling_mean;
            extractor< tag::immediate_rolling_mean > const immediate_rolling_mean;
            extractor< tag::rolling_mean > const rolling_mean;
        }
        namespace impl {
            template<typename Sample> struct immediate_rolling_mean_impl;
            template<typename Sample> struct lazy_rolling_mean_impl;
        }
        namespace tag {
            struct immediate_rolling_mean;
            struct lazy_rolling_mean;
            struct rolling_mean;
        }
    }
}
```

## Global lazy\_rolling\_mean

boost::accumulators::extract::lazy\_rolling\_mean

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_mean.hpp>

extractor< tag::lazy_rolling_mean > const lazy_rolling_mean;
```

### Global immediate\_rolling\_mean

boost::accumulators::extract::immediate\_rolling\_mean

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_mean.hpp>

extractor< tag::immediate_rolling_mean > const immediate_rolling_mean;
```

### Global rolling\_mean

boost::accumulators::extract::rolling\_mean

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_mean.hpp>

extractor< tag::rolling_mean > const rolling_mean;
```

### Struct template immediate\_rolling\_mean\_impl

boost::accumulators::impl::immediate\_rolling\_mean\_impl

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_mean.hpp>

template<typename Sample>
struct immediate_rolling_mean_impl : public accumulator_base {
    // types
    typedef numeric::functional::fddiv< Sample, std::size_t >::result_type result_type;

    // construct/copy/destruct
    template<typename Args> immediate_rolling_mean_impl(Args const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    template<typename Args> result_type result(Args const &) const;
};
```

### Description

**immediate\_rolling\_mean\_impl public construct/copy/destruct**

1. 

```
template<typename Args> immediate_rolling_mean_impl(Args const & args);
```



**immediate\_rolling\_mean\_impl public member functions**

1. 

```
template<typename Args> void operator()(Args const & args);
```
2. 

```
template<typename Args> result_type result(Args const &) const;
```

**Struct template lazy\_rolling\_mean\_impl**

boost::accumulators::impl::lazy\_rolling\_mean\_impl

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_mean.hpp>

template<typename Sample>
struct lazy_rolling_mean_impl : public accumulator_base {
    // types
    typedef numeric::functional::fddiv< Sample, std::size_t, void, void >::result_type result_type;

    // construct/copy/destroy
    lazy_rolling_mean_impl(dont_care);

    // public member functions
    template<typename Args> result_type result(Args const &) const;
};
```

**Description****lazy\_rolling\_mean\_impl public construct/copy/destroy**

1. 

```
lazy_rolling_mean_impl(dont_care);
```

**lazy\_rolling\_mean\_impl public member functions**

1. 

```
template<typename Args> result_type result(Args const & args) const;
```

**Struct immediate\_rolling\_mean**

boost::accumulators::tag::immediate\_rolling\_mean

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_mean.hpp>

struct immediate_rolling_mean : public boost::accumulators::depends_on< rolling_window_plus1, ↓
rolling_count >
{

    // public data members
    static boost::parameter::keyword< tag::rolling_window_size > const window_size; // ↓
    tag::rolling_window::window_size named parameter
};
```

### Struct lazy\_rolling\_mean

boost::accumulators::tag::lazy\_rolling\_mean

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_mean.hpp>

struct lazy_rolling_mean :
    public boost::accumulators::depends_on< rolling_sum, rolling_count >
{

    // public data members
    static boost::parameter::keyword< tag::rolling_window_size > const window_size; // ↓
    tag::rolling_window::window_size named parameter
};
```

### Struct rolling\_mean

boost::accumulators::tag::rolling\_mean

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_mean.hpp>

struct rolling_mean : public boost::accumulators::tag::immediate_rolling_mean {
};
```

### Struct as\_feature<tag::rolling\_mean(immediate)>

boost::accumulators::as\_feature<tag::rolling\_mean(immediate)>

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_mean.hpp>

struct as_feature<tag::rolling_mean(immediate)> {
    // types
    typedef tag::immediate_rolling_mean type;
};
```

### Struct as\_feature<tag::rolling\_mean(lazy)>

boost::accumulators::as\_feature<tag::rolling\_mean(lazy)>

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_mean.hpp>

struct as_feature<tag::rolling_mean(lazy)> {
    // types
    typedef tag::lazy_rolling_mean type;
};
```

### Struct feature\_of<tag::immediate\_rolling\_mean>

boost::accumulators::feature\_of<tag::immediate\_rolling\_mean>

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_mean.hpp>

struct feature_of<tag::immediate_rolling_mean> :
    public boost::accumulators::feature_of< tag::rolling_mean >
{
};
```

### Struct feature\_of<tag::lazy\_rolling\_mean>

boost::accumulators::feature\_of<tag::lazy\_rolling\_mean>

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_mean.hpp>

struct feature_of<tag::lazy_rolling_mean> :
    public boost::accumulators::feature_of< tag::rolling_mean >
{
};
```

## Header `<boost/accumulators/statistics/rolling_moment.hpp>`

```
namespace boost {
  namespace accumulators {
    namespace extract {
    }
    namespace impl {
      template<typename N, typename Sample> struct rolling_moment_impl;
    }
    namespace tag {
      template<int N> struct rolling_moment;
    }
  }
}
```

### Struct template `rolling_moment_impl`

`boost::accumulators::impl::rolling_moment_impl`

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_moment.hpp>

template<typename N, typename Sample>
struct rolling_moment_impl : public accumulator_base {
  // types
  typedef numeric::functional::fdiv< Sample, std::size_t, void, void >::result_type result_type;

  // construct/copy/destruct
  template<typename Args> rolling_moment_impl(Args const &);

  // public member functions
  BOOST_MPL_ASSERT_RELATION(N::value, 0);
  template<typename Args> void operator()(Args const &);
  template<typename Args> result_type result(Args const &) const;
};
```

### Description

#### `rolling_moment_impl` public construct/copy/destruct

1. `template<typename Args> rolling_moment_impl(Args const & args);`

#### `rolling_moment_impl` public member functions

1. `BOOST_MPL_ASSERT_RELATION(N::value, 0);`
2. `template<typename Args> void operator()(Args const & args);`
3. `template<typename Args> result_type result(Args const & args) const;`

## Struct template rolling\_moment

boost::accumulators::tag::rolling\_moment

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_moment.hpp>

template<int N>
struct rolling_moment : public boost::accumulators::depends_on< rolling_window_plus1, ↵
rolling_count >
{

    // public data members
    static boost::parameter::keyword< tag::rolling_window_size > const window_size; // ↵
    tag::rolling_window::window_size named parameter
};
```

## Header <boost/accumulators/statistics/rolling\_sum.hpp>

```
namespace boost {
    namespace accumulators {
        namespace extract {
            extractor< tag::rolling_sum > const rolling_sum;
        }
        namespace impl {
        }
        namespace tag {
            struct rolling_sum;
        }
    }
}
```

## Global rolling\_sum

boost::accumulators::extract::rolling\_sum

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_sum.hpp>

extractor< tag::rolling_sum > const rolling_sum;
```

## Struct rolling\_sum

boost::accumulators::tag::rolling\_sum

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_sum.hpp>

struct rolling_sum :
    public boost::accumulators::depends_on< rolling_window_plus1 >
{
    // public data members
    static boost::parameter::keyword< tag::rolling_window_size > const window_size; // ↵
    tag::rolling_window::window_size named parameter
};
```

## Header <boost/accumulators/statistics/rolling\_variance.hpp>

```
namespace boost {
    namespace accumulators {
        template<> struct as_feature<tag::rolling_variance(immediate)>;
        template<> struct as_feature<tag::rolling_variance(lazy)>;
        template<> struct feature_of<tag::immediate_rolling_variance>;
        template<> struct feature_of<tag::lazy_rolling_variance>;
        namespace extract {
            extractor< tag::lazy_rolling_variance > const lazy_rolling_variance;
            extractor< tag::immediate_rolling_variance > const immediate_rolling_variance;
            extractor< tag::rolling_variance > const rolling_variance;
        }
        namespace impl {
            template<typename Sample> struct immediate_rolling_variance_impl;
            template<typename Sample> struct lazy_rolling_variance_impl;
        }
        namespace tag {
            struct immediate_rolling_variance;
            struct lazy_rolling_variance;
            struct rolling_variance;
        }
    }
}
```

### Global lazy\_rolling\_variance

boost::accumulators::extract::lazy\_rolling\_variance

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_variance.hpp>

extractor< tag::lazy_rolling_variance > const lazy_rolling_variance;
```

### Global immediate\_rolling\_variance

boost::accumulators::extract::immediate\_rolling\_variance

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_variance.hpp>

extractor< tag::immediate_rolling_variance > const immediate_rolling_variance;
```

### Global rolling\_variance

boost::accumulators::extract::rolling\_variance

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_variance.hpp>

extractor< tag::rolling_variance > const rolling_variance;
```

### Struct template immediate\_rolling\_variance\_impl

boost::accumulators::impl::immediate\_rolling\_variance\_impl — Iterative calculation of the rolling variance.

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_variance.hpp>

template<typename Sample>
struct immediate_rolling_variance_impl : public accumulator_base {
    // types
    typedef numeric::functional::fdiv< Sample, std::size_t >::result_type result_type;

    // construct/copy/destruct
    template<typename Args> immediate_rolling_variance_impl(Args const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    template<typename Args> result_type result(Args const &) const;

    // private member functions
    template<typename T>
        void prevent_underflow(T &,
                                typename boost::enable_if< boost::is_arithmetic< T >, T >::type * = 0);
    template<typename T>
        void prevent_underflow(T &,
                                typename boost::disable_if< boost::is_arithmetic< T >, T >::type * = 0);
};
```

### Description

Iterative calculation of sample variance  $\sigma_n^2$  is done as follows, see also [http://en.wikipedia.org/wiki/Algorithms\\_for\\_calculating\\_variance](http://en.wikipedia.org/wiki/Algorithms_for_calculating_variance). For a rolling window of size  $N$ , for the first  $N$  samples, the variance is computed according to the formula

### Equation 7.

$$\sigma_n^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \mu_n)^2 = \frac{1}{n-1} M_{2,n},$$

where the sum of squares  $M_{2,n}$  can be recursively computed as:

#### Equation 8.

$$M_{2,n} = \sum_{i=1}^n (x_i - \mu_n)^2 = M_{2,n-1} + (x_n - \mu_n)(x_n - \mu_{n-1}),$$

and the estimate of the sample mean as:

#### Equation 9.

$$\mu_n = \frac{1}{n} \sum_{i=1}^n x_i = \mu_{n-1} + \frac{1}{n}(x_n - \mu_{n-1}).$$

For further samples, when the rolling window is fully filled with data, one has to take into account that the oldest sample  $x_{n-N}$  is dropped from the window. The sample variance over the window now becomes:

#### Equation 10.

$$\sigma_n^2 = \frac{1}{N-1} \sum_{i=n-N+1}^n (x_i - \mu_n)^2 = \frac{1}{n-1} M_{2,n},$$

where the sum of squares  $M_{2,n}$  now equals:

#### Equation 11.

$$M_{2,n} = \sum_{i=n-N+1}^n (x_i - \mu_n)^2 = M_{2,n-1} + (x_n - \mu_n)(x_n - \mu_{n-1}) - (x_{n-N} - \mu_n)(x_{n-N} - \mu_{n-1}),$$

and the estimated mean is:

#### Equation 12.

$$\mu_n = \frac{1}{N} \sum_{i=n-N+1}^n x_i = \mu_{n-1} + \frac{1}{N}(x_n - x_{n-N}).$$

Note that the sample variance is not defined for  $n \leq 1$ .

**immediate\_rolling\_variance\_impl public construct/copy/destruct**

```
1. template<typename Args> immediate_rolling_variance_impl(Args const & args);
```

**immediate\_rolling\_variance\_impl public member functions**

```
1. template<typename Args> void operator()(Args const & args);
```

```
2. template<typename Args> result_type result(Args const & args) const;
```



**immediate\_rolling\_variance\_impl private member functions**

1. 

```
template<typename T>
    void prevent_underflow(T & non_negative_number,
                           typename boost::enable_if< boost::is_arithmetic< T >, T >::type * =
0);
```
2. 

```
template<typename T>
    void prevent_underflow(T & non_arithmetic_quantity,
                           typename boost::disable_if< boost::is_arithmetic< T >, T >::type *
= 0);
```

**Struct template lazy\_rolling\_variance\_impl**

boost::accumulators::impl::lazy\_rolling\_variance\_impl — Immediate (lazy) calculation of the rolling variance.

**Synopsis**

```
// In header: <boost/accumulators/statistics/rolling_variance.hpp>

template<typename Sample>
struct lazy_rolling_variance_impl : public accumulator_base {
    // types
    typedef numeric::functional::fddiv< Sample, std::size_t, void, void >::result_type result_type;

    // construct/copy/destruct
    lazy_rolling_variance_impl(dont_care);

    // public member functions
    template<typename Args> result_type result(Args const &) const;
};
```

**Description**

Calculation of sample variance  $\sigma_n^2$  is done as follows, see also [http://en.wikipedia.org/wiki/Algorithms\\_for\\_calculating\\_variance](http://en.wikipedia.org/wiki/Algorithms_for_calculating_variance). For a rolling window of size  $N$ , when  $n \leq N$ , the variance is computed according to the formula

**Equation 13.**

$$\sigma_n^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \mu_n)^2.$$

When  $n > N$ , the sample variance over the window becomes:

**Equation 14.**

$$\sigma_n^2 = \frac{1}{N-1} \sum_{i=n-N+1}^n (x_i - \mu_n)^2.$$

**lazy\_rolling\_variance\_impl public construct/copy/destruct**

1. 

```
lazy_rolling_variance_impl(dont_care);
```

**lazy\_rolling\_variance\_impl public member functions**

```
1. template<typename Args> result_type result(Args const & args) const;
```

**Struct immediate\_rolling\_variance**

boost::accumulators::tag::immediate\_rolling\_variance

**Synopsis**

```
// In header: <boost/accumulators/statistics/rolling_variance.hpp>

struct immediate_rolling_variance : public boost::accumulators::depends_on< rolling_window_plus1, ↵
rolling_count, immediate_rolling_mean >
{
    // public data members
    static boost::parameter::keyword< tag::rolling_window_size > const window_size; // ↵
    tag::rolling_window::window_size named parameter
};
```

**Struct lazy\_rolling\_variance**

boost::accumulators::tag::lazy\_rolling\_variance

**Synopsis**

```
// In header: <boost/accumulators/statistics/rolling_variance.hpp>

struct lazy_rolling_variance : public boost::accumulators::depends_on< rolling_count, ↵
rolling_mean, rolling_moment< 2 > >
{
    // public data members
    static boost::parameter::keyword< tag::rolling_window_size > const window_size; // ↵
    tag::rolling_window::window_size named parameter
};
```

**Struct rolling\_variance**

boost::accumulators::tag::rolling\_variance

**Synopsis**

```
// In header: <boost/accumulators/statistics/rolling_variance.hpp>

struct rolling_variance :
    public boost::accumulators::tag::immediate_rolling_variance
{
};
```

**Struct `as_feature<tag::rolling_variance(immediate)>`**

`boost::accumulators::as_feature<tag::rolling_variance(immediate)>`

**Synopsis**

```
// In header: <boost/accumulators/statistics/rolling_variance.hpp>

struct as_feature<tag::rolling_variance(immediate)> {
    // types
    typedef tag::immediate_rolling_variance type;
};
```

**Struct `as_feature<tag::rolling_variance(lazy)>`**

`boost::accumulators::as_feature<tag::rolling_variance(lazy)>`

**Synopsis**

```
// In header: <boost/accumulators/statistics/rolling_variance.hpp>

struct as_feature<tag::rolling_variance(lazy)> {
    // types
    typedef tag::lazy_rolling_variance type;
};
```

**Struct `feature_of<tag::immediate_rolling_variance>`**

`boost::accumulators::feature_of<tag::immediate_rolling_variance>`

**Synopsis**

```
// In header: <boost/accumulators/statistics/rolling_variance.hpp>

struct feature_of<tag::immediate_rolling_variance> :
    public boost::accumulators::feature_of< tag::rolling_variance >
{
};
```

**Struct `feature_of<tag::lazy_rolling_variance>`**

`boost::accumulators::feature_of<tag::lazy_rolling_variance>`

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_variance.hpp>

struct feature_of<tag::lazy_rolling_variance> :
    public boost::accumulators::feature_of< tag::rolling_variance >
{
};
```

## Header **<boost/accumulators/statistics/rolling\_window.hpp>**

```
namespace boost {
    namespace accumulators {
        namespace extract {
            extractor< tag::rolling_window_plus1 > const rolling_window_plus1;
            extractor< tag::rolling_window > const rolling_window;
        }
        namespace impl {
            template<typename Args>
            bool is_rolling_window_plus1_full(Args const & args);
        }
        namespace tag {
            struct rolling_window;
            struct rolling_window_plus1;
        }
    }
}
```

### Global rolling\_window\_plus1

boost::accumulators::extract::rolling\_window\_plus1

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_window.hpp>

extractor< tag::rolling_window_plus1 > const rolling_window_plus1;
```

### Global rolling\_window

boost::accumulators::extract::rolling\_window

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_window.hpp>

extractor< tag::rolling_window > const rolling_window;
```

### Struct rolling\_window

boost::accumulators::tag::rolling\_window

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_window.hpp>

struct rolling_window :
    public boost::accumulators::depends_on< rolling_window_plus1 >
{
    // public data members
    static boost::parameter::keyword< tag::rolling_window_size > const window_size; // ↵
    tag::rolling_window::size named parameter
};
```

### Struct rolling\_window\_plus1

boost::accumulators::tag::rolling\_window\_plus1

## Synopsis

```
// In header: <boost/accumulators/statistics/rolling_window.hpp>

struct rolling_window_plus1 :
    public boost::accumulators::depends_on<>, public rolling_window_size
{
    // public data members
    static boost::parameter::keyword< tag::rolling_window_size > const window_size; // ↵
    tag::rolling_window::size named parameter
};
```

### Header <boost/accumulators/statistics/skewness.hpp>

```
namespace boost {
    namespace accumulators {
        template<> struct as_weighted_feature<tag::skewness>;
        template<> struct feature_of<tag::weighted_skewness>;
        namespace extract {
            extractor< tag::skewness > const skewness;
        }
        namespace impl {
        }
        namespace tag {
            struct skewness;
        }
    }
}
```

### Global skewness

boost::accumulators::extract::skewness

## Synopsis

```
// In header: <boost/accumulators/statistics/skewness.hpp>

extractor< tag::skewness > const skewness;
```

### Struct skewness

boost::accumulators::tag::skewness

## Synopsis

```
// In header: <boost/accumulators/statistics/skewness.hpp>

struct skewness :
    public boost::accumulators::depends_on< mean, moment< 2 >, moment< 3 > >
{
};
```

### Struct as\_weighted\_feature<tag::skewness>

boost::accumulators::as\_weighted\_feature<tag::skewness>

## Synopsis

```
// In header: <boost/accumulators/statistics/skewness.hpp>

struct as_weighted_feature<tag::skewness> {
    // types
    typedef tag::weighted_skewness type;
};
```

### Struct feature\_of<tag::weighted\_skewness>

boost::accumulators::feature\_of<tag::weighted\_skewness>

## Synopsis

```
// In header: <boost/accumulators/statistics/skewness.hpp>

struct feature_of<tag::weighted_skewness> : public boost::accumulators::feature_of< tag::skewness > {
};
```

### Header <boost/accumulators/statistics/stats.hpp>

Contains the stats<> template.

## Header `<boost/accumulators/statistics/sum.hpp>`

```
namespace boost {
  namespace accumulators {
    template<> struct as_weighted_feature<tag::sum>;
    template<typename VariateType, typename VariateTag>
      struct feature_of<tag::sum_of_variates< VariateType, VariateTag >>;
    template<> struct feature_of<tag::weighted_sum>;
    namespace extract {
      extractor< tag::sum > const sum;
      extractor< tag::sum_of_weights > const sum_of_weights;
      extractor< tag::abstract_sum_of_variates > const sum_of_variates;
    }
    namespace impl {
    }
    namespace tag {
      struct abstract_sum_of_variates;
      struct sum;
      struct sum_of_weights;
    }
  }
}
```

### Global sum

boost::accumulators::extract::sum

## Synopsis

```
// In header: <boost/accumulators/statistics/sum.hpp>

extractor< tag::sum > const sum;
```

### Global sum\_of\_weights

boost::accumulators::extract::sum\_of\_weights

## Synopsis

```
// In header: <boost/accumulators/statistics/sum.hpp>

extractor< tag::sum_of_weights > const sum_of_weights;
```

### Global sum\_of\_variates

boost::accumulators::extract::sum\_of\_variates

## Synopsis

```
// In header: <boost/accumulators/statistics/sum.hpp>

extractor< tag::abstract_sum_of_variates > const sum_of_variates;
```

## Struct `abstract_sum_of_variates`

`boost::accumulators::tag::abstract_sum_of_variates`

## Synopsis

```
// In header: <boost/accumulators/statistics/sum.hpp>

struct abstract_sum_of_variates : public boost::accumulators::depends_on<> {
};
```

## Struct `sum`

`boost::accumulators::tag::sum`

## Synopsis

```
// In header: <boost/accumulators/statistics/sum.hpp>

struct sum : public boost::accumulators::depends_on<> {
};
```

## Struct `sum_of_weights`

`boost::accumulators::tag::sum_of_weights`

## Synopsis

```
// In header: <boost/accumulators/statistics/sum.hpp>

struct sum_of_weights : public boost::accumulators::depends_on<> {
    // types
    typedef mpl::true_ is_weight_accumulator;
};
```

## Struct `as_weighted_feature<tag::sum>`

`boost::accumulators::as_weighted_feature<tag::sum>`

## Synopsis

```
// In header: <boost/accumulators/statistics/sum.hpp>

struct as_weighted_feature<tag::sum> {
    // types
    typedef tag::weighted_sum type;
};
```



## Struct template `feature_of<tag::sum_of_variates< VariateType, VariateTag >>`

`boost::accumulators::feature_of<tag::sum_of_variates< VariateType, VariateTag >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/sum.hpp>

template<typename VariateType, typename VariateTag>
struct feature_of<tag::sum_of_variates< VariateType, VariateTag >> :
    public boost::accumulators::feature_of< tag::abstract_sum_of_variates >
{
};
```

## Struct `feature_of<tag::weighted_sum>`

`boost::accumulators::feature_of<tag::weighted_sum>`

## Synopsis

```
// In header: <boost/accumulators/statistics/sum.hpp>

struct feature_of<tag::weighted_sum> : public boost::accumulators::feature_of< tag::sum > {
};
```

## Header `<boost/accumulators/statistics/sum_kahan.hpp>`

```
namespace boost {
    namespace accumulators {
        template<> struct as_feature<tag::sum(kahan)>;
        template<> struct as_feature<tag::sum_of_weights(kahan)>;
        template<> struct as_weighted_feature<tag::sum_kahan>;
        template<> struct feature_of<tag::sum_kahan>;
        template<typename VariateType, typename VariateTag>
            struct feature_of<tag::sum_of_variates_kahan< VariateType, VariateTag >>;
        template<> struct feature_of<tag::sum_of_weights_kahan>;
        template<> struct feature_of<tag::weighted_sum_kahan>;
        namespace extract {
            extractor< tag::sum_kahan > const sum_kahan;
            extractor< tag::sum_of_weights_kahan > const sum_of_weights_kahan;
            extractor< tag::abstract_sum_of_variates > const sum_of_variates_kahan;
        }
        namespace impl {
        }
        namespace tag {
            struct sum_kahan;
            struct sum_of_weights_kahan;
        }
    }
}
```

## Global `sum_kahan`

`boost::accumulators::extract::sum_kahan`

## Synopsis

```
// In header: <boost/accumulators/statistics/sum_kahan.hpp>

extractor< tag::sum_kahan > const sum_kahan;
```

### Global sum\_of\_weights\_kahan

boost::accumulators::extract::sum\_of\_weights\_kahan

## Synopsis

```
// In header: <boost/accumulators/statistics/sum_kahan.hpp>

extractor< tag::sum_of_weights_kahan > const sum_of_weights_kahan;
```

### Global sum\_of\_variates\_kahan

boost::accumulators::extract::sum\_of\_variates\_kahan

## Synopsis

```
// In header: <boost/accumulators/statistics/sum_kahan.hpp>

extractor< tag::abstract_sum_of_variates > const sum_of_variates_kahan;
```

### Struct sum\_kahan

boost::accumulators::tag::sum\_kahan

## Synopsis

```
// In header: <boost/accumulators/statistics/sum_kahan.hpp>

struct sum_kahan : public boost::accumulators::depends_on<> {
};
```

### Struct sum\_of\_weights\_kahan

boost::accumulators::tag::sum\_of\_weights\_kahan

## Synopsis

```
// In header: <boost/accumulators/statistics/sum_kahan.hpp>

struct sum_of_weights_kahan : public boost::accumulators::depends_on<> {
    // types
    typedef mpl::true_ is_weight_accumulator;
};
```

**Struct `as_feature<tag::sum(kahan)>`**`boost::accumulators::as_feature<tag::sum(kahan)>`**Synopsis**

```
// In header: <boost/accumulators/statistics/sum_kahan.hpp>

struct as_feature<tag::sum(kahan)> {
    // types
    typedef tag::sum_kahan type;
};
```

**Struct `as_feature<tag::sum_of_weights(kahan)>`**`boost::accumulators::as_feature<tag::sum_of_weights(kahan)>`**Synopsis**

```
// In header: <boost/accumulators/statistics/sum_kahan.hpp>

struct as_feature<tag::sum_of_weights(kahan)> {
    // types
    typedef tag::sum_of_weights_kahan type;
};
```

**Struct `as_weighted_feature<tag::sum_kahan>`**`boost::accumulators::as_weighted_feature<tag::sum_kahan>`**Synopsis**

```
// In header: <boost/accumulators/statistics/sum_kahan.hpp>

struct as_weighted_feature<tag::sum_kahan> {
    // types
    typedef tag::weighted_sum_kahan type;
};
```

**Struct `feature_of<tag::sum_kahan>`**`boost::accumulators::feature_of<tag::sum_kahan>`**Synopsis**

```
// In header: <boost/accumulators/statistics/sum_kahan.hpp>

struct feature_of<tag::sum_kahan> : public boost::accumulators::feature_of< tag::sum > {
};
```

**Struct template feature\_of<tag::sum\_of\_variates\_kahan< VariateType, VariateTag >>**

boost::accumulators::feature\_of<tag::sum\_of\_variates\_kahan< VariateType, VariateTag >>

## Synopsis

```
// In header: <boost/accumulators/statistics/sum_kahan.hpp>

template<typename VariateType, typename VariateTag>
struct feature_of<tag::sum_of_variates_kahan< VariateType, VariateTag >> :
    public boost::accumulators::feature_of< tag::abstract_sum_of_variates >
{
};
```

**Struct feature\_of<tag::sum\_of\_weights\_kahan>**

boost::accumulators::feature\_of<tag::sum\_of\_weights\_kahan>

## Synopsis

```
// In header: <boost/accumulators/statistics/sum_kahan.hpp>

struct feature_of<tag::sum_of_weights_kahan> :
    public boost::accumulators::feature_of< tag::sum_of_weights >
{
};
```

**Struct feature\_of<tag::weighted\_sum\_kahan>**

boost::accumulators::feature\_of<tag::weighted\_sum\_kahan>

## Synopsis

```
// In header: <boost/accumulators/statistics/sum_kahan.hpp>

struct feature_of<tag::weighted_sum_kahan> : public boost::accumulators::feature_of< tag::sum > {
};
```

## Header `<boost/accumulators/statistics/tail.hpp>`

```
namespace boost {
    namespace accumulators {
        template<typename LeftRight> struct feature_of<tag::tail< LeftRight >>;

        template<typename T> struct tail_cache_size_named_arg;

        template<> struct tail_cache_size_named_arg<left>;
        template<> struct tail_cache_size_named_arg<right>;
        namespace extract {
            extractor< tag::abstract_tail > const tail;
        }
        namespace impl {
        }
        namespace tag {
            struct abstract_tail;
        }
    }
}
```

### Global tail

boost::accumulators::extract::tail

## Synopsis

```
// In header: <boost/accumulators/statistics/tail.hpp>

extractor< tag::abstract_tail > const tail;
```

### Struct abstract\_tail

boost::accumulators::tag::abstract\_tail

## Synopsis

```
// In header: <boost/accumulators/statistics/tail.hpp>

struct abstract_tail : public boost::accumulators::depends_on<> {
};
```

### Struct template feature\_of<tag::tail< LeftRight >>

boost::accumulators::feature\_of<tag::tail< LeftRight >>

## Synopsis

```
// In header: <boost/accumulators/statistics/tail.hpp>

template<typename LeftRight>
struct feature_of<tag::tail< LeftRight >> :
    public boost::accumulators::feature_of< tag::abstract_tail >
{
};
```

### Struct template tail\_cache\_size\_named\_arg

boost::accumulators::tail\_cache\_size\_named\_arg

## Synopsis

```
// In header: <boost/accumulators/statistics/tail.hpp>

template<typename T>
struct tail_cache_size_named_arg {
};
```

### Struct tail\_cache\_size\_named\_arg<left>

boost::accumulators::tail\_cache\_size\_named\_arg<left>

## Synopsis

```
// In header: <boost/accumulators/statistics/tail.hpp>

struct tail_cache_size_named_arg<left> : public left_tail_cache_size {
};
```

### Struct tail\_cache\_size\_named\_arg<right>

boost::accumulators::tail\_cache\_size\_named\_arg<right>

## Synopsis

```
// In header: <boost/accumulators/statistics/tail.hpp>

struct tail_cache_size_named_arg<right> : public right_tail_cache_size {
};
```

## Header `<boost/accumulators/statistics/tail_mean.hpp>`

```
namespace boost {
  namespace accumulators {
    template<typename LeftRight>
      struct as_weighted_feature<tag::non_coherent_tail_mean< LeftRight >>;
    template<typename LeftRight>
      struct feature_of<tag::coherent_tail_mean< LeftRight >>;
    template<typename LeftRight>
      struct feature_of<tag::non_coherent_tail_mean< LeftRight >>;
    template<typename LeftRight>
      struct feature_of<tag::non_coherent_weighted_tail_mean< LeftRight >>;
    namespace extract {
      extractor< tag::abstract_non_coherent_tail_mean > const non_coherent_tail_mean;
      extractor< tag::tail_mean > const coherent_tail_mean;
    }
    namespace impl {
    }
    namespace tag {
      struct abstract_non_coherent_tail_mean;
    }
  }
}
```

### Global `non_coherent_tail_mean`

`boost::accumulators::extract::non_coherent_tail_mean`

## Synopsis

```
// In header: <boost/accumulators/statistics/tail_mean.hpp>

extractor< tag::abstract_non_coherent_tail_mean > const non_coherent_tail_mean;
```

### Global `coherent_tail_mean`

`boost::accumulators::extract::coherent_tail_mean`

## Synopsis

```
// In header: <boost/accumulators/statistics/tail_mean.hpp>

extractor< tag::tail_mean > const coherent_tail_mean;
```

### Struct `abstract_non_coherent_tail_mean`

`boost::accumulators::tag::abstract_non_coherent_tail_mean`

## Synopsis

```
// In header: <boost/accumulators/statistics/tail_mean.hpp>

struct abstract_non_coherent_tail_mean :
    public boost::accumulators::depends_on<>
{
};
```

### Struct template `as_weighted_feature<tag::non_coherent_tail_mean< LeftRight >>`

`boost::accumulators::as_weighted_feature<tag::non_coherent_tail_mean< LeftRight >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/tail_mean.hpp>

template<typename LeftRight>
struct as_weighted_feature<tag::non_coherent_tail_mean< LeftRight >> {
    // types
    typedef tag::non_coherent_weighted_tail_mean< LeftRight > type;
};
```

### Struct template `feature_of<tag::coherent_tail_mean< LeftRight >>`

`boost::accumulators::feature_of<tag::coherent_tail_mean< LeftRight >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/tail_mean.hpp>

template<typename LeftRight>
struct feature_of<tag::coherent_tail_mean< LeftRight >> : public boost::accumulators::feature_of< tag::tail_mean > {
};
```

### Struct template `feature_of<tag::non_coherent_tail_mean< LeftRight >>`

`boost::accumulators::feature_of<tag::non_coherent_tail_mean< LeftRight >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/tail_mean.hpp>

template<typename LeftRight>
struct feature_of<tag::non_coherent_tail_mean< LeftRight >> : public boost::accumulators::feature_of< tag::abstract_non_coherent_tail_mean >
{
};
```



**Struct template `feature_of<tag::non_coherent_weighted_tail_mean< LeftRight >>`**

`boost::accumulators::feature_of<tag::non_coherent_weighted_tail_mean< LeftRight >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/tail_mean.hpp>

template<typename LeftRight>
struct feature_of<tag::non_coherent_weighted_tail_mean< LeftRight >> : public boost::accumulators::feature_of< tag::non_coherent_tail_mean< LeftRight > >
{
};
```

**Header `<boost/accumulators/statistics/tail_quantile.hpp>`**

```
namespace boost {
  namespace accumulators {
    template<typename LeftRight>
      struct as_weighted_feature<tag::tail_quantile< LeftRight >>;
    template<typename LeftRight>
      struct feature_of<tag::tail_quantile< LeftRight >>;
    template<typename LeftRight>
      struct feature_of<tag::weighted_tail_quantile< LeftRight >>;
    namespace extract {
      extractor< tag::quantile > const tail_quantile;
    }
    namespace impl {
    }
    namespace tag {
    }
  }
}
```

**Global `tail_quantile`**

`boost::accumulators::extract::tail_quantile`

## Synopsis

```
// In header: <boost/accumulators/statistics/tail_quantile.hpp>

extractor< tag::quantile > const tail_quantile;
```

**Struct template `as_weighted_feature<tag::tail_quantile< LeftRight >>`**

`boost::accumulators::as_weighted_feature<tag::tail_quantile< LeftRight >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/tail_quantile.hpp>

template<typename LeftRight>
struct as_weighted_feature<tag::tail_quantile< LeftRight >> {
    // types
    typedef tag::weighted_tail_quantile< LeftRight > type;
};
```

### Struct template feature\_of<tag::tail\_quantile< LeftRight >>

boost::accumulators::feature\_of<tag::tail\_quantile< LeftRight >>

## Synopsis

```
// In header: <boost/accumulators/statistics/tail_quantile.hpp>

template<typename LeftRight>
struct feature_of<tag::tail_quantile< LeftRight >> : public boost::accumulators::fea-
ture_of< tag::quantile > {
};
```

### Struct template feature\_of<tag::weighted\_tail\_quantile< LeftRight >>

boost::accumulators::feature\_of<tag::weighted\_tail\_quantile< LeftRight >>

## Synopsis

```
// In header: <boost/accumulators/statistics/tail_quantile.hpp>

template<typename LeftRight>
struct feature_of<tag::weighted_tail_quantile< LeftRight >> :
    public boost::accumulators::feature_of< tag::tail_quantile< LeftRight > >
{
};
```

## Header `<boost/accumulators/statistics/tail_variate.hpp>`

```
namespace boost {
  namespace accumulators {
    template<typename VariateType, typename VariateTag, typename LeftRight>
      struct feature_of<tag::tail_variate< VariateType, VariateTag, LeftRight >>;
    template<typename LeftRight>
      struct feature_of<tag::tail_weights< LeftRight >>;
    namespace extract {
      extractor< tag::abstract_tail_variate > const tail_variate;
      extractor< tag::abstract_tail_weights > const tail_weights;
    }
    namespace impl {
    }
    namespace tag {
      struct abstract_tail_variate;
      struct abstract_tail_weights;
    }
  }
}
```

### Global `tail_variate`

`boost::accumulators::extract::tail_variate`

## Synopsis

```
// In header: <boost/accumulators/statistics/tail_variate.hpp>

extractor< tag::abstract_tail_variate > const tail_variate;
```

### Global `tail_weights`

`boost::accumulators::extract::tail_weights`

## Synopsis

```
// In header: <boost/accumulators/statistics/tail_variate.hpp>

extractor< tag::abstract_tail_weights > const tail_weights;
```

### Struct `abstract_tail_variate`

`boost::accumulators::tag::abstract_tail_variate`

## Synopsis

```
// In header: <boost/accumulators/statistics/tail_variate.hpp>

struct abstract_tail_variate : public boost::accumulators::depends_on<> {
};
```

## Struct `abstract_tail_weights`

`boost::accumulators::tag::abstract_tail_weights`

## Synopsis

```
// In header: <boost/accumulators/statistics/tail_variate.hpp>

struct abstract_tail_weights : public boost::accumulators::depends_on<> {
};
```

## Struct template `feature_of<tag::tail_variate< VariateType, VariateTag, LeftRight >>`

`boost::accumulators::feature_of<tag::tail_variate< VariateType, VariateTag, LeftRight >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/tail_variate.hpp>

template<typename VariateType, typename VariateTag, typename LeftRight>
struct feature_of<tag::tail_variate< VariateType, VariateTag, LeftRight >> :
    public boost::accumulators::feature_of< tag::abstract_tail_variate >
{
};
```

## Struct template `feature_of<tag::tail_weights< LeftRight >>`

`boost::accumulators::feature_of<tag::tail_weights< LeftRight >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/tail_variate.hpp>

template<typename LeftRight>
struct feature_of<tag::tail_weights< LeftRight >> {
    // types
    typedef tag::abstract_tail_weights type;
};
```

## Header `<boost/accumulators/statistics/tail_variate_means.hpp>`

```

namespace boost {
    namespace accumulators {
        template<typename LeftRight, typename VariateType, typename VariateTag>
            struct as_feature<tag::tail_variate_means< LeftRight, VariateType, VariateTag >(absolute)>;
        template<typename LeftRight, typename VariateType, typename VariateTag>
            struct as_feature<tag::tail_variate_means< LeftRight, VariateType, VariateTag >(relative)>;
        template<typename LeftRight, typename VariateType, typename VariateTag>
            struct as_weighted_feature<tag::absolute_tail_variate_means< LeftRight, VariateType, VariateTag >;
        template<typename LeftRight, typename VariateType, typename VariateTag>
            struct as_weighted_feature<tag::relative_tail_variate_means< LeftRight, VariateType, VariateTag >;
        template<typename LeftRight, typename VariateType, typename VariateTag>
            struct feature_of<tag::absolute_tail_variate_means< LeftRight, VariateType, VariateTag >;
        template<typename LeftRight, typename VariateType, typename VariateTag>
            struct feature_of<tag::absolute_weighted_tail_variate_means< LeftRight, VariateType, VariateTag >;
        template<typename LeftRight, typename VariateType, typename VariateTag>
            struct feature_of<tag::relative_tail_variate_means< LeftRight, VariateType, VariateTag >;
        template<typename LeftRight, typename VariateType, typename VariateTag>
            struct feature_of<tag::relative_weighted_tail_variate_means< LeftRight, VariateType, VariateTag >;
        namespace extract {
            extractor< tag::abstract_absolute_tail_variate_means > const tail_variate_means;
            extractor< tag::abstract_relative_tail_variate_means > const relative_tail_variate_means;
        }
        namespace impl {
        }
        namespace tag {
            struct abstract_absolute_tail_variate_means;
            struct abstract_relative_tail_variate_means;
        }
    }
}

```

### Global tail\_variate\_means

boost::accumulators::extract::tail\_variate\_means

## Synopsis

```

// In header: <boost/accumulators/statistics/tail_variate_means.hpp>

extractor< tag::abstract_absolute_tail_variate_means > const tail_variate_means;

```

### Global relative\_tail\_variate\_means

boost::accumulators::extract::relative\_tail\_variate\_means

## Synopsis

```

// In header: <boost/accumulators/statistics/tail_variate_means.hpp>

extractor< tag::abstract_relative_tail_variate_means > const relative_tail_variate_means;

```

## Struct `abstract_absolute_tail_variate_means`

`boost::accumulators::tag::abstract_absolute_tail_variate_means`

## Synopsis

```
// In header: <boost/accumulators/statistics/tail_variate_means.hpp>

struct abstract_absolute_tail_variate_means :
    public boost::accumulators::depends_on<>
{
};
```

## Struct `abstract_relative_tail_variate_means`

`boost::accumulators::tag::abstract_relative_tail_variate_means`

## Synopsis

```
// In header: <boost/accumulators/statistics/tail_variate_means.hpp>

struct abstract_relative_tail_variate_means :
    public boost::accumulators::depends_on<>
{
};
```

## Struct template `as_feature<tag::tail_variate_means< LeftRight, VariateType, VariateTag >(absolute)>`

`boost::accumulators::as_feature<tag::tail_variate_means< LeftRight, VariateType, VariateTag >(absolute)>`

## Synopsis

```
// In header: <boost/accumulators/statistics/tail_variate_means.hpp>

template<typename LeftRight, typename VariateType, typename VariateTag>
struct as_feature<tag::tail_variate_means< LeftRight, VariateType, VariateTag >(absolute)> {
    // types
    typedef tag::absolute_tail_variate_means< LeftRight, VariateType, VariateTag > type;
};
```

## Struct template `as_feature<tag::tail_variate_means< LeftRight, VariateType, VariateTag >(relative)>`

`boost::accumulators::as_feature<tag::tail_variate_means< LeftRight, VariateType, VariateTag >(relative)>`

## Synopsis

```
// In header: <boost/accumulators/statistics/tail_variate_means.hpp>

template<typename LeftRight, typename VariateType, typename VariateTag>
struct as_feature<tag::tail_variate_means< LeftRight, VariateType, VariateTag >(relative)> {
    // types
    typedef tag::relative_tail_variate_means< LeftRight, VariateType, VariateTag > type;
};
```

### Struct template `as_weighted_feature<tag::absolute_tail_variate_means< LeftRight, VariateType, VariateTag >>`

`boost::accumulators::as_weighted_feature<tag::absolute_tail_variate_means< LeftRight, VariateType, VariateTag >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/tail_variate_means.hpp>

template<typename LeftRight, typename VariateType, typename VariateTag>
struct as_weighted_feature<tag::absolute_tail_variate_means< LeftRight, VariateType, VariateTag >> {
    // types
    typedef tag::absolute_weighted_tail_variate_means< LeftRight, VariateType, VariateTag > type;
};
```

### Struct template `as_weighted_feature<tag::relative_tail_variate_means< LeftRight, VariateType, VariateTag >>`

`boost::accumulators::as_weighted_feature<tag::relative_tail_variate_means< LeftRight, VariateType, VariateTag >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/tail_variate_means.hpp>

template<typename LeftRight, typename VariateType, typename VariateTag>
struct as_weighted_feature<tag::relative_tail_variate_means< LeftRight, VariateType, VariateTag >> {
    // types
    typedef tag::relative_weighted_tail_variate_means< LeftRight, VariateType, VariateTag > type;
};
```

### Struct template `feature_of<tag::absolute_tail_variate_means< LeftRight, VariateType, VariateTag >>`

`boost::accumulators::feature_of<tag::absolute_tail_variate_means< LeftRight, VariateType, VariateTag >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/tail_variate_means.hpp>

template<typename LeftRight, typename VariateType, typename VariateTag>
struct feature_of<tag::absolute_tail_variate_means< LeftRight, VariateType, VariateTag >> : public
    boost::accumulators::feature_of< tag::abstract_absolute_tail_variate_means >
{
};
```

### Struct template feature\_of<tag::absolute\_weighted\_tail\_variate\_means< LeftRight, VariateType, VariateTag >>

boost::accumulators::feature\_of<tag::absolute\_weighted\_tail\_variate\_means< LeftRight, VariateType, VariateTag >>

## Synopsis

```
// In header: <boost/accumulators/statistics/tail_variate_means.hpp>

template<typename LeftRight, typename VariateType, typename VariateTag>
struct feature_of<tag::absolute_weighted_tail_variate_means< LeftRight, VariateType, VariateTag >> : public
    boost::accumulators::feature_of< tag::absolute_tail_variate_means< LeftRight, VariateType, VariateTag > >
{
};
```

### Struct template feature\_of<tag::relative\_tail\_variate\_means< LeftRight, VariateType, VariateTag >>

boost::accumulators::feature\_of<tag::relative\_tail\_variate\_means< LeftRight, VariateType, VariateTag >>

## Synopsis

```
// In header: <boost/accumulators/statistics/tail_variate_means.hpp>

template<typename LeftRight, typename VariateType, typename VariateTag>
struct feature_of<tag::relative_tail_variate_means< LeftRight, VariateType, VariateTag >> : public
    boost::accumulators::feature_of< tag::abstract_relative_tail_variate_means >
{
};
```

### Struct template feature\_of<tag::relative\_weighted\_tail\_variate\_means< LeftRight, VariateType, VariateTag >>

boost::accumulators::feature\_of<tag::relative\_weighted\_tail\_variate\_means< LeftRight, VariateType, VariateTag >>



## Synopsis

```
// In header: <boost/accumulators/statistics/tail_variate_means.hpp>

template<typename LeftRight, typename VariateType, typename VariateTag>
struct feature_of<tag::relative_weighted_tail_variate_means< LeftRight, VariateType, VariateTag >> : public boost::accumulators::feature_of< tag::relative_tail_variate_means< LeftRight, VariateType, VariateTag > >
{
};
```

Header <[boost/accumulators/statistics/times2\\_iterator.hpp](#)>

Header <[boost/accumulators/statistics/variance.hpp](#)>

```
namespace boost {
  namespace accumulators {
    template<> struct as_feature<tag::variance(immediate)>;
    template<> struct as_feature<tag::variance(lazy)>;
    template<> struct as_weighted_feature<tag::lazy_variance>;
    template<> struct as_weighted_feature<tag::variance>;
    template<> struct feature_of<tag::lazy_variance>;
    template<> struct feature_of<tag::lazy_weighted_variance>;
    template<> struct feature_of<tag::weighted_variance>;
    namespace extract {
      extractor< tag::lazy_variance > const lazy_variance;
      extractor< tag::variance > const variance;
    }
    namespace impl {
    }
    namespace tag {
      struct lazy_variance;
      struct variance;
    }
  }
}
```

### Global lazy\_variance

boost::accumulators::extract::lazy\_variance

## Synopsis

```
// In header: <boost/accumulators/statistics/variance.hpp>

extractor< tag::lazy_variance > const lazy_variance;
```

### Global variance

boost::accumulators::extract::variance

## Synopsis

```
// In header: <boost/accumulators/statistics/variance.hpp>

extractor< tag::variance > const variance;
```

### Struct lazy\_variance

boost::accumulators::tag::lazy\_variance

## Synopsis

```
// In header: <boost/accumulators/statistics/variance.hpp>

struct lazy_variance :
    public boost::accumulators::depends_on< moment< 2 >, mean >
{
};
```

### Struct variance

boost::accumulators::tag::variance

## Synopsis

```
// In header: <boost/accumulators/statistics/variance.hpp>

struct variance :
    public boost::accumulators::depends_on< count, immediate_mean >
{
};
```

### Struct as\_feature<tag::variance(immediate)>

boost::accumulators::as\_feature<tag::variance(immediate)>

## Synopsis

```
// In header: <boost/accumulators/statistics/variance.hpp>

struct as_feature<tag::variance(immediate)> {
    // types
    typedef tag::variance type;
};
```

### Struct as\_feature<tag::variance(lazy)>

boost::accumulators::as\_feature<tag::variance(lazy)>

## Synopsis

```
// In header: <boost/accumulators/statistics/variance.hpp>

struct as_feature<tag::variance(lazy)> {
    // types
    typedef tag::lazy_variance type;
};
```

### Struct `as_weighted_feature<tag::lazy_variance>`

`boost::accumulators::as_weighted_feature<tag::lazy_variance>`

## Synopsis

```
// In header: <boost/accumulators/statistics/variance.hpp>

struct as_weighted_feature<tag::lazy_variance> {
    // types
    typedef tag::lazy_weighted_variance type;
};
```

### Struct `as_weighted_feature<tag::variance>`

`boost::accumulators::as_weighted_feature<tag::variance>`

## Synopsis

```
// In header: <boost/accumulators/statistics/variance.hpp>

struct as_weighted_feature<tag::variance> {
    // types
    typedef tag::weighted_variance type;
};
```

### Struct `feature_of<tag::lazy_variance>`

`boost::accumulators::feature_of<tag::lazy_variance>`

## Synopsis

```
// In header: <boost/accumulators/statistics/variance.hpp>

struct feature_of<tag::lazy_variance> : public boost::accumulators::feature_of< tag::variance > {
};
```

### Struct `feature_of<tag::lazy_weighted_variance>`

`boost::accumulators::feature_of<tag::lazy_weighted_variance>`

## Synopsis

```
// In header: <boost/accumulators/statistics/variance.hpp>

struct feature_of<tag::lazy_weighted_variance> :
    public boost::accumulators::feature_of< tag::lazy_variance >
{
};
```

### Struct feature\_of<tag::weighted\_variance>

boost::accumulators::feature\_of<tag::weighted\_variance>

## Synopsis

```
// In header: <boost/accumulators/statistics/variance.hpp>

struct feature_of<tag::weighted_variance> : public boost::accumulators::feature_of< tag::vari-
ance > {
};
```

### Header <boost/accumulators/statistics/variates/covariate.hpp>

```
namespace boost {
    namespace accumulators {
        boost::parameter::keyword< tag::covariate1 > const covariate1;
        boost::parameter::keyword< tag::covariate2 > const covariate2;
        namespace tag {
            struct covariate1;
            struct covariate2;
        }
    }
}
```

### Struct covariate1

boost::accumulators::tag::covariate1

## Synopsis

```
// In header: <boost/accumulators/statistics/variates/covariate.hpp>

struct covariate1 {
};
```

### Struct covariate2

boost::accumulators::tag::covariate2

## Synopsis

```
// In header: <boost/accumulators/statistics/variates/covariate.hpp>

struct covariate2 {
};
```

### Global covariate1

boost::accumulators::covariate1

## Synopsis

```
// In header: <boost/accumulators/statistics/variates/covariate.hpp>

boost::parameter::keyword< tag::covariate1 > const covariate1;
```

### Global covariate2

boost::accumulators::covariate2

## Synopsis

```
// In header: <boost/accumulators/statistics/variates/covariate.hpp>

boost::parameter::keyword< tag::covariate2 > const covariate2;
```

## Header <boost/accumulators/statistics/weighted\_covariance.hpp>

### Global weighted\_covariance

boost::accumulators::extract::weighted\_covariance

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_covariance.hpp>

extractor< tag::abstract_covariance > const weighted_covariance;
```

## Header `<boost/accumulators/statistics/weighted_density.hpp>`

```
namespace boost {
  namespace accumulators {
    namespace extract {
      extractor< tag::density > const weighted_density;
    }
    namespace impl {
    }
    namespace tag {
      struct weighted_density;
    }
  }
}
```

### Global `weighted_density`

`boost::accumulators::extract::weighted_density`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_density.hpp>

extractor< tag::density > const weighted_density;
```

### Struct `weighted_density`

`boost::accumulators::tag::weighted_density`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_density.hpp>

struct weighted_density : public boost::accumulators::depends_on< count, sum_of_weights, min, max >,
                          public density_cache_size,
                          public density_num_bins
{
    // public data members
    static boost::parameter::keyword< density_cache_size > const cache_size;
    static boost::parameter::keyword< density_num_bins > const num_bins;
};
```

## Header `<boost/accumulators/statistics/weighted_extended_p_square.hpp>`

```
namespace boost {
    namespace accumulators {
        namespace extract {
            extractor< tag::weighted_extended_p_square > const weighted_extended_p_square;
        }
        namespace impl {
        }
        namespace tag {
            struct weighted_extended_p_square;
        }
    }
}
```

### Global `weighted_extended_p_square`

`boost::accumulators::extract::weighted_extended_p_square`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_extended_p_square.hpp>

extractor< tag::weighted_extended_p_square > const weighted_extended_p_square;
```

### Struct `weighted_extended_p_square`

`boost::accumulators::tag::weighted_extended_p_square`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_extended_p_square.hpp>

struct weighted_extended_p_square : public boost::accumulators::depends_on< count, sum_of_weights >,
                                   public extended_p_square_probabilities
{
    // types
    typedef accumulators::impl::weighted_extended_p_square_impl< mpl::_1, mpl::_2 > impl;
};
```

## Header `<boost/accumulators/statistics/weighted_kurtosis.hpp>`

```
namespace boost {
  namespace accumulators {
    namespace extract {
      extractor< tag::weighted_kurtosis > const weighted_kurtosis;
    }
    namespace impl {
    }
    namespace tag {
      struct weighted_kurtosis;
    }
  }
}
```

### Global `weighted_kurtosis`

`boost::accumulators::extract::weighted_kurtosis`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_kurtosis.hpp>

extractor< tag::weighted_kurtosis > const weighted_kurtosis;
```

### Struct `weighted_kurtosis`

`boost::accumulators::tag::weighted_kurtosis`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_kurtosis.hpp>

struct weighted_kurtosis : public boost::accumulators::depends_on< weighted_mean, weighted_moment< 2 >, weighted_moment< 3 >, weighted_moment< 4 > >
{
};
```



## Header `<boost/accumulators/statistics/weighted_mean.hpp>`

```
namespace boost {
  namespace accumulators {
    template<> struct as_feature<tag::weighted_mean(immediate)>;
    template<> struct as_feature<tag::weighted_mean(lazy)>;
    template<typename VariateType, typename VariateTag>
      struct as_feature<tag::weighted_mean_of_variates< VariateType, VariateTag >(immediate)>;
    template<typename VariateType, typename VariateTag>
      struct as_feature<tag::weighted_mean_of_variates< VariateType, VariateTag >(lazy)>;
    namespace extract {
      extractor< tag::mean > const weighted_mean;
    }
    namespace impl {
    }
    namespace tag {
      struct immediate_weighted_mean;
      struct weighted_mean;
    }
  }
}
```

### Global `weighted_mean`

`boost::accumulators::extract::weighted_mean`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_mean.hpp>

extractor< tag::mean > const weighted_mean;
```

### Struct `immediate_weighted_mean`

`boost::accumulators::tag::immediate_weighted_mean`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_mean.hpp>

struct immediate_weighted_mean :
  public boost::accumulators::depends_on< sum_of_weights >
{
};
```

### Struct `weighted_mean`

`boost::accumulators::tag::weighted_mean`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_mean.hpp>

struct weighted_mean :
    public boost::accumulators::depends_on< sum_of_weights, weighted_sum >
{
};
```

### Struct `as_feature<tag::weighted_mean(immediate)>`

`boost::accumulators::as_feature<tag::weighted_mean(immediate)>`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_mean.hpp>

struct as_feature<tag::weighted_mean(immediate)> {
    // types
    typedef tag::immediate_weighted_mean type;
};
```

### Struct `as_feature<tag::weighted_mean(lazy)>`

`boost::accumulators::as_feature<tag::weighted_mean(lazy)>`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_mean.hpp>

struct as_feature<tag::weighted_mean(lazy)> {
    // types
    typedef tag::weighted_mean type;
};
```

### Struct template `as_feature<tag::weighted_mean_of_variates< VariateType, VariateTag >(immediate)>`

`boost::accumulators::as_feature<tag::weighted_mean_of_variates< VariateType, VariateTag >(immediate)>`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_mean.hpp>

template<typename VariateType, typename VariateTag>
struct as_feature<tag::weighted_mean_of_variates< VariateType, VariateTag >(immediate)> {
    // types
    typedef tag::immediate_weighted_mean_of_variates< VariateType, VariateTag > type;
};
```

**Struct template `as_feature<tag::weighted_mean_of_variates<VariateType, VariateTag >(lazy)>`**

`boost::accumulators::as_feature<tag::weighted_mean_of_variates< VariateType, VariateTag >(lazy)>`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_mean.hpp>

template<typename VariateType, typename VariateTag>
struct as_feature<tag::weighted_mean_of_variates< VariateType, VariateTag >(lazy)> {
    // types
    typedef tag::weighted_mean_of_variates< VariateType, VariateTag > type;
};
```

**Header `<boost/accumulators/statistics/weighted_median.hpp>`**

```
namespace boost {
    namespace accumulators {
        template<> struct as_feature<tag::weighted_median(with_density)>;
        template<>
            struct as_feature<tag::weighted_median(with_p_square_cumulative_distribution)>;
        template<> struct as_feature<tag::weighted_median(with_p_square_quantile)>;
        namespace extract {
            extractor< tag::median > const weighted_median;
        }
        namespace impl {
        }
        namespace tag {
            struct weighted_median;
            struct with_density_weighted_median;
            struct with_p_square_cumulative_distribution_weighted_median;
        }
    }
}
```

**Global `weighted_median`**

`boost::accumulators::extract::weighted_median`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_median.hpp>

extractor< tag::median > const weighted_median;
```

**Struct `weighted_median`**

`boost::accumulators::tag::weighted_median`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_median.hpp>

struct weighted_median : public boost::accumulators::depends_on< weighted_p_square_quantile_for_med
dian >
{
};
```

### Struct with\_density\_weighted\_median

boost::accumulators::tag::with\_density\_weighted\_median

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_median.hpp>

struct with_density_weighted_median :
    public boost::accumulators::depends_on< count, weighted_density >
{
};
```

### Struct with\_p\_square\_cumulative\_distribution\_weighted\_median

boost::accumulators::tag::with\_p\_square\_cumulative\_distribution\_weighted\_median

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_median.hpp>

struct with_p_square_cumulative_distribution_weighted_median : public boost::accumulators::de
pends_on< weighted_p_square_cumulative_distribution >
{
};
```

### Struct as\_feature<tag::weighted\_median(with\_density)>

boost::accumulators::as\_feature<tag::weighted\_median(with\_density)>

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_median.hpp>

struct as_feature<tag::weighted_median(with_density)> {
    // types
    typedef tag::with_density_weighted_median type;
};
```

**Struct `as_feature<tag::weighted_median(with_p_square_cumulative_distribution)>`**`boost::accumulators::as_feature<tag::weighted_median(with_p_square_cumulative_distribution)>`**Synopsis**

```
// In header: <boost/accumulators/statistics/weighted_median.hpp>

struct as_feature<tag::weighted_median(with_p_square_cumulative_distribution)> {
    // types
    typedef tag::with_p_square_cumulative_distribution_weighted_median type;
};
```

**Struct `as_feature<tag::weighted_median(with_p_square_quantile)>`**`boost::accumulators::as_feature<tag::weighted_median(with_p_square_quantile)>`**Synopsis**

```
// In header: <boost/accumulators/statistics/weighted_median.hpp>

struct as_feature<tag::weighted_median(with_p_square_quantile)> {
    // types
    typedef tag::weighted_median type;
};
```

**Header `<boost/accumulators/statistics/weighted_moment.hpp>`****Header `<boost/accumulators/statistics/weighted_p_square_cumul_dist.hpp>`**

```
namespace boost {
    namespace accumulators {
        namespace extract {
            extractor< tag::weighted_p_square_cumulative_distribution > const weighted_p_square_cumulative_distribution;
        }
        namespace impl {
        }
        namespace tag {
            struct weighted_p_square_cumulative_distribution;
        }
    }
}
```

**Global `weighted_p_square_cumulative_distribution`**`boost::accumulators::extract::weighted_p_square_cumulative_distribution`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_p_square_cumul_dist.hpp>

extractor< tag::weighted_p_square_cumulative_distribution > const weighted_p_square_cumulative_distribution;
```

### Struct `weighted_p_square_cumulative_distribution`

`boost::accumulators::tag::weighted_p_square_cumulative_distribution`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_p_square_cumul_dist.hpp>

struct weighted_p_square_cumulative_distribution : public boost::accumulators::depends_on< count, sum_of_weights >,
                                                    public p_square_cumulative_distribution_num_cells
{
    // types
    typedef accumulators::impl::weighted_p_square_cumulative_distribution_impl< mpl::_1, mpl::_2 > impl;
};
```

### Header `<boost/accumulators/statistics/weighted_p_square_quantile.hpp>`

```
namespace boost {
    namespace accumulators {
        namespace extract {
            extractor< tag::weighted_p_square_quantile > const weighted_p_square_quantile;
            extractor< tag::weighted_p_square_quantile_for_median > const weighted_p_square_quantile_for_median;
        }
        namespace impl {
        }
        namespace tag {
            struct weighted_p_square_quantile;
            struct weighted_p_square_quantile_for_median;
        }
    }
}
```

### Global `weighted_p_square_quantile`

`boost::accumulators::extract::weighted_p_square_quantile`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_p_square_quantile.hpp>

extractor< tag::weighted_p_square_quantile > const weighted_p_square_quantile;
```

## Global weighted\_p\_square\_quantile\_for\_median

boost::accumulators::extract::weighted\_p\_square\_quantile\_for\_median

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_p_square_quantile.hpp>

extractor< tag::weighted_p_square_quantile_for_median > const weighted_p_square_quantile_for_med-
dian;
```

## Struct weighted\_p\_square\_quantile

boost::accumulators::tag::weighted\_p\_square\_quantile

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_p_square_quantile.hpp>

struct weighted_p_square_quantile :
    public boost::accumulators::depends_on< count, sum_of_weights >
{
    // types
    typedef accumulators::impl::weighted_p_square_quantile_impl< mpl::_1, mpl::_2, regular > impl;
};
```

## Struct weighted\_p\_square\_quantile\_for\_median

boost::accumulators::tag::weighted\_p\_square\_quantile\_for\_median

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_p_square_quantile.hpp>

struct weighted_p_square_quantile_for_median :
    public boost::accumulators::depends_on< count, sum_of_weights >
{
    // types
    typedef accumulators::impl::weighted_p_square_quantile_impl< mpl::_1, mpl::_2, for_median > impl;
};
```

**Header** <boost/accumulators/statistics/weighted\_peaks\_over\_threshold.hpp>

```

namespace boost {
    namespace accumulators {
        template<typename LeftRight>
        struct as_feature<tag::weighted_peaks_over_threshold< LeftRight >(with_threshold_probability)>;
        template<typename LeftRight>
        struct as_feature<tag::weighted_peaks_over_threshold< LeftRight >(with_threshold_value)>;
        namespace extract {
            extractor< tag::abstract_peaks_over_threshold > const weighted_peaks_over_threshold;
        }
        namespace impl {
        }
        namespace tag {
        }
    }
}

```

**Global weighted\_peaks\_over\_threshold**

boost::accumulators::extract::weighted\_peaks\_over\_threshold

**Synopsis**

```

// In header: <boost/accumulators/statistics/weighted_peaks_over_threshold.hpp>

extractor< tag::abstract_peaks_over_threshold > const weighted_peaks_over_threshold;

```

**Struct**      **template**      **as\_feature<tag::weighted\_peaks\_over\_threshold<**      **LeftRight**  
**>(with\_threshold\_probability)>**

boost::accumulators::as\_feature<tag::weighted\_peaks\_over\_threshold< LeftRight >(with\_threshold\_probability)>

**Synopsis**

```

// In header: <boost/accumulators/statistics/weighted_peaks_over_threshold.hpp>

template<typename LeftRight>
struct as_feature<tag::weighted_peaks_over_threshold< LeftRight >(with_threshold_probability)> {
    // types
    typedef tag::weighted_peaks_over_threshold_prob< LeftRight > type;
};

```

**Struct**      **template**      **as\_feature<tag::weighted\_peaks\_over\_threshold<**      **LeftRight**  
**>(with\_threshold\_value)>**

boost::accumulators::as\_feature<tag::weighted\_peaks\_over\_threshold< LeftRight >(with\_threshold\_value)>



## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_peaks_over_threshold.hpp>

template<typename LeftRight>
struct as_feature<tag::weighted_peaks_over_threshold< LeftRight >(with_threshold_value)> {
    // types
    typedef tag::weighted_peaks_over_threshold< LeftRight > type;
};
```

## Header <boost/accumulators/statistics/weighted\_skewness.hpp>

```
namespace boost {
    namespace accumulators {
        namespace extract {
            extractor< tag::weighted_skewness > const weighted_skewness;
        }
        namespace impl {
        }
        namespace tag {
            struct weighted_skewness;
        }
    }
}
```

### Global weighted\_skewness

boost::accumulators::extract::weighted\_skewness

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_skewness.hpp>

extractor< tag::weighted_skewness > const weighted_skewness;
```

### Struct weighted\_skewness

boost::accumulators::tag::weighted\_skewness

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_skewness.hpp>

struct weighted_skewness : public boost::accumulators::depends_on< weighted_mean, weighted_moment< 2 >, weighted_moment< 3 > >
{
};
```

## Header `<boost/accumulators/statistics/weighted_sum.hpp>`

```
namespace boost {
  namespace accumulators {
    template<typename VariateType, typename VariateTag>
      struct feature_of<tag::weighted_sum_of_variates< VariateType, VariateTag >>;
    namespace extract {
      extractor< tag::weighted_sum > const weighted_sum;
      extractor< tag::abstract_weighted_sum_of_variates > const weighted_sum_of_variates;
    }
    namespace impl {
    }
    namespace tag {
      struct abstract_weighted_sum_of_variates;
      struct weighted_sum;
    }
  }
}
```

### Global `weighted_sum`

`boost::accumulators::extract::weighted_sum`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_sum.hpp>

extractor< tag::weighted_sum > const weighted_sum;
```

### Global `weighted_sum_of_variates`

`boost::accumulators::extract::weighted_sum_of_variates`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_sum.hpp>

extractor< tag::abstract_weighted_sum_of_variates > const weighted_sum_of_variates;
```

### Struct `abstract_weighted_sum_of_variates`

`boost::accumulators::tag::abstract_weighted_sum_of_variates`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_sum.hpp>

struct abstract_weighted_sum_of_variates :
  public boost::accumulators::depends_on<>
{
};
```

## Struct `weighted_sum`

`boost::accumulators::tag::weighted_sum`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_sum.hpp>

struct weighted_sum : public boost::accumulators::depends_on<> {
};
```

## Struct template `feature_of<tag::weighted_sum_of_variates< VariateType, VariateTag >>`

`boost::accumulators::feature_of<tag::weighted_sum_of_variates< VariateType, VariateTag >>`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_sum.hpp>

template<typename VariateType, typename VariateTag>
struct feature_of<tag::weighted_sum_of_variates< VariateType, VariateTag >> : public boost::ac-
cumulators::feature_of< tag::abstract_weighted_sum_of_variates >
{
};
```

## Header `<boost/accumulators/statistics/weighted_sum_kahan.hpp>`

```
namespace boost {
  namespace accumulators {
    template<> struct as_feature<tag::weighted_sum(kahan)>;
    template<typename VariateType, typename VariateTag>
      struct feature_of<tag::weighted_sum_of_variates_kahan< VariateType, VariateTag >>;
    namespace extract {
      extractor< tag::weighted_sum_kahan > const weighted_sum_kahan;
      extractor< tag::abstract_weighted_sum_of_variates > const weighted_sum_of_variates_kahan;
    }
    namespace impl {
    }
    namespace tag {
      struct weighted_sum_kahan;
      template<typename VariateType, typename VariateTag>
        struct weighted_sum_of_variates_kahan;
    }
  }
}
```

## Global `weighted_sum_kahan`

`boost::accumulators::extract::weighted_sum_kahan`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_sum_kahan.hpp>

extractor< tag::weighted_sum_kahan > const weighted_sum_kahan;
```

### Global weighted\_sum\_of\_variates\_kahan

boost::accumulators::extract::weighted\_sum\_of\_variates\_kahan

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_sum_kahan.hpp>

extractor< tag::abstract_weighted_sum_of_variates > const weighted_sum_of_variates_kahan;
```

### Struct weighted\_sum\_kahan

boost::accumulators::tag::weighted\_sum\_kahan

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_sum_kahan.hpp>

struct weighted_sum_kahan : public boost::accumulators::depends_on<> {
};
```

### Struct template weighted\_sum\_of\_variates\_kahan

boost::accumulators::tag::weighted\_sum\_of\_variates\_kahan

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_sum_kahan.hpp>

template<typename VariateType, typename VariateTag>
struct weighted_sum_of_variates_kahan :
    public boost::accumulators::depends_on<>
{
};
```

### Struct as\_feature<tag::weighted\_sum(kahan)>

boost::accumulators::as\_feature<tag::weighted\_sum(kahan)>

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_sum_kahan.hpp>

struct as_feature<tag::weighted_sum(kahan)> {
    // types
    typedef tag::weighted_sum_kahan type;
};
```

### Struct template feature\_of<tag::weighted\_sum\_of\_variates\_kahan< VariateType, VariateTag >>

boost::accumulators::feature\_of<tag::weighted\_sum\_of\_variates\_kahan< VariateType, VariateTag >>

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_sum_kahan.hpp>

template<typename VariateType, typename VariateTag>
struct feature_of<tag::weighted_sum_of_variates_kahan< VariateType, VariateTag >> : public
    boost::accumulators::feature_of< tag::abstract_weighted_sum_of_variates >
{
};
```

### Header <boost/accumulators/statistics/weighted\_tail\_mean.hpp>

#### Global non\_coherent\_weighted\_tail\_mean

boost::accumulators::extract::non\_coherent\_weighted\_tail\_mean

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_tail_mean.hpp>

extractor< tag::abstract_non_coherent_tail_mean > const non_coherent_weighted_tail_mean;
```

### Header <boost/accumulators/statistics/weighted\_tail\_quantile.hpp>

#### Global weighted\_tail\_quantile

boost::accumulators::extract::weighted\_tail\_quantile

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_tail_quantile.hpp>

extractor< tag::quantile > const weighted_tail_quantile;
```

**Header <boost/accumulators/statistics/weighted\_tail\_variate\_means.hpp>**

```

namespace boost {
    namespace accumulators {
        template<typename LeftRight, typename VariateType, typename VariateTag>
        struct as_feature<tag::weighted_tail_variate_means< LeftRight, VariateType, VariateTag >(absolute)>;
        template<typename LeftRight, typename VariateType, typename VariateTag>
        struct as_feature<tag::weighted_tail_variate_means< LeftRight, VariateType, VariateTag >(relative)>;
        namespace extract {
            extractor< tag::abstract_absolute_tail_variate_means > const weighted_tail_variate_means;
            extractor< tag::abstract_relative_tail_variate_means > const relative_weighted_tail_variate_means;
        }
        namespace impl {
        }
        namespace tag {
        }
    }
    namespace numeric {
        namespace functional {
            template<typename T, typename U> struct multiply_and_promote_to_double;
        }
    }
}

```

**Global weighted\_tail\_variate\_means**

boost::accumulators::extract::weighted\_tail\_variate\_means

**Synopsis**

```

// In header: <boost/accumulators/statistics/weighted_tail_variate_means.hpp>

extractor< tag::abstract_absolute_tail_variate_means > const weighted_tail_variate_means;

```

**Global relative\_weighted\_tail\_variate\_means**

boost::accumulators::extract::relative\_weighted\_tail\_variate\_means

**Synopsis**

```

// In header: <boost/accumulators/statistics/weighted_tail_variate_means.hpp>

extractor< tag::abstract_relative_tail_variate_means > const relative_weighted_tail_variate_means;

```

**Struct template as\_feature<tag::weighted\_tail\_variate\_means< LeftRight, VariateType, VariateTag >(absolute)>**

boost::accumulators::as\_feature<tag::weighted\_tail\_variate\_means< LeftRight, VariateType, VariateTag >(absolute)>

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_tail_variate_means.hpp>

template<typename LeftRight, typename VariateType, typename VariateTag>
struct as_feature<tag::weighted_tail_variate_means< LeftRight, VariateType, VariateTag >(absolute)> {
    // types
    typedef tag::absolute_weighted_tail_variate_means< LeftRight, VariateType, VariateTag > type;
};
```

### Struct template as\_feature<tag::weighted\_tail\_variate\_means< LeftRight, VariateType, VariateTag >(relative)>

boost::accumulators::as\_feature<tag::weighted\_tail\_variate\_means< LeftRight, VariateType, VariateTag >(relative)>

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_tail_variate_means.hpp>

template<typename LeftRight, typename VariateType, typename VariateTag>
struct as_feature<tag::weighted_tail_variate_means< LeftRight, VariateType, VariateTag >(relative)> {
    // types
    typedef tag::relative_weighted_tail_variate_means< LeftRight, VariateType, VariateTag > type;
};
```

### Struct template multiply\_and\_promote\_to\_double

boost::numeric::functional::multiply\_and\_promote\_to\_double

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_tail_variate_means.hpp>

template<typename T, typename U>
struct multiply_and_promote_to_double :
    public functional::multiplies< T, double const >
{
};
```

## Header `<boost/accumulators/statistics/weighted_variance.hpp>`

```
namespace boost {
  namespace accumulators {
    template<> struct as_feature<tag::weighted_variance(immediate)>;
    template<> struct as_feature<tag::weighted_variance(lazy)>;
    namespace extract {
      extractor< tag::lazy_weighted_variance > const lazy_weighted_variance;
      extractor< tag::weighted_variance > const weighted_variance;
    }
    namespace impl {
    }
    namespace tag {
      struct lazy_weighted_variance;
      struct weighted_variance;
    }
  }
}
```

### Global lazy\_weighted\_variance

boost::accumulators::extract::lazy\_weighted\_variance

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_variance.hpp>

extractor< tag::lazy_weighted_variance > const lazy_weighted_variance;
```

### Global weighted\_variance

boost::accumulators::extract::weighted\_variance

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_variance.hpp>

extractor< tag::weighted_variance > const weighted_variance;
```

### Struct lazy\_weighted\_variance

boost::accumulators::tag::lazy\_weighted\_variance

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_variance.hpp>

struct lazy_weighted_variance : public boost::accumulators::depends_on< weighted_moment< 2 >,
weighted_mean >
{
};
```



## Struct `weighted_variance`

`boost::accumulators::tag::weighted_variance`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_variance.hpp>

struct weighted_variance :
    public boost::accumulators::depends_on< count, immediate_weighted_mean >
{
};
```

## Struct `as_feature<tag::weighted_variance(immediate)>`

`boost::accumulators::as_feature<tag::weighted_variance(immediate)>`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_variance.hpp>

struct as_feature<tag::weighted_variance(immediate)> {
    // types
    typedef tag::weighted_variance type;
};
```

## Struct `as_feature<tag::weighted_variance(lazy)>`

`boost::accumulators::as_feature<tag::weighted_variance(lazy)>`

## Synopsis

```
// In header: <boost/accumulators/statistics/weighted_variance.hpp>

struct as_feature<tag::weighted_variance(lazy)> {
    // types
    typedef tag::lazy_weighted_variance type;
};
```

**Header <boost/accumulators/statistics/with\_error.hpp>****Header <boost/accumulators/statistics\_fwd.hpp>**

```
namespace boost {
    namespace accumulators {
        struct absolute;
        struct for_median;
        struct immediate;
        struct kahan;
        struct lazy;
        struct left;
        struct linear;
        struct quadratic;
        struct regular;
        struct relative;
        struct right;
        template<BOOST_PP_ENUM_PARAMS_WITH_A_DEFAULT(BOOST_ACCUMULATORS_MAX_FEATURES, typename Feature, mpl::na) >
            struct stats;
        struct unweighted;
        struct weighted;
        struct with_density;
        template<BOOST_PP_ENUM_PARAMS_WITH_A_DEFAULT(BOOST_ACCUMULATORS_MAX_FEATURES, typename Feature, mpl::na) >
            struct with_error;
        struct with_p_square_cumulative_distribution;
        struct with_p_square_quantile;
        struct with_threshold_probability;
        struct with_threshold_value;
        namespace extract {
            extractor< tag::quantile > const quantile;
            extractor< tag::tail_mean > const tail_mean;
        }
        namespace impl {
            template<typename Sample, typename LeftRight>
                struct coherent_tail_mean_impl;
            template<typename Sample, typename VariateType, typename VariateTag>
                struct covariance_impl;
            template<typename Sample> struct density_impl;
            template<typename Sample, typename Variance> struct error_of_mean_impl;
            template<typename Sample> struct extended_p_square_impl;
            template<typename Sample, typename Impl1, typename Impl2>
                struct extended_p_square_quantile_impl;
            template<typename Sample, typename Tag = tag::sample>
                struct immediate_mean_impl;
            template<typename Sample, typename Weight, typename Tag>
                struct immediate_weighted_mean_impl;
            template<typename Sample> struct kurtosis_impl;
            template<typename Sample, typename MeanFeature> struct lazy_variance_impl;
            template<typename Sample, typename Weight, typename MeanFeature>
                struct lazy_weighted_variance_impl;
            template<typename Sample> struct max_impl;
            template<typename Sample, typename SumFeature = tag::sum> struct mean_impl;
            template<typename Sample> struct median_impl;
            template<typename Sample> struct min_impl;
            template<typename N, typename Sample> struct moment_impl;
            template<typename Sample, typename LeftRight>
                struct non_coherent_tail_mean_impl;
            template<typename Sample, typename Weight, typename LeftRight>
                struct non_coherent_weighted_tail_mean_impl;
            template<typename Sample> struct p_square_cumulative_distribution_impl;
```

```
template<typename Sample, typename Impl> struct p_square_quantile_impl;
template<typename Sample, typename LeftRight>
    struct peaks_over_threshold_prob_impl;
template<typename Sample, typename Impl, typename LeftRight>
    struct pot_quantile_impl;
template<typename Sample, typename Impl, typename LeftRight>
    struct pot_tail_mean_impl;
template<typename Sample> struct rolling_count_impl;
template<typename Sample> struct rolling_mean_impl;
template<typename Sample> struct rolling_sum_impl;
template<typename Sample> struct rolling_window_impl;
template<typename Sample> struct rolling_window_plus1_impl;
template<typename Sample> struct skewness_impl;
template<typename Sample, typename Tag = tag::sample> struct sum_impl;
template<typename Sample, typename Tag> struct sum_kahan_impl;
template<typename Sample, typename LeftRight> struct tail_impl;
template<typename Sample, typename LeftRight> struct tail_quantile_impl;
template<typename VariateType, typename VariateTag, typename LeftRight>
    struct tail_variate_impl;
template<typename Sample, typename Impl, typename LeftRight,
        typename VariateTag>
    struct tail_variate_means_impl;
template<typename Sample, typename MeanFeature, typename Tag>
    struct variance_impl;
template<typename Sample, typename Weight, typename VariateType,
        typename VariateTag>
    struct weighted_covariance_impl;
template<typename Sample, typename Weight> struct weighted_density_impl;
template<typename Sample, typename Weight>
    struct weighted_extended_p_square_impl;
template<typename Sample, typename Weight> struct weighted_kurtosis_impl;
template<typename Sample, typename Weight, typename Tag>
    struct weighted_mean_impl;
template<typename Sample> struct weighted_median_impl;
template<typename N, typename Sample, typename Weight>
    struct weighted_moment_impl;
template<typename Sample, typename Weight>
    struct weighted_p_square_cumulative_distribution_impl;
template<typename Sample, typename Weight, typename Impl>
    struct weighted_p_square_quantile_impl;
template<typename Sample, typename Weight, typename LeftRight>
    struct weighted_peaks_over_threshold_impl;
template<typename Sample, typename Weight, typename LeftRight>
    struct weighted_peaks_over_threshold_prob_impl;
template<typename Sample, typename Weight> struct weighted_skewness_impl;
template<typename Sample, typename Weight, typename Tag>
    struct weighted_sum_impl;
template<typename Sample, typename Weight, typename Tag>
    struct weighted_sum_kahan_impl;
template<typename Sample, typename Weight, typename LeftRight>
    struct weighted_tail_quantile_impl;
template<typename Sample, typename Weight, typename Impl,
        typename LeftRight, typename VariateType>
    struct weighted_tail_variate_means_impl;
template<typename Sample, typename Weight, typename MeanFeature,
        typename Tag>
    struct weighted_variance_impl;
template<typename Sample> struct with_density_median_impl;
template<typename Sample> struct with_density_weighted_median_impl;
template<typename Sample>
    struct with_p_square_cumulative_distribution_median_impl;
template<typename Sample, typename Weight>
    struct with_p_square_cumulative_distribution_weighted_median_impl;
```

```

}
namespace tag {
    template<typename LeftRight, typename VariateType, typename VariateTag>
        struct absolute_tail_variate_means;
    template<typename LeftRight, typename VariateType, typename VariateTag>
        struct absolute_weighted_tail_variate_means;
    template<typename LeftRight> struct coherent_tail_mean;
    template<typename VariateType, typename VariateTag> struct covariance;
    template<typename Feature> struct error_of;
    template<typename VariateType, typename VariateTag>
        struct immediate_mean_of_variates;
    template<typename VariateType, typename VariateTag>
        struct immediate_weighted_mean_of_variates;
    template<typename VariateType, typename VariateTag, typename LeftRight>
        struct left_tail_variate;
    template<typename VariateType, typename VariateTag> struct mean_of_variates;
    template<int N> struct moment;
    template<typename LeftRight> struct non_coherent_tail_mean;
    template<typename LeftRight> struct non_coherent_weighted_tail_mean;
    template<typename LeftRight> struct peaks_over_threshold;
    template<typename LeftRight> struct peaks_over_threshold_prob;
    template<typename LeftRight> struct pot_quantile;
    template<typename LeftRight> struct pot_quantile_prob;
    template<typename LeftRight> struct pot_tail_mean;
    template<typename LeftRight> struct pot_tail_mean_prob;
    struct quantile;
    template<typename LeftRight, typename VariateType, typename VariateTag>
        struct relative_tail_variate_means;
    template<typename LeftRight, typename VariateType, typename VariateTag>
        struct relative_weighted_tail_variate_means;
    template<typename VariateType, typename VariateTag, typename LeftRight>
        struct right_tail_variate;
    template<typename VariateType, typename VariateTag> struct sum_of_variates;
    template<typename VariateType, typename VariateTag>
        struct sum_of_variates_kahan;
    template<typename LeftRight> struct tail;
    struct tail_mean;
    template<typename LeftRight> struct tail_quantile;
    template<typename VariateType, typename VariateTag, typename LeftRight>
        struct tail_variate;
    template<typename LeftRight, typename VariateType, typename VariateTag>
        struct tail_variate_means;
    template<typename LeftRight> struct tail_weights;
    template<typename VariateType, typename VariateTag>
        struct weighted_covariance;
    template<typename VariateType, typename VariateTag>
        struct weighted_mean_of_variates;
    template<int N> struct weighted_moment;
    template<typename LeftRight> struct weighted_peaks_over_threshold;
    template<typename LeftRight> struct weighted_peaks_over_threshold_prob;
    template<typename LeftRight> struct weighted_pot_quantile;
    template<typename LeftRight> struct weighted_pot_quantile_prob;
    template<typename LeftRight> struct weighted_pot_tail_mean;
    template<typename LeftRight> struct weighted_pot_tail_mean_prob;
    template<typename VariateType, typename VariateTag>
        struct weighted_sum_of_variates;
    template<typename LeftRight> struct weighted_tail_quantile;
    template<typename LeftRight, typename VariateType, typename VariateTag>
        struct weighted_tail_variate_means;
}
}
}

```

## Global quantile

boost::accumulators::extract::quantile

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

extractor< tag::quantile > const quantile;
```

## Global tail\_mean

boost::accumulators::extract::tail\_mean

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

extractor< tag::tail_mean > const tail_mean;
```

## Struct template coherent\_tail\_mean\_impl

boost::accumulators::impl::coherent\_tail\_mean\_impl — Estimation of the coherent tail mean based on order statistics (for both left and right tails)

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename LeftRight>
struct coherent_tail_mean_impl : public accumulator_base {
    // construct/copy/destruct
    coherent_tail_mean_impl(dont_care);

    // public member functions
    template<typename Args> result_type result(Args const &) const;
};
```

## Description

The coherent tail mean  $\widehat{CTM}_{n,\alpha}(X)$  is equal to the non-coherent tail mean  $\widehat{NCTM}_{n,\alpha}(X)$  plus a correction term that ensures coherence in case of non-continuous distributions.

### Equation 15.

$$\widehat{CTM}_{n,\alpha}^{\text{right}}(X) = \widehat{NCTM}_{n,\alpha}^{\text{right}}(X) + \frac{1}{\lceil n(1-\alpha) \rceil} \hat{q}_{n,\alpha}(X) \left( 1 - \alpha - \frac{1}{n} \lceil n(1-\alpha) \rceil \right)$$

### Equation 16.

$$\widehat{CTM}_{n,\alpha}^{\text{left}}(X) = \widehat{NCTM}_{n,\alpha}^{\text{left}}(X) + \frac{1}{\lceil n\alpha \rceil} \hat{q}_{n,\alpha}(X) \left( \alpha - \frac{1}{n} \lceil n\alpha \rceil \right)$$

**coherent\_tail\_mean\_impl public construct/copy/destruct**

```
1. coherent_tail_mean_impl(dont_care);
```

**coherent\_tail\_mean\_impl public member functions**

```
1. template<typename Args> result_type result(Args const & args) const;
```

**Struct template covariance\_impl**

boost::accumulators::impl::covariance\_impl — Covariance Estimator.

**Synopsis**

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename VariateType, typename VariateTag>
struct covariance_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> covariance_impl(Args const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    result_type result(dont_care) const;
};
```

**Description**

An iterative Monte Carlo estimator for the covariance  $\text{Cov}(\mathbf{X}, \mathbf{X}')$ , where  $\mathbf{X}$  is a sample and  $\mathbf{X}'$  is a variate, is given by:

**Equation 17.**

$$\hat{c}_n = \frac{n-1}{n} \hat{c}_{n-1} + \frac{1}{n-1} (X_n - \hat{\mu}_n)(X'_n - \hat{\mu}'_n), \quad n \geq 2, \quad \hat{c}_1 = 0,$$

$\hat{\mu}_n$  and  $\hat{\mu}'_n$  being the means of the samples and variates.

**covariance\_impl public construct/copy/destruct**

```
1. template<typename Args> covariance_impl(Args const & args);
```

**covariance\_impl public member functions**

```
1. template<typename Args> void operator()(Args const & args);
```

```
2. result_type result(dont_care) const;
```

**Struct template density\_impl**

boost::accumulators::impl::density\_impl — Histogram density estimator.

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample>
struct density_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> density_impl(Args const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    template<typename Args> result_type result(Args const &) const;
};
```

### Description

The histogram density estimator returns a histogram of the sample distribution. The positions and sizes of the bins are determined using a specifiable number of cached samples (`cache_size`). The range between the minimum and the maximum of the cached samples is subdivided into a specifiable number of bins (`num_bins`) of same size. Additionally, an under- and an overflow bin is added to capture future under- and overflow samples. Once the bins are determined, the cached samples and all subsequent samples are added to the correct bins. At the end, a range of `std::pair` is return, where each pair contains the position of the bin (lower bound) and the samples count (normalized with the total number of samples).

#### `density_impl` public construct/copy/destruct

1. 

```
template<typename Args> density_impl(Args const & args);
```

#### `density_impl` public member functions

1. 

```
template<typename Args> void operator()(Args const & args);
```
2. 

```
template<typename Args> result_type result(Args const & args) const;
```

Requires:      The number of samples must meet or exceed the cache size

## Struct template `error_of_mean_impl`

`boost::accumulators::impl::error_of_mean_impl`

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Variance>
struct error_of_mean_impl : public accumulator_base {
    // construct/copy/destruct
    error_of_mean_impl(dont_care);

    // public member functions
    template<typename Args> result_type result(Args const &) const;
};
```

## Description

### `error_of_mean_impl` public construct/copy/destruct

```
1. error_of_mean_impl(dont_care);
```

### `error_of_mean_impl` public member functions

```
1. template<typename Args> result_type result(Args const & args) const;
```

## Struct template `extended_p_square_impl`

`boost::accumulators::impl::extended_p_square_impl` — Multiple quantile estimation with the extended  $P^2$  algorithm.

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample>
struct extended_p_square_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> extended_p_square_impl(Args const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    result_type result(dont_care) const;
};
```

## Description

Extended  $P^2$  algorithm for estimation of several quantiles without storing samples. Assume that  $m$  quantiles  $\xi_{p_1}, \dots, \xi_{p_m}$  are to be estimated. Instead of storing the whole sample cumulative distribution, the algorithm maintains only  $m+2$  principal markers and  $m+1$  middle markers, whose positions are updated with each sample and whose heights are adjusted (if necessary) using a piecewise-parabolic formula. The heights of these central markers are the current estimates of the quantiles and returned as an iterator range.

For further details, see

K. E. E. Raatikainen, Simultaneous estimation of several quantiles, *Simulation*, Volume 49, Number 4 (October), 1986, p. 159-164.

The extended  $P^2$  algorithm generalizes the  $P^2$  algorithm of

R. Jain and I. Chlamtac, The  $P^2$  algorithm for dynamic calculation of quantiles and histograms without storing observations, *Communications of the ACM*, Volume 28 (October), Number 10, 1985, p. 1076-1085.

### `extended_p_square_impl` public construct/copy/destruct

```
1. template<typename Args> extended_p_square_impl(Args const & args);
```

### `extended_p_square_impl` public member functions

```
1. template<typename Args> void operator()(Args const & args);
```



```
2. result_type result(dont_care) const;
```

## Struct template `extended_p_square_quantile_impl`

`boost::accumulators::impl::extended_p_square_quantile_impl` — Quantile estimation using the extended  $P^2$  algorithm for weighted and unweighted samples.

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Impl1, typename Impl2>
struct extended_p_square_quantile_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> extended_p_square_quantile_impl(Args const &);

    // public member functions
    template<typename Args> result_type result(Args const &) const;
};
```

### Description

Uses the quantile estimates calculated by the extended  $P^2$  algorithm to compute intermediate quantile estimates by means of quadratic interpolation.

#### `extended_p_square_quantile_impl` public construct/copy/destruct

```
1. template<typename Args> extended_p_square_quantile_impl(Args const & args);
```

#### `extended_p_square_quantile_impl` public member functions

```
1. template<typename Args> result_type result(Args const & args) const;
```

## Struct template `immediate_mean_impl`

`boost::accumulators::impl::immediate_mean_impl`

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Tag = tag::sample>
struct immediate_mean_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> immediate_mean_impl(Args const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    result_type result(dont_care) const;
};
```

## Description

### `immediate_mean_impl` public construct/copy/destruct

1. 

```
template<typename Args> immediate_mean_impl(Args const & args);
```

### `immediate_mean_impl` public member functions

1. 

```
template<typename Args> void operator()(Args const & args);
```
2. 

```
result_type result(dont_care) const;
```

## Struct template `immediate_weighted_mean_impl`

`boost::accumulators::impl::immediate_weighted_mean_impl`

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Weight, typename Tag>
struct immediate_weighted_mean_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> immediate_weighted_mean_impl(Args const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    result_type result(dont_care) const;
};
```

## Description

### `immediate_weighted_mean_impl` public construct/copy/destruct

1. 

```
template<typename Args> immediate_weighted_mean_impl(Args const & args);
```

### `immediate_weighted_mean_impl` public member functions

1. 

```
template<typename Args> void operator()(Args const & args);
```
2. 

```
result_type result(dont_care) const;
```

## Struct template `kurtosis_impl`

`boost::accumulators::impl::kurtosis_impl` — Kurtosis estimation.

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample>
struct kurtosis_impl : public accumulator_base {
    // construct/copy/destroy
    kurtosis_impl(dont_care);

    // public member functions
    template<typename Args> result_type result(Args const &) const;
};
```

### Description

The kurtosis of a sample distribution is defined as the ratio of the 4th central moment and the square of the 2nd central moment (the variance) of the samples, minus 3. The term  $-3$  is added in order to ensure that the normal distribution has zero kurtosis. The kurtosis can also be expressed by the simple moments:

### Equation 18.

$$\hat{g}_2 = \frac{\hat{m}_n^{(4)} - 4\hat{m}_n^{(3)}\hat{\mu}_n + 6\hat{m}_n^{(2)}\hat{\mu}_n^2 - 3\hat{\mu}_n^4}{\left(\hat{m}_n^{(2)} - \hat{\mu}_n^2\right)^2} - 3,$$

where  $\hat{m}_n^{(i)}$  are the  $i$ -th moment and  $\hat{\mu}_n$  the mean (first moment) of the  $n$  samples.

**kurtosis\_impl public construct/copy/destroy**

1. `kurtosis_impl(dont_care);`

**kurtosis\_impl public member functions**

1. `template<typename Args> result_type result(Args const & args) const;`

## Struct template lazy\_variance\_impl

boost::accumulators::impl::lazy\_variance\_impl — Lazy calculation of variance.

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename MeanFeature>
struct lazy_variance_impl : public accumulator_base {
    // construct/copy/destroy
    lazy_variance_impl(dont_care);

    // public member functions
    template<typename Args> result_type result(Args const &) const;
};
```

## Description

Default sample variance implementation based on the second moment  $M_n^{(2)}$  moment<2>, mean and count.

### Equation 19.

$$\sigma_n^2 = M_n^{(2)} - \mu_n^2.$$

where

### Equation 20.

$$\mu_n = \frac{1}{n} \sum_{i=1}^n x_i.$$

is the estimate of the sample mean and  $n$  is the number of samples.

**lazy\_variance\_impl public construct/copy/destruct**

```
1. lazy_variance_impl(dont_care);
```

**lazy\_variance\_impl public member functions**

```
1. template<typename Args> result_type result(Args const & args) const;
```

## Struct template lazy\_weighted\_variance\_impl

boost::accumulators::impl::lazy\_weighted\_variance\_impl — Lazy calculation of variance of weighted samples.

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Weight, typename MeanFeature>
struct lazy_weighted_variance_impl : public accumulator_base {
    // construct/copy/destruct
    lazy_weighted_variance_impl(dont_care);

    // public member functions
    template<typename Args> result_type result(Args const &) const;
};
```

## Description

The default implementation of the variance of weighted samples is based on the second moment  $\hat{m}_n^{(2)}$  (weighted\_moment<2>) and the mean  $\hat{\mu}_n$  (weighted\_mean):

### Equation 21.

$$\hat{\sigma}_n^2 = \hat{m}_n^{(2)} - \hat{\mu}_n^2,$$

where  $n$  is the number of samples.

**lazy\_weighted\_variance\_impl public construct/copy/destruct**

```
1. lazy_weighted_variance_impl(dont_care);
```

**lazy\_weighted\_variance\_impl public member functions**

```
1. template<typename Args> result_type result(Args const & args) const;
```

**Struct template max\_impl**

boost::accumulators::impl::max\_impl

**Synopsis**

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample>
struct max_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> max_impl(Args const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    result_type result(dont_care) const;
};
```

**Description****max\_impl public construct/copy/destruct**

```
1. template<typename Args> max_impl(Args const & args);
```

**max\_impl public member functions**

```
1. template<typename Args> void operator()(Args const & args);
```

```
2. result_type result(dont_care) const;
```

**Struct template mean\_impl**

boost::accumulators::impl::mean\_impl

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename SumFeature = tag::sum>
struct mean_impl : public accumulator_base {
    // construct/copy/destruct
    mean_impl(dont_care);

    // public member functions
    template<typename Args> result_type result(Args const &) const;
};
```

### Description

**mean\_impl** public construct/copy/destruct

```
1. mean_impl(dont_care);
```

**mean\_impl** public member functions

```
1. template<typename Args> result_type result(Args const & args) const;
```

## Struct template median\_impl

boost::accumulators::impl::median\_impl — Median estimation based on the  $P^2$  quantile estimator.

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample>
struct median_impl : public accumulator_base {
    // construct/copy/destruct
    median_impl(dont_care);

    // public member functions
    template<typename Args> result_type result(Args const &) const;
};
```

### Description

The  $P^2$  algorithm is invoked with a quantile probability of 0.5.

**median\_impl** public construct/copy/destruct

```
1. median_impl(dont_care);
```

**median\_impl** public member functions

```
1. template<typename Args> result_type result(Args const & args) const;
```

## Struct template min\_impl

boost::accumulators::impl::min\_impl

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample>
struct min_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> min_impl(Args const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    result_type result(dont_care) const;
};
```

### Description

**min\_impl public construct/copy/destruct**

1. 

```
template<typename Args> min_impl(Args const & args);
```

**min\_impl public member functions**

1. 

```
template<typename Args> void operator()(Args const & args);
```

2. 

```
result_type result(dont_care) const;
```

## Struct template moment\_impl

boost::accumulators::impl::moment\_impl

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename N, typename Sample>
struct moment_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> moment_impl(Args const &);

    // public member functions
    BOOST_MPL_ASSERT_RELATION(N::value, 0);
    template<typename Args> void operator()(Args const &);
    template<typename Args> result_type result(Args const &) const;
};
```

## Description

**moment\_impl** public construct/copy/destruct

```
1. template<typename Args> moment_impl(Args const & args);
```

**moment\_impl** public member functions

```
1. BOOST_MPL_ASSERT_RELATION(N::value, 0);
```

```
2. template<typename Args> void operator()(Args const & args);
```

```
3. template<typename Args> result_type result(Args const & args) const;
```

## Struct template non\_coherent\_tail\_mean\_impl

boost::accumulators::impl::non\_coherent\_tail\_mean\_impl — Estimation of the (non-coherent) tail mean based on order statistics (for both left and right tails)

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename LeftRight>
struct non_coherent_tail_mean_impl : public accumulator_base {
    // construct/copy/destruct
    non_coherent_tail_mean_impl(dont_care);

    // public member functions
    template<typename Args> result_type result(Args const &) const;
};
```

## Description

An estimation of the non-coherent tail mean  $\widehat{NCTM}_{n,\alpha}(X)$  is given by the mean of the  $\lceil n\alpha \rceil$  smallest samples (left tail) or the mean of the  $\lceil n(1-\alpha) \rceil$  largest samples (right tail),  $n$  being the total number of samples and  $\alpha$  the quantile level:

### Equation 22.

$$\widehat{NCTM}_{n,\alpha}^{\text{right}}(X) = \frac{1}{\lceil n(1-\alpha) \rceil} \sum_{i=\lceil n\alpha \rceil}^n X_{i:n}$$

### Equation 23.

$$\widehat{NCTM}_{n,\alpha}^{\text{left}}(X) = \frac{1}{\lceil n\alpha \rceil} \sum_{i=1}^{\lceil n\alpha \rceil} X_{i:n}$$

It thus requires the caching of at least the  $\lceil n\alpha \rceil$  smallest or the  $\lceil n(1-\alpha) \rceil$  largest samples.



**non\_coherent\_tail\_mean\_impl public construct/copy/destruct**

```
1. non_coherent_tail_mean_impl(dont_care);
```

**non\_coherent\_tail\_mean\_impl public member functions**

```
1. template<typename Args> result_type result(Args const & args) const;
```

## Struct template non\_coherent\_weighted\_tail\_mean\_impl

boost::accumulators::impl::non\_coherent\_weighted\_tail\_mean\_impl — Estimation of the (non-coherent) weighted tail mean based on order statistics (for both left and right tails)

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Weight, typename LeftRight>
struct non_coherent_weighted_tail_mean_impl : public accumulator_base {
    // construct/copy/destruct
    non_coherent_weighted_tail_mean_impl(dont_care);

    // public member functions
    template<typename Args> result_type result(Args const &) const;
};
```

## Description

An estimation of the non-coherent, weighted tail mean  $\widehat{NCTM}_{n,\alpha}(X)$  is given by the weighted mean of the

### Equation 24.

$$\lambda = \inf \left\{ l \left| \frac{1}{\bar{w}_n} \sum_{i=1}^l w_i \geq \alpha \right. \right\}$$

smallest samples (left tail) or the weighted mean of the

### Equation 25.

$$n+1-\rho = n+1 - \sup \left\{ r \left| \frac{1}{\bar{w}_n} \sum_{i=r}^n w_i \geq (1-\alpha) \right. \right\}$$

largest samples (right tail) above a quantile  $\hat{q}_\alpha$  of level  $\alpha$ ,  $n$  being the total number of sample and  $\bar{w}_n$  the sum of all  $n$  weights:

### Equation 26.

$$\widehat{NCTM}_{n,\alpha}^{\text{left}}(X) = \frac{\sum_{i=1}^{\lambda} w_i X_{i:n}}{\sum_{i=1}^{\lambda} w_i},$$

### Equation 27.

$$\widehat{NCTM}_{n,\alpha}^{\text{right}}(X) = \frac{\sum_{i=\rho}^n w_i X_{i:n}}{\sum_{i=\rho}^n w_i}.$$

**non\_coherent\_weighted\_tail\_mean\_impl public construct/copy/destruct**

```
1. non_coherent_weighted_tail_mean_impl(dont_care);
```

**non\_coherent\_weighted\_tail\_mean\_impl public member functions**

```
1. template<typename Args> result_type result(Args const & args) const;
```

**Struct template p\_square\_cumulative\_distribution\_impl**

boost::accumulators::impl::p\_square\_cumulative\_distribution\_impl — Histogram calculation of the cumulative distribution with the  $P^2$  algorithm.

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample>
struct p_square_cumulative_distribution_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> p_square_cumulative_distribution_impl(Args const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    template<typename Args> result_type result(Args const &) const;
};
```

**Description**

A histogram of the sample cumulative distribution is computed dynamically without storing samples based on the  $P^2$  algorithm. The returned histogram has a specifiable amount (num\_cells) equiprobable (and not equal-sized) cells.

For further details, see

R. Jain and I. Chlamtac, The  $P^2$  algorithm for dynamic calculation of quantiles and histograms without storing observations, Communications of the ACM, Volume 28 (October), Number 10, 1985, p. 1076-1085.

**p\_square\_cumulative\_distribution\_impl public construct/copy/destruct**

```
1. template<typename Args>
    p_square_cumulative_distribution_impl(Args const & args);
```

**p\_square\_cumulative\_distribution\_impl public member functions**

```
1. template<typename Args> void operator()(Args const & args);
```

```
2. template<typename Args> result_type result(Args const & args) const;
```

## Struct template `p_square_quantile_impl`

`boost::accumulators::impl::p_square_quantile_impl` — Single quantile estimation with the  $P^2$  algorithm.

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Impl>
struct p_square_quantile_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> p_square_quantile_impl(Args const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    result_type result(dont_care) const;
};
```

### Description

The  $P^2$  algorithm estimates a quantile dynamically without storing samples. Instead of storing the whole sample cumulative distribution, only five points (markers) are stored. The heights of these markers are the minimum and the maximum of the samples and the current estimates of the  $(p/2)$ -,  $p$ - and  $(1+p)/2$ -quantiles. Their positions are equal to the number of samples that are smaller or equal to the markers. Each time a new samples is recorded, the positions of the markers are updated and if necessary their heights are adjusted using a piecewise- parabolic formula.

For further details, see

R. Jain and I. Chlamtac, The  $P^2$  algorithm for dynamic calculation of quantiles and histograms without storing observations, Communications of the ACM, Volume 28 (October), Number 10, 1985, p. 1076-1085.

### `p_square_quantile_impl` public construct/copy/destruct

```
1. template<typename Args> p_square_quantile_impl(Args const & args);
```

### `p_square_quantile_impl` public member functions

```
1. template<typename Args> void operator()(Args const & args);
```

```
2. result_type result(dont_care) const;
```

## Struct template `peaks_over_threshold_prob_impl`

`boost::accumulators::impl::peaks_over_threshold_prob_impl` — Peaks over Threshold Method for Quantile and Tail Mean Estimation.

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename LeftRight>
struct peaks_over_threshold_prob_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> peaks_over_threshold_prob_impl(Args const &);

    // public member functions
    void operator()(dont_care);
    template<typename Args> result_type result(Args const &) const;
};
```

### Description

#### See Also:

[peaks\\_over\\_threshold\\_impl](#)

**peaks\_over\_threshold\_prob\_impl public construct/copy/destruct**

1. 

```
template<typename Args> peaks_over_threshold_prob_impl(Args const & args);
```

**peaks\_over\_threshold\_prob\_impl public member functions**

1. 

```
void operator()(dont_care);
```
2. 

```
template<typename Args> result_type result(Args const & args) const;
```

## Struct template pot\_quantile\_impl

boost::accumulators::impl::pot\_quantile\_impl — Quantile Estimation based on Peaks over Threshold Method (for both left and right tails)

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Impl, typename LeftRight>
struct pot_quantile_impl : public accumulator_base {
    // construct/copy/destruct
    pot_quantile_impl(dont_care);

    // public member functions
    template<typename Args> result_type result(Args const &) const;
};
```

### Description

Computes an estimate

**Equation 28.**

$$\hat{q}_\alpha = \hat{u} + \frac{\hat{\beta}}{\xi} [(1 - \alpha)^{-\xi} - 1]$$

for a right or left extreme quantile,  $\hat{u}$ ,  $\hat{\beta}$  and  $\xi$  being the parameters of the generalized Pareto distribution that approximates the right tail of the distribution (or the mirrored left tail, in case the left tail is used). In the latter case, the result is mirrored back, yielding the correct result.

**pot\_quantile\_impl** public construct/copy/destruct

```
1. pot_quantile_impl(dont_care);
```

**pot\_quantile\_impl** public member functions

```
1. template<typename Args> result_type result(Args const & args) const;
```

**Struct template pot\_tail\_mean\_impl**

boost::accumulators::impl::pot\_tail\_mean\_impl — Estimation of the (coherent) tail mean based on the peaks over threshold method (for both left and right tails)

**Synopsis**

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Impl, typename LeftRight>
struct pot_tail_mean_impl : public accumulator_base {
    // construct/copy/destruct
    pot_tail_mean_impl(dont_care);

    // public member functions
    template<typename Args> result_type result(Args const &) const;
};
```

**Description**

Computes an estimate for the (coherent) tail mean

**Equation 29.**

$$\widehat{CTM}_\alpha = \hat{q}_\alpha - \frac{\hat{\beta}}{\xi - 1} (1 - \alpha)^{-\xi},$$

where  $\hat{u}$ ,  $\hat{\beta}$  and  $\xi$  are the parameters of the generalized Pareto distribution that approximates the right tail of the distribution (or the mirrored left tail, in case the left tail is used). In the latter case, the result is mirrored back, yielding the correct result.

**pot\_tail\_mean\_impl** public construct/copy/destruct

```
1. pot_tail_mean_impl(dont_care);
```

**pot\_tail\_mean\_impl public member functions**

1. 

```
template<typename Args> result_type result(Args const & args) const;
```

**Struct template rolling\_count\_impl**

boost::accumulators::impl::rolling\_count\_impl

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample>
struct rolling_count_impl : public accumulator_base {
    // construct/copy/destruct
    rolling_count_impl(dont_care);

    // public member functions
    template<typename Args> result_type result(Args const &) const;
};
```

**Description****rolling\_count\_impl public construct/copy/destruct**

1. 

```
rolling_count_impl(dont_care);
```

**rolling\_count\_impl public member functions**

1. 

```
template<typename Args> result_type result(Args const & args) const;
```

**Struct template rolling\_mean\_impl**

boost::accumulators::impl::rolling\_mean\_impl

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample>
struct rolling_mean_impl {
};
```

**Struct template rolling\_sum\_impl**

boost::accumulators::impl::rolling\_sum\_impl

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample>
struct rolling_sum_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> rolling_sum_impl(Args const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    template<typename Args> result_type result(Args const &) const;
};
```

### Description

**rolling\_sum\_impl public construct/copy/destruct**

1. 

```
template<typename Args> rolling_sum_impl(Args const & args);
```

**rolling\_sum\_impl public member functions**

1. 

```
template<typename Args> void operator()(Args const & args);
```
2. 

```
template<typename Args> result_type result(Args const &) const;
```

## Struct template rolling\_window\_impl

boost::accumulators::impl::rolling\_window\_impl

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample>
struct rolling_window_impl : public accumulator_base {
    // construct/copy/destruct
    rolling_window_impl(dont_care);

    // public member functions
    template<typename Args> result_type result(Args const &) const;
};
```

### Description

**rolling\_window\_impl public construct/copy/destruct**

1. 

```
rolling_window_impl(dont_care);
```

**rolling\_window\_impl public member functions**

1. 

```
template<typename Args> result_type result(Args const & args) const;
```

**Struct template rolling\_window\_plus1\_impl**

boost::accumulators::impl::rolling\_window\_plus1\_impl

**Synopsis**

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample>
struct rolling_window_plus1_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> rolling_window_plus1_impl(Args const &);
    rolling_window_plus1_impl(rolling_window_plus1_impl const &);
    rolling_window_plus1_impl & operator=(rolling_window_plus1_impl const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    bool full() const;
    result_type result(dont_care) const;
};
```

**Description****rolling\_window\_plus1\_impl public construct/copy/destruct**

1. 

```
template<typename Args> rolling_window_plus1_impl(Args const & args);
```
2. 

```
rolling_window_plus1_impl(rolling_window_plus1_impl const & that);
```
3. 

```
rolling_window_plus1_impl & operator=(rolling_window_plus1_impl const & that);
```

**rolling\_window\_plus1\_impl public member functions**

1. 

```
template<typename Args> void operator()(Args const & args);
```
2. 

```
bool full() const;
```
3. 

```
result_type result(dont_care) const;
```

**Struct template skewness\_impl**

boost::accumulators::impl::skewness\_impl — Skewness estimation.



## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample>
struct skewness_impl : public accumulator_base {
    // construct/copy/destruct
    skewness_impl(dont_care);

    // public member functions
    template<typename Args> result_type result(Args const &) const;
};
```

### Description

The skewness of a sample distribution is defined as the ratio of the 3rd central moment and the  $3/2$ -th power of the 2nd central moment (the variance) of the samples. The skewness can also be expressed by the simple moments:

### Equation 30.

$$\hat{g}_1 = \frac{\hat{m}_n^{(3)} - 3\hat{m}_n^{(2)}\hat{\mu}_n + 2\hat{\mu}_n^3}{\left(\hat{m}_n^{(2)} - \hat{\mu}_n^2\right)^{3/2}}$$

where  $\hat{m}_n^{(i)}$  are the  $i$ -th moment and  $\hat{\mu}_n$  the mean (first moment) of the  $n$  samples.

#### **skewness\_impl** public construct/copy/destruct

1. `skewness_impl(dont_care);`

#### **skewness\_impl** public member functions

1. `template<typename Args> result_type result(Args const & args) const;`

## Struct template sum\_impl

`boost::accumulators::impl::sum_impl`

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Tag = tag::sample>
struct sum_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> sum_impl(Args const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    result_type result(dont_care) const;
};
```

## Description

### **sum\_impl public construct/copy/destruct**

```
1. template<typename Args> sum_impl(Args const & args);
```

### **sum\_impl public member functions**

```
1. template<typename Args> void operator()(Args const & args);

2. result_type result(dont_care) const;
```

## Struct template sum\_kahan\_impl

boost::accumulators::impl::sum\_kahan\_impl

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Tag>
struct sum_kahan_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> sum_kahan_impl(Args const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    result_type result(dont_care) const;
};
```

## Description

### **sum\_kahan\_impl public construct/copy/destruct**

```
1. template<typename Args> sum_kahan_impl(Args const & args);
```

Kahan summation algorithm.

The Kahan summation algorithm reduces the numerical error obtained with standard sequential sum.

### **sum\_kahan\_impl public member functions**

```
1. template<typename Args> void operator()(Args const & args);

2. result_type result(dont_care) const;
```

## Struct template tail\_impl

boost::accumulators::impl::tail\_impl

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename LeftRight>
struct tail_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> tail_impl(Args const &);
    tail_impl(tail_impl const &);

    // public member functions
    BOOST_MPL_ASSERT((mpl::or_< is_same< LeftRight, right >, is_same< LeftRight, left > >));
    template<typename Args> void operator()(Args const &);
    result_type result(dont_care) const;

    // private member functions
    template<typename Args> void assign(Args const &, std::size_t);
};
```

### Description

#### **tail\_impl public construct/copy/destruct**

1. `template<typename Args> tail_impl(Args const & args);`

2. `tail_impl(tail_impl const & that);`

#### **tail\_impl public member functions**

1. `BOOST_MPL_ASSERT((mpl::or_< is_same< LeftRight, right >, is_same< LeftRight, left > >));`

2. `template<typename Args> void operator()(Args const & args);`

3. `result_type result(dont_care) const;`

#### **tail\_impl private member functions**

1. `template<typename Args> void assign(Args const & args, std::size_t index);`

### Struct template tail\_quantile\_impl

`boost::accumulators::impl::tail_quantile_impl` — Tail quantile estimation based on order statistics (for both left and right tails)

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename LeftRight>
struct tail_quantile_impl : public accumulator_base {
    // construct/copy/destruct
    tail_quantile_impl(dont_care);

    // public member functions
    template<typename Args> result_type result(Args const &) const;
};
```

### Description

The estimation of a tail quantile  $\hat{q}$  with level  $\alpha$  based on order statistics requires the caching of at least the  $\lceil n\alpha \rceil$  smallest or the  $\lceil n(1-\alpha) \rceil$  largest samples,  $n$  being the total number of samples. The largest of the  $\lceil n\alpha \rceil$  smallest samples or the smallest of the  $\lceil n(1-\alpha) \rceil$  largest samples provides an estimate for the quantile:

### Equation 31.

$$\hat{q}_{n,\alpha} = X_{[\alpha n]:n}$$

**tail\_quantile\_impl public construct/copy/destruct**

```
1. tail_quantile_impl(dont_care);
```

**tail\_quantile\_impl public member functions**

```
1. template<typename Args> result_type result(Args const & args) const;
```

## Strut template tail\_variate\_impl

boost::accumulators::impl::tail\_variate\_impl

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename VariateType, typename VariateTag, typename LeftRight>
struct tail_variate_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> tail_variate_impl(Args const &);

    // public member functions
    template<typename Args> void assign(Args const &, std::size_t);
    template<typename Args> result_type result(Args const &) const;

    // private member functions
    template<typename TailRng> result_type do_result(TailRng const &) const;
};
```

## Description

**tail\_variate\_impl** public construct/copy/destruct

```
1. template<typename Args> tail_variate_impl(Args const & args);
```

**tail\_variate\_impl** public member functions

```
1. template<typename Args> void assign(Args const & args, std::size_t index);
```

```
2. template<typename Args> result_type result(Args const & args) const;
```

**tail\_variate\_impl** private member functions

```
1. template<typename TailRng> result_type do_result(TailRng const & rng) const;
```

## Struct template tail\_variate\_means\_impl

boost::accumulators::impl::tail\_variate\_means\_impl — Estimation of the absolute and relative tail variate means (for both left and right tails)

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Impl, typename LeftRight,
        typename VariateTag>
struct tail_variate_means_impl : public accumulator_base {
    // construct/copy/destruct
    tail_variate_means_impl(dont_care);

    // public member functions
    template<typename Args> result_type result(Args const &) const;
};
```

## Description

For all  $j$ -th variates associated to the  $\lceil n(1-\alpha) \rceil$  largest samples (or the  $\lceil n(1-\alpha) \rceil$  smallest samples in case of the left tail), the absolute tail means  $\widehat{ATM}_{n,\alpha}(X, j)$  are computed and returned as an iterator range. Alternatively, the relative tail means  $\widehat{RTM}_{n,\alpha}(X, j)$  are returned, which are the absolute tail means normalized with the (non-coherent) sample tail mean  $\widehat{NCTM}_{n,\alpha}(X)$ .

### Equation 32.

$$\widehat{ATM}_{n,\alpha}^{\text{right}}(X, j) = \frac{1}{\lceil n(1-\alpha) \rceil} \sum_{i=\lceil n\alpha \rceil}^n \xi_{j,i}$$

### Equation 33.

$$\widehat{ATM}_{n,\alpha}^{\text{left}}(X, j) = \frac{1}{\lceil n\alpha \rceil} \sum_{i=1}^{\lceil n\alpha \rceil} \xi_{j,i}$$

**Equation 34.**

$$\widehat{RTM}_{n,\alpha}^{right}(X,j) = \frac{\sum_{i=[n\alpha]}^n \xi_{j,i}}{[n(1-\alpha)] \widehat{NCTM}_{n,\alpha}^{right}(X)}$$

**Equation 35.**

$$\widehat{RTM}_{n,\alpha}^{left}(X,j) = \frac{\sum_{i=1}^{[n\alpha]} \xi_{j,i}}{[n\alpha] \widehat{NCTM}_{n,\alpha}^{left}(X)}$$

**tail\_variate\_means\_impl public construct/copy/destruct**

```
1. tail_variate_means_impl(dont_care);
```

**tail\_variate\_means\_impl public member functions**

```
1. template<typename Args> result_type result(Args const & args) const;
```

**Struct template variance\_impl**

boost::accumulators::impl::variance\_impl — Iterative calculation of variance.

**Synopsis**

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename MeanFeature, typename Tag>
struct variance_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> variance_impl(Args const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    result_type result(dont_care) const;
};
```

**Description**Iterative calculation of sample variance  $\sigma_n^2$  according to the formula**Equation 36.**

$$\sigma_n^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \mu_n)^2 = \frac{n-1}{n} \sigma_{n-1}^2 + \frac{1}{n-1} (x_n - \mu_n)^2.$$

where

**Equation 37.**

$$\mu_n = \frac{1}{n} \sum_{i=1}^n x_i.$$

is the estimate of the sample mean and  $n$  is the number of samples.

Note that the sample variance is not defined for  $n \leq 1$ .

A simplification can be obtained by the approximate recursion

### Equation 38.

$$\sigma_n^2 \approx \frac{n-1}{n} \sigma_{n-1}^2 + \frac{1}{n} (x_n - \mu_n)^2.$$

because the difference

### Equation 39.

$$\left( \frac{1}{n-1} - \frac{1}{n} \right) (x_n - \mu_n)^2 = \frac{1}{n(n-1)} (x_n - \mu_n)^2.$$

converges to zero as  $n \rightarrow \infty$ . However, for small  $n$  the difference can be non-negligible.

**variance\_impl public construct/copy/destruct**

```
1. template<typename Args> variance_impl(Args const & args);
```

**variance\_impl public member functions**

```
1. template<typename Args> void operator()(Args const & args);
```

```
2. result_type result(dont_care) const;
```

## Struct template weighted\_covariance\_impl

boost::accumulators::impl::weighted\_covariance\_impl — Weighted Covariance Estimator.

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Weight, typename VariateType,
        typename VariateTag>
struct weighted_covariance_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> weighted_covariance_impl(Args const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    result_type result(dont_care) const;
};
```

## Description

An iterative Monte Carlo estimator for the weighted covariance  $\text{Cov}(\mathbf{X}, \mathbf{X}')$ , where  $\mathbf{X}$  is a sample and  $\mathbf{X}'$  a variate, is given by:

**Equation 40.**

$$\hat{c}_n = \frac{\bar{w}_n - w_n}{\bar{w}_n} \hat{c}_{n-1} + \frac{w_n}{\bar{w}_n - w_n} (X_n - \hat{\mu}_n)(X'_n - \hat{\mu}'_n), \quad n \geq 2, \quad \hat{c}_1 = 0,$$

$\hat{\mu}_n$  and  $\hat{\mu}'_n$  being the weighted means of the samples and variates and  $\bar{w}_n$  the sum of the  $n$  first weights  $w_i$ .

**weighted\_covariance\_impl public construct/copy/destruct**

1. 

```
template<typename Args> weighted_covariance_impl(Args const & args);
```

**weighted\_covariance\_impl public member functions**

1. 

```
template<typename Args> void operator()(Args const & args);
```
2. 

```
result_type result(dont_care) const;
```

**Struct template weighted\_density\_impl**

boost::accumulators::impl::weighted\_density\_impl — Histogram density estimator for weighted samples.

**Synopsis**

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Weight>
struct weighted_density_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> weighted_density_impl(Args const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    template<typename Args> result_type result(Args const &) const;
};
```

**Description**

The histogram density estimator returns a histogram of the sample distribution. The positions and sizes of the bins are determined using a specifiable number of cached samples (`cache_size`). The range between the minimum and the maximum of the cached samples is subdivided into a specifiable number of bins (`num_bins`) of same size. Additionally, an under- and an overflow bin is added to capture future under- and overflow samples. Once the bins are determined, the cached samples and all subsequent samples are added to the correct bins. At the end, a range of `std::pair` is returned, where each pair contains the position of the bin (lower bound) and the sum of the weights (normalized with the sum of all weights).

**weighted\_density\_impl public construct/copy/destruct**

1. 

```
template<typename Args> weighted_density_impl(Args const & args);
```

**weighted\_density\_impl public member functions**

1. 

```
template<typename Args> void operator()(Args const & args);
```



2. `template<typename Args> result_type result(Args const & args) const;`

## Struct template `weighted_extended_p_square_impl`

`boost::accumulators::impl::weighted_extended_p_square_impl` — Multiple quantile estimation with the extended  $P^2$  algorithm for weighted samples.

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Weight>
struct weighted_extended_p_square_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> weighted_extended_p_square_impl(Args const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    result_type result(dont_care) const;
};
```

## Description

This version of the extended  $P^2$  algorithm extends the extended  $P^2$  algorithm to support weighted samples. The extended  $P^2$  algorithm dynamically estimates several quantiles without storing samples. Assume that  $m$  quantiles  $\xi_{p_1}, \dots, \xi_{p_m}$  are to be estimated. Instead of storing the whole sample cumulative distribution, the algorithm maintains only  $m + 2$  principal markers and  $m + 1$  middle markers, whose positions are updated with each sample and whose heights are adjusted (if necessary) using a piecewise-parabolic formula. The heights of the principal markers are the current estimates of the quantiles and are returned as an iterator range.

For further details, see

K. E. E. Raatikainen, Simultaneous estimation of several quantiles, *Simulation*, Volume 49, Number 4 (October), 1986, p. 159-164.

The extended  $P^2$  algorithm generalizes the  $P^2$  algorithm of

R. Jain and I. Chlamtac, The  $P^2$  algorithm for dynamic calculation of quantiles and histograms without storing observations, *Communications of the ACM*, Volume 28 (October), Number 10, 1985, p. 1076-1085.

**`weighted_extended_p_square_impl` public construct/copy/destruct**

1. `template<typename Args> weighted_extended_p_square_impl(Args const & args);`

**`weighted_extended_p_square_impl` public member functions**

1. `template<typename Args> void operator()(Args const & args);`

2. `result_type result(dont_care) const;`

## Struct template `weighted_kurtosis_impl`

`boost::accumulators::impl::weighted_kurtosis_impl` — Kurtosis estimation for weighted samples.

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Weight>
struct weighted_kurtosis_impl : public accumulator_base {
    // construct/copy/destroy
    weighted_kurtosis_impl(dont_care);

    // public member functions
    template<typename Args> result_type result(Args const &) const;
};
```

### Description

The kurtosis of a sample distribution is defined as the ratio of the 4th central moment and the square of the 2nd central moment (the variance) of the samples, minus 3. The term  $-3$  is added in order to ensure that the normal distribution has zero kurtosis. The kurtosis can also be expressed by the simple moments:

### Equation 41.

$$\hat{g}_2 = \frac{\hat{m}_n^{(4)} - 4\hat{m}_n^{(3)}\hat{\mu}_n + 6\hat{m}_n^{(2)}\hat{\mu}_n^2 - 3\hat{\mu}_n^4}{(\hat{m}_n^{(2)} - \hat{\mu}_n^2)^2} - 3,$$

where  $\hat{m}_n^{(i)}$  are the  $i$ -th moment and  $\hat{\mu}_n$  the mean (first moment) of the  $n$  samples.

The kurtosis estimator for weighted samples is formally identical to the estimator for unweighted samples, except that the weighted counterparts of all measures it depends on are to be taken.

#### **weighted\_kurtosis\_impl public construct/copy/destroy**

```
1. weighted_kurtosis_impl(dont_care);
```

#### **weighted\_kurtosis\_impl public member functions**

```
1. template<typename Args> result_type result(Args const & args) const;
```

## Struct template weighted\_mean\_impl

boost::accumulators::impl::weighted\_mean\_impl

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Weight, typename Tag>
struct weighted_mean_impl : public accumulator_base {
    // construct/copy/destroy
    weighted_mean_impl(dont_care);

    // public member functions
    template<typename Args> result_type result(Args const &) const;
};
```

## Description

**weighted\_mean\_impl** public construct/copy/destruct

```
1. weighted_mean_impl(dont_care);
```

**weighted\_mean\_impl** public member functions

```
1. template<typename Args> result_type result(Args const & args) const;
```

## Struct template weighted\_median\_impl

boost::accumulators::impl::weighted\_median\_impl — Median estimation for weighted samples based on the  $P^2$  quantile estimator.

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample>
struct weighted_median_impl : public accumulator_base {
    // construct/copy/destruct
    weighted_median_impl(dont_care);

    // public member functions
    template<typename Args> result_type result(Args const &) const;
};
```

## Description

The  $P^2$  algorithm for weighted samples is invoked with a quantile probability of 0.5.

**weighted\_median\_impl** public construct/copy/destruct

```
1. weighted_median_impl(dont_care);
```

**weighted\_median\_impl** public member functions

```
1. template<typename Args> result_type result(Args const & args) const;
```

## Struct template weighted\_moment\_impl

boost::accumulators::impl::weighted\_moment\_impl

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename N, typename Sample, typename Weight>
struct weighted_moment_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> weighted_moment_impl(Args const &);

    // public member functions
    BOOST_MPL_ASSERT_RELATION(N::value, 0);
    template<typename Args> void operator()(Args const &);
    template<typename Args> result_type result(Args const &) const;
};
```

### Description

**weighted\_moment\_impl public construct/copy/destruct**

1. 

```
template<typename Args> weighted_moment_impl(Args const & args);
```

**weighted\_moment\_impl public member functions**

1. 

```
BOOST_MPL_ASSERT_RELATION(N::value, 0);
```
2. 

```
template<typename Args> void operator()(Args const & args);
```
3. 

```
template<typename Args> result_type result(Args const & args) const;
```

## Struct template weighted\_p\_square\_cumulative\_distribution\_impl

boost::accumulators::impl::weighted\_p\_square\_cumulative\_distribution\_impl — Histogram calculation of the cumulative distribution with the  $P^2$  algorithm for weighted samples.

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Weight>
struct weighted_p_square_cumulative_distribution_impl :
    public accumulator_base
{
    // construct/copy/destruct
    template<typename Args>
        weighted_p_square_cumulative_distribution_impl(Args const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    template<typename Args> result_type result(Args const &) const;
};
```

## Description

A histogram of the sample cumulative distribution is computed dynamically without storing samples based on the  $P^2$  algorithm for weighted samples. The returned histogram has a specifiable amount (num\_cells) equiprobable (and not equal-sized) cells.

Note that applying importance sampling results in regions to be more and other regions to be less accurately estimated than without importance sampling, i.e., with unweighted samples.

For further details, see

R. Jain and I. Chlamtac, The  $P^2$  algorithm for dynamic calculation of quantiles and histograms without storing observations, Communications of the ACM, Volume 28 (October), Number 10, 1985, p. 1076-1085.

**weighted\_p\_square\_cumulative\_distribution\_impl public construct/copy/destruct**

```
1. template<typename Args>
   weighted_p_square_cumulative_distribution_impl(Args const & args);
```

**weighted\_p\_square\_cumulative\_distribution\_impl public member functions**

```
1. template<typename Args> void operator()(Args const & args);

2. template<typename Args> result_type result(Args const & args) const;
```

## Struct template weighted\_p\_square\_quantile\_impl

boost::accumulators::impl::weighted\_p\_square\_quantile\_impl — Single quantile estimation with the  $P^2$  algorithm for weighted samples.

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Weight, typename Impl>
struct weighted_p_square_quantile_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> weighted_p_square_quantile_impl(Args const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    result_type result(dont_care) const;
};
```

## Description

This version of the  $P^2$  algorithm extends the  $P^2$  algorithm to support weighted samples. The  $P^2$  algorithm estimates a quantile dynamically without storing samples. Instead of storing the whole sample cumulative distribution, only five points (markers) are stored. The heights of these markers are the minimum and the maximum of the samples and the current estimates of the  $(p/2)$ -,  $p$  - and  $(1+p)/2$  -quantiles. Their positions are equal to the number of samples that are smaller or equal to the markers. Each time a new sample is added, the positions of the markers are updated and if necessary their heights are adjusted using a piecewise- parabolic formula.

For further details, see

R. Jain and I. Chlamtac, The  $P^2$  algorithm for dynamic calculation of quantiles and histograms without storing observations, Communications of the ACM, Volume 28 (October), Number 10, 1985, p. 1076-1085.

**weighted\_p\_square\_quantile\_impl public construct/copy/destruct**

```
1. template<typename Args> weighted_p_square_quantile_impl(Args const & args);
```

**weighted\_p\_square\_quantile\_impl public member functions**

```
1. template<typename Args> void operator()(Args const & args);
```

```
2. result_type result(dont_care) const;
```

**Struct template weighted\_peaks\_over\_threshold\_impl**

boost::accumulators::impl::weighted\_peaks\_over\_threshold\_impl — Weighted Peaks over Threshold Method for Weighted Quantile and Weighted Tail Mean Estimation.

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Weight, typename LeftRight>
struct weighted_peaks_over_threshold_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> weighted_peaks_over_threshold_impl(Args const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    template<typename Args> result_type result(Args const &) const;
};
```

**Description****See Also:**

[peaks\\_over\\_threshold\\_impl](#)

**weighted\_peaks\_over\_threshold\_impl public construct/copy/destruct**

```
1. template<typename Args> weighted_peaks_over_threshold_impl(Args const & args);
```

**weighted\_peaks\_over\_threshold\_impl public member functions**

```
1. template<typename Args> void operator()(Args const & args);
```

```
2. template<typename Args> result_type result(Args const & args) const;
```

## Struct template `weighted_peaks_over_threshold_prob_impl`

`boost::accumulators::impl::weighted_peaks_over_threshold_prob_impl` — Peaks over Threshold Method for Quantile and Tail Mean Estimation.

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Weight, typename LeftRight>
struct weighted_peaks_over_threshold_prob_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args>
        weighted_peaks_over_threshold_prob_impl(Args const &);

    // public member functions
    void operator()(dont_care);
    template<typename Args> result_type result(Args const &) const;
};
```

### Description

#### See Also:

[weighted\\_peaks\\_over\\_threshold\\_impl](#)

**`weighted_peaks_over_threshold_prob_impl` public construct/copy/destruct**

1. 

```
template<typename Args>
    weighted_peaks_over_threshold_prob_impl(Args const & args);
```

**`weighted_peaks_over_threshold_prob_impl` public member functions**

1. 

```
void operator()(dont_care);
```
2. 

```
template<typename Args> result_type result(Args const & args) const;
```

## Struct template `weighted_skewness_impl`

`boost::accumulators::impl::weighted_skewness_impl` — Skewness estimation for weighted samples.

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Weight>
struct weighted_skewness_impl : public accumulator_base {
    // construct/copy/destruct
    weighted_skewness_impl(dont_care);

    // public member functions
    template<typename Args> result_type result(Args const &) const;
};
```

## Description

The skewness of a sample distribution is defined as the ratio of the 3rd central moment and the  $3/2$ -th power of the 2nd central moment (the variance) of the samples. The skewness can also be expressed by the sample moments:

### Equation 42.

$$\hat{g}_1 = \frac{\hat{m}_n^{(3)} - 3\hat{m}_n^{(2)}\hat{\mu}_n + 2\hat{\mu}_n^3}{\left(\hat{m}_n^{(2)} - \hat{\mu}_n^2\right)^{3/2}}$$

where  $\hat{m}_n^{(i)}$  are the  $i$ -th moment and  $\hat{\mu}_n$  the mean (first moment) of the  $n$  samples.

The skewness estimator for weighted samples is formally identical to the estimator for unweighted samples, except that the weighted counterparts of all measures it depends on are to be taken.

**weighted\_skewness\_impl** public construct/copy/destruct

```
1. weighted_skewness_impl(dont_care);
```

**weighted\_skewness\_impl** public member functions

```
1. template<typename Args> result_type result(Args const & args) const;
```

## Struct template weighted\_sum\_impl

boost::accumulators::impl::weighted\_sum\_impl

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Weight, typename Tag>
struct weighted_sum_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> weighted_sum_impl(Args const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    result_type result(dont_care) const;
};
```

## Description

**weighted\_sum\_impl** public construct/copy/destruct

```
1. template<typename Args> weighted_sum_impl(Args const & args);
```

**weighted\_sum\_impl** public member functions

```
1. template<typename Args> void operator()(Args const & args);
```



```
2. result_type result(dont_care) const;
```

## Struct template `weighted_sum_kahan_impl`

`boost::accumulators::impl::weighted_sum_kahan_impl`

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Weight, typename Tag>
struct weighted_sum_kahan_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> weighted_sum_kahan_impl(Args const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    result_type result(dont_care) const;
};
```

### Description

**`weighted_sum_kahan_impl` public construct/copy/destruct**

```
1. template<typename Args> weighted_sum_kahan_impl(Args const & args);
```

**`weighted_sum_kahan_impl` public member functions**

```
1. template<typename Args> void operator()(Args const & args);
```

```
2. result_type result(dont_care) const;
```

## Struct template `weighted_tail_quantile_impl`

`boost::accumulators::impl::weighted_tail_quantile_impl` — Tail quantile estimation based on order statistics of weighted samples (for both left and right tails)

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Weight, typename LeftRight>
struct weighted_tail_quantile_impl : public accumulator_base {
    // construct/copy/destruct
    weighted_tail_quantile_impl(dont_care);

    // public member functions
    template<typename Args> result_type result(Args const &) const;
};
```

## Description

An estimator  $\hat{q}$  of tail quantiles with level  $\alpha$  based on order statistics  $X_{1:n} \leq X_{2:n} \leq \dots \leq X_{n:n}$  of weighted samples are given by  $X_{\lambda:n}$  (left tail) and  $X_{\rho:n}$  (right tail), where

### Equation 43.

$$\lambda = \inf \left\{ l \left| \frac{1}{\bar{w}_n} \sum_{i=1}^l w_i \geq \alpha \right. \right\}$$

and

### Equation 44.

$$\rho = \sup \left\{ r \left| \frac{1}{\bar{w}_n} \sum_{i=r}^n w_i \geq (1 - \alpha) \right. \right\},$$

$n$  being the number of samples and  $\bar{w}_n$  the sum of all weights.

**weighted\_tail\_quantile\_impl** public construct/copy/destroy

```
1. weighted_tail_quantile_impl(dont_care);
```

**weighted\_tail\_quantile\_impl** public member functions

```
1. template<typename Args> result_type result(Args const & args) const;
```

## Struct template weighted\_tail\_variate\_means\_impl

boost::accumulators::impl::weighted\_tail\_variate\_means\_impl — Estimation of the absolute and relative weighted tail variate means (for both left and right tails)

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Weight, typename Impl, typename LeftRight,
        typename VariateType>
struct weighted_tail_variate_means_impl : public accumulator_base {
    // construct/copy/destroy
    weighted_tail_variate_means_impl(dont_care);

    // public member functions
    template<typename Args> result_type result(Args const &) const;
};
```

## Description

For all  $j$ -th variates associated to the

**Equation 45.**

$$\lambda = \inf \left\{ l \left| \frac{1}{\bar{w}_n} \sum_{i=1}^l w_i \geq \alpha \right. \right\}$$

smallest samples (left tail) or the weighted mean of the

**Equation 46.**

$$n+1-\rho = n+1-\sup \left\{ r \left| \frac{1}{\bar{w}_n} \sum_{i=r}^n w_i \geq (1-\alpha) \right. \right\}$$

largest samples (right tail), the absolute weighted tail means  $\widehat{ATM}_{n,\alpha}(X, j)$  are computed and returned as an iterator range. Alternatively, the relative weighted tail means  $\widehat{RTM}_{n,\alpha}(X, j)$  are returned, which are the absolute weighted tail means normalized with the weighted (non-coherent) sample tail mean  $\widehat{NCTM}_{n,\alpha}(X)$ .

**Equation 47.**

$$\widehat{ATM}_{n,\alpha}^{\text{right}}(X, j) = \frac{1}{\sum_{i=\rho}^n w_i} \sum_{i=\rho}^n w_i \xi_{j,i}$$

**Equation 48.**

$$\widehat{ATM}_{n,\alpha}^{\text{left}}(X, j) = \frac{1}{\sum_{i=1}^{\lambda} w_i} \sum_{i=1}^{\lambda} w_i \xi_{j,i}$$

**Equation 49.**

$$\widehat{RTM}_{n,\alpha}^{\text{right}}(X, j) = \frac{\sum_{i=\rho}^n w_i \xi_{j,i}}{\sum_{i=\rho}^n w_i \widehat{NCTM}_{n,\alpha}^{\text{right}}(X)}$$

**Equation 50.**

$$\widehat{RTM}_{n,\alpha}^{\text{left}}(X, j) = \frac{\sum_{i=1}^{\lambda} w_i \xi_{j,i}}{\sum_{i=1}^{\lambda} w_i \widehat{NCTM}_{n,\alpha}^{\text{left}}(X)}$$

`weighted_tail_variate_means_impl` **public construct/copy/destruct**

```
1. weighted_tail_variate_means_impl(dont_care);
```

`weighted_tail_variate_means_impl` **public member functions**

```
1. template<typename Args> result_type result(Args const & args) const;
```

**Struct template `weighted_variance_impl`**

`boost::accumulators::impl::weighted_variance_impl` — Iterative calculation of variance of weighted samples.

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Weight, typename MeanFeature, typename Tag>
struct weighted_variance_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> weighted_variance_impl(Args const &);

    // public member functions
    template<typename Args> void operator()(Args const &);
    result_type result(dont_care) const;
};
```

### Description

Iterative calculation of variance of weighted samples:

#### Equation 51.

$$\hat{\sigma}_n^2 = \frac{\bar{w}_n - w_n}{\bar{w}_n} \hat{\sigma}_{n-1}^2 + \frac{w_n}{\bar{w}_n} (X_n - \hat{\mu}_n)^2, \quad n \geq 2, \quad \hat{\sigma}_0^2 = 0.$$

where  $\bar{w}_n$  is the sum of the  $n$  weights  $w_i$  and  $\hat{\mu}_n$  the estimate of the mean of the weighted samples. Note that the sample variance is not defined for  $n \leq 1$ .

#### weighted\_variance\_impl public construct/copy/destruct

1. 

```
template<typename Args> weighted_variance_impl(Args const & args);
```

#### weighted\_variance\_impl public member functions

1. 

```
template<typename Args> void operator()(Args const & args);
```
2. 

```
result_type result(dont_care) const;
```

### Struct template with\_density\_median\_impl

boost::accumulators::impl::with\_density\_median\_impl — Median estimation based on the density estimator.

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample>
struct with_density_median_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> with_density_median_impl(Args const &);

    // public member functions
    void operator()(dont_care);
    template<typename Args> result_type result(Args const &) const;
};
```

## Description

The algorithm determines the bin in which the  $0.5 * cnt$ -th sample lies, *cnt* being the total number of samples. It returns the approximate horizontal position of this sample, based on a linear interpolation inside the bin.

**with\_density\_median\_impl public construct/copy/destruct**

1. 

```
template<typename Args> with_density_median_impl(Args const & args);
```

**with\_density\_median\_impl public member functions**

1. 

```
void operator()(dont_care);
```
2. 

```
template<typename Args> result_type result(Args const & args) const;
```

## Struct template with\_density\_weighted\_median\_impl

boost::accumulators::impl::with\_density\_weighted\_median\_impl — Median estimation for weighted samples based on the density estimator.

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample>
struct with_density_weighted_median_impl : public accumulator_base {
    // construct/copy/destruct
    template<typename Args> with_density_weighted_median_impl(Args const &);

    // public member functions
    void operator()(dont_care);
    template<typename Args> result_type result(Args const &) const;
};
```

## Description

The algorithm determines the bin in which the  $0.5 * cnt$ -th sample lies, *cnt* being the total number of samples. It returns the approximate horizontal position of this sample, based on a linear interpolation inside the bin.

**with\_density\_weighted\_median\_impl public construct/copy/destruct**

1. 

```
template<typename Args> with_density_weighted_median_impl(Args const & args);
```

**with\_density\_weighted\_median\_impl public member functions**

1. 

```
void operator()(dont_care);
```
2. 

```
template<typename Args> result_type result(Args const & args) const;
```

## Struct template with\_p\_square\_cumulative\_distribution\_median\_impl

boost::accumulators::impl::with\_p\_square\_cumulative\_distribution\_median\_impl — Median estimation based on the  $P^2$  cumulative distribution estimator.

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample>
struct with_p_square_cumulative_distribution_median_impl :
    public accumulator_base
{
    // construct/copy/destroy
    with_p_square_cumulative_distribution_median_impl(dont_care);

    // public member functions
    void operator()(dont_care);
    template<typename Args> result_type result(Args const &) const;
};
```

## Description

The algorithm determines the first (leftmost) bin with a height exceeding 0.5. It returns the approximate horizontal position of where the cumulative distribution equals 0.5, based on a linear interpolation inside the bin.

**with\_p\_square\_cumulative\_distribution\_median\_impl public construct/copy/destroy**

```
1. with_p_square_cumulative_distribution_median_impl(dont_care);
```

**with\_p\_square\_cumulative\_distribution\_median\_impl public member functions**

```
1. void operator()(dont_care);
```

```
2. template<typename Args> result_type result(Args const & args) const;
```

## Struct template with\_p\_square\_cumulative\_distribution\_weighted\_median\_impl

boost::accumulators::impl::with\_p\_square\_cumulative\_distribution\_weighted\_median\_impl — Median estimation for weighted samples based on the  $P^2$  cumulative distribution estimator.

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Sample, typename Weight>
struct with_p_square_cumulative_distribution_weighted_median_impl :
    public accumulator_base
{
    // construct/copy/destruct
    with_p_square_cumulative_distribution_weighted_median_impl(dont_care);

    // public member functions
    void operator()(dont_care);
    template<typename Args> result_type result(Args const &) const;
};
```

### Description

The algorithm determines the first (leftmost) bin with a height exceeding 0.5. It returns the approximate horizontal position of where the cumulative distribution equals 0.5, based on a linear interpolation inside the bin.

**with\_p\_square\_cumulative\_distribution\_weighted\_median\_impl public construct/copy/destruct**

1. 

```
with_p_square_cumulative_distribution_weighted_median_impl(dont_care);
```

**with\_p\_square\_cumulative\_distribution\_weighted\_median\_impl public member functions**

1. 

```
void operator()(dont_care);
```
2. 

```
template<typename Args> result_type result(Args const & args) const;
```

## Struct template absolute\_tail\_variate\_means

boost::accumulators::tag::absolute\_tail\_variate\_means

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename LeftRight, typename VariateType, typename VariateTag>
struct absolute_tail_variate_means : public boost::accumulators::depends_on< count, non_coherent_tail_mean< LeftRight >, tail_variate< VariateType, VariateTag, LeftRight > >
{
};
```

## Struct template absolute\_weighted\_tail\_variate\_means

boost::accumulators::tag::absolute\_weighted\_tail\_variate\_means

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename LeftRight, typename VariateType, typename VariateTag>
struct absolute_weighted_tail_variate_means : public boost::accumulators::depends_on< non_coherent_weighted_tail_mean< LeftRight >, tail_variate< VariateType, VariateTag, LeftRight >, tail_weights< LeftRight > >
{
};
```

### Struct template coherent\_tail\_mean

boost::accumulators::tag::coherent\_tail\_mean

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename LeftRight>
struct coherent_tail_mean : public boost::accumulators::depends_on< count, quantile, non_coherent_tail_mean< LeftRight > >
{
};
```

### Struct template covariance

boost::accumulators::tag::covariance

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename VariateType, typename VariateTag>
struct covariance : public boost::accumulators::depends_on< count, mean, mean_of_variates< VariateType, VariateTag > >
{
};
```

### Struct template error\_of

boost::accumulators::tag::error\_of

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename Feature>
struct error_of : public boost::accumulators::depends_on< Feature > {
};
```



## Struct template immediate\_mean\_of\_variates

boost::accumulators::tag::immediate\_mean\_of\_variates

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename VariateType, typename VariateTag>
struct immediate_mean_of_variates :
    public boost::accumulators::depends_on< count >
{
};
```

## Struct template immediate\_weighted\_mean\_of\_variates

boost::accumulators::tag::immediate\_weighted\_mean\_of\_variates

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename VariateType, typename VariateTag>
struct immediate_weighted_mean_of_variates :
    public boost::accumulators::depends_on< sum_of_weights >
{
};
```

## Struct template left\_tail\_variate

boost::accumulators::tag::left\_tail\_variate

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename VariateType, typename VariateTag, typename LeftRight>
struct left_tail_variate {
};
```

## Struct template mean\_of\_variates

boost::accumulators::tag::mean\_of\_variates

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename VariateType, typename VariateTag>
struct mean_of_variates : public boost::accumulators::depends_on< count, sum_of_variates< VariateType, VariateTag > >
{
};
```

## Struct template moment

boost::accumulators::tag::moment

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<int N>
struct moment : public boost::accumulators::depends_on< count > {
};
```

## Struct template non\_coherent\_tail\_mean

boost::accumulators::tag::non\_coherent\_tail\_mean

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename LeftRight>
struct non_coherent_tail_mean :
    public boost::accumulators::depends_on< count, tail< LeftRight > >
{
};
```

## Struct template non\_coherent\_weighted\_tail\_mean

boost::accumulators::tag::non\_coherent\_weighted\_tail\_mean

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename LeftRight>
struct non_coherent_weighted_tail_mean : public boost::accumulators::depends_on< sum_of_weights,
tail_weights< LeftRight > >
{
};
```

## Struct template peaks\_over\_threshold

boost::accumulators::tag::peaks\_over\_threshold

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename LeftRight>
struct peaks_over_threshold :
    public boost::accumulators::depends_on< count >, public pot_threshold_value
{
};
```

## Struct template peaks\_over\_threshold\_prob

boost::accumulators::tag::peaks\_over\_threshold\_prob

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename LeftRight>
struct peaks_over_threshold_prob : public boost::accumulators::depends_on< count, tail< LeftRight > >,
                                public pot_threshold_probability
{
};
```

## Struct template pot\_quantile

boost::accumulators::tag::pot\_quantile

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename LeftRight>
struct pot_quantile :
    public boost::accumulators::depends_on< peaks_over_threshold< LeftRight > >
{
};
```

## Struct template pot\_quantile\_prob

boost::accumulators::tag::pot\_quantile\_prob

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename LeftRight>
struct pot_quantile_prob : public boost::accumulators::depends_on< peaks_over_threshold_prob< LeftRight > >
{
};
```

## Struct template pot\_tail\_mean

boost::accumulators::tag::pot\_tail\_mean

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename LeftRight>
struct pot_tail_mean : public boost::accumulators::depends_on< peaks_over_threshold< LeftRight >,
    pot_quantile< LeftRight > >
{
};
```

### Struct template pot\_tail\_mean\_prob

boost::accumulators::tag::pot\_tail\_mean\_prob

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename LeftRight>
struct pot_tail_mean_prob : public boost::accumulators::depends_on< peaks_over_threshold_prob<
    LeftRight >, pot_quantile_prob< LeftRight > >
{
};
```

### Struct quantile

boost::accumulators::tag::quantile

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

struct quantile : public boost::accumulators::depends_on<> {
    // types
    typedef mpl::print< class ____MISSING_SPECIFIC_QUANTILE_FEATURE_IN_ACCUMULATOR_SET____ > impl;
};
```

### Struct template relative\_tail\_variate\_means

boost::accumulators::tag::relative\_tail\_variate\_means

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename LeftRight, typename VariateType, typename VariateTag>
struct relative_tail_variate_means : public boost::accumulators::depends_on< count, non_coherent_tail_mean<
    LeftRight >, tail_variate< VariateType, VariateTag, LeftRight > >
{
};
```

## Struct template `relative_weighted_tail_variate_means`

`boost::accumulators::tag::relative_weighted_tail_variate_means`

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename LeftRight, typename VariateType, typename VariateTag>
struct relative_weighted_tail_variate_means : public boost::accumulators::depends_on< non_coherent_weighted_tail_mean< LeftRight >, tail_variate< VariateType, VariateTag, LeftRight >, tail_weights< LeftRight > >
{
};
```

## Struct template `right_tail_variate`

`boost::accumulators::tag::right_tail_variate`

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename VariateType, typename VariateTag, typename LeftRight>
struct right_tail_variate {
};
```

## Struct template `sum_of_variates`

`boost::accumulators::tag::sum_of_variates`

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename VariateType, typename VariateTag>
struct sum_of_variates : public boost::accumulators::depends_on<> {
};
```

## Struct template `sum_of_variates_kahan`

`boost::accumulators::tag::sum_of_variates_kahan`

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename VariateType, typename VariateTag>
struct sum_of_variates_kahan : public boost::accumulators::depends_on<> {
};
```

## Struct template tail

boost::accumulators::tag::tail

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename LeftRight>
struct tail : public boost::accumulators::depends_on<>,
              public boost::accumulators::tail_cache_size_named_arg< LeftRight >
{
};
```

## Struct tail\_mean

boost::accumulators::tag::tail\_mean

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

struct tail_mean : public boost::accumulators::depends_on<> {
    // types
    typedef mpl::print< class ____MISSING_SPECIFIC_TAIL_MEAN_FEATURE_IN_ACCUMULATOR_SET____ > impl;
};
```

## Struct template tail\_quantile

boost::accumulators::tag::tail\_quantile

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename LeftRight>
struct tail_quantile :
    public boost::accumulators::depends_on< count, tail< LeftRight > >
{
};
```

## Struct template tail\_variate

boost::accumulators::tag::tail\_variate

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename VariateType, typename VariateTag, typename LeftRight>
struct tail_variate :
    public boost::accumulators::depends_on< tail< LeftRight > >
{
};
```

### Struct template tail\_variate\_means

boost::accumulators::tag::tail\_variate\_means

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename LeftRight, typename VariateType, typename VariateTag>
struct tail_variate_means {
};
```

### Struct template tail\_weights

boost::accumulators::tag::tail\_weights

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename LeftRight>
struct tail_weights :
    public boost::accumulators::depends_on< tail< LeftRight > >
{
};
```

### Struct template weighted\_covariance

boost::accumulators::tag::weighted\_covariance

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename VariateType, typename VariateTag>
struct weighted_covariance : public boost::accumulators::depends_on< count, sum_of_weights,
weighted_mean, weighted_mean_of_variates< VariateType, VariateTag > >
{
};
```

### Struct template weighted\_mean\_of\_variates

boost::accumulators::tag::weighted\_mean\_of\_variates

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename VariateType, typename VariateTag>
struct weighted_mean_of_variates : public boost::accumulators::depends_on< sum_of_weights, ↵
weighted_sum_of_variates< VariateType, VariateTag > >
{
};
```

### Struct template weighted\_moment

boost::accumulators::tag::weighted\_moment

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<int N>
struct weighted_moment :
    public boost::accumulators::depends_on< count, sum_of_weights >
{
};
```

### Struct template weighted\_peaks\_over\_threshold

boost::accumulators::tag::weighted\_peaks\_over\_threshold

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename LeftRight>
struct weighted_peaks_over_threshold : public boost::accumulators::depends_on< sum_of_weights >,
    public pot_threshold_value
{
};
```

### Struct template weighted\_peaks\_over\_threshold\_prob

boost::accumulators::tag::weighted\_peaks\_over\_threshold\_prob

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename LeftRight>
struct weighted_peaks_over_threshold_prob : public boost::accumulators::depends_on< ↵
sum_of_weights, tail_weights< LeftRight > >,
    public pot_threshold_probability
{
};
```



## Struct template `weighted_pot_quantile`

`boost::accumulators::tag::weighted_pot_quantile`

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename LeftRight>
struct weighted_pot_quantile : public boost::accumulators::depends_on<
weighted_peaks_over_threshold< LeftRight > >
{
};
```

## Struct template `weighted_pot_quantile_prob`

`boost::accumulators::tag::weighted_pot_quantile_prob`

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename LeftRight>
struct weighted_pot_quantile_prob : public boost::accumulators::depends_on<
weighted_peaks_over_threshold_prob< LeftRight > >
{
};
```

## Struct template `weighted_pot_tail_mean`

`boost::accumulators::tag::weighted_pot_tail_mean`

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename LeftRight>
struct weighted_pot_tail_mean : public boost::accumulators::depends_on<
weighted_peaks_over_threshold< LeftRight >, weighted_pot_quantile< LeftRight > >
{
};
```

## Struct template `weighted_pot_tail_mean_prob`

`boost::accumulators::tag::weighted_pot_tail_mean_prob`

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename LeftRight>
struct weighted_pot_tail_mean_prob : public boost::accumulators::depends_on<
weighted_peaks_over_threshold_prob< LeftRight >, weighted_pot_quantile_prob< LeftRight > >
{
};
```

### Struct template weighted\_sum\_of\_variates

boost::accumulators::tag::weighted\_sum\_of\_variates

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename VariateType, typename VariateTag>
struct weighted_sum_of_variates : public boost::accumulators::depends_on<> {
};
```

### Struct template weighted\_tail\_quantile

boost::accumulators::tag::weighted\_tail\_quantile

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename LeftRight>
struct weighted_tail_quantile : public boost::accumulators::depends_on< sum_of_weights,
tail_weights< LeftRight > >
{
};
```

### Struct template weighted\_tail\_variate\_means

boost::accumulators::tag::weighted\_tail\_variate\_means

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<typename LeftRight, typename VariateType, typename VariateTag>
struct weighted_tail_variate_means {
};
```

### Struct absolute

boost::accumulators::absolute

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

struct absolute {
};
```

### Struct for\_median

boost::accumulators::for\_median

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

struct for_median {
};
```

### Struct immediate

boost::accumulators::immediate

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

struct immediate {
};
```

### Struct kahan

boost::accumulators::kahan

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

struct kahan {
};
```

### Struct lazy

boost::accumulators::lazy

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

struct lazy {
};
```

### Struct left

boost::accumulators::left

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

struct left {
};
```

### Struct linear

boost::accumulators::linear

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

struct linear {
};
```

### Struct quadratic

boost::accumulators::quadratic

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

struct quadratic {
};
```

### Struct regular

boost::accumulators::regular

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

struct regular {
};
```

### Struct relative

boost::accumulators::relative

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

struct relative {
};
```

### Struct right

boost::accumulators::right

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

struct right {
};
```

### Struct template stats

boost::accumulators::stats — An MPL sequence of statistics.

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<BOOST_PP_ENUM_PARAMS_WITH_A_DEFAULT(BOOST_ACCUMULATORS_MAX_FEATURES, typename Feature, mpl::na) >
struct stats : public mpl::vector< Stat1, Stat2, ... > {
};
```

### Struct unweighted

boost::accumulators::unweighted

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

struct unweighted {
};
```

### Struct weighted

boost::accumulators::weighted

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

struct weighted {
};
```

### Struct with\_density

boost::accumulators::with\_density

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

struct with_density {
};
```

### Struct template with\_error

boost::accumulators::with\_error

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

template<BOOST_PP_ENUM_PARAMS_WITH_A_DEFAULT(BOOST_ACCUMULATORS_MAX_FEATURES, typename Feature, mpl::na) >
struct with_error {
};
```

### Struct with\_p\_square\_cumulative\_distribution

boost::accumulators::with\_p\_square\_cumulative\_distribution

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

struct with_p_square_cumulative_distribution {
};
```

### Struct with\_p\_square\_quantile

boost::accumulators::with\_p\_square\_quantile

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

struct with_p_square_quantile {
};
```

### Struct with\_threshold\_probability

boost::accumulators::with\_threshold\_probability

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

struct with_threshold_probability {
};
```

### Struct with\_threshold\_value

boost::accumulators::with\_threshold\_value

## Synopsis

```
// In header: <boost/accumulators/statistics_fwd.hpp>

struct with_threshold_value {
};
```

# Numeric Operators Library Reference

## Header <boost/accumulators/numeric/functional.hpp>

```

namespace boost {
    namespace numeric {
        template<typename T> struct default_;
        template<typename T> struct one;
        template<typename T> struct one_or_default;
        template<typename T> struct zero;
        template<typename T> struct zero_or_default;
        template<typename To, typename From>
            lazy_disable_if< is_const< From >, mpl::if_< is_same< To, From >, To &, To > >::type
            promote(From & from);
        template<typename To, typename From>
            mpl::if_< is_same< To const, From const >, To const &, To const >::type
            promote(From const & from);
    }
    namespace functional {
        template<typename Arg, typename Tag> struct as_max;
        template<typename Arg, typename EnableIf> struct as_max_base;
        template<typename Arg, typename Tag> struct as_min;
        template<typename Arg, typename EnableIf> struct as_min_base;

        template<typename Arg>
            struct as_min_base<Arg, typename enable_if< is_floating_point< Arg > >::type>;

        template<typename Arg, typename Tag> struct as_one;
        template<typename Arg, typename EnableIf> struct as_one_base;
        template<typename Arg, typename Tag> struct as_zero;
        template<typename Arg, typename EnableIf> struct as_zero_base;
        template<typename Left, typename Right,
                typename LeftTag = typename tag<Left>::type,
                typename RightTag = typename tag<Right>::type>
            struct assign;
        template<typename Left, typename Right, typename EnableIf = void>
            struct assign_base;
        template<typename Arg, typename Tag = typename tag<Arg>::type>
            struct complement;
        template<typename Arg, typename EnableIf = void> struct complement_base;
        template<typename Left, typename Right,
                typename LeftTag = typename tag<Left>::type,
                typename RightTag = typename tag<Right>::type>
            struct divides;
        template<typename Left, typename Right,
                typename LeftTag = typename tag<Left>::type,
                typename RightTag = typename tag<Right>::type>
            struct divides_assign;
        template<typename Left, typename Right, typename EnableIf = void>
            struct divides_assign_base;
        template<typename Left, typename Right, typename EnableIf = void>
            struct divides_base;
        template<typename Left, typename Right,
                typename LeftTag = typename tag<Left>::type,
                typename RightTag = typename tag<Right>::type>
            struct equal_to;
        template<typename Left, typename Right, typename EnableIf = void>
            struct equal_to_base;
        template<typename Left, typename Right, typename LeftTag,
                typename RightTag>
            struct fdiv;
        template<typename Left, typename Right, typename EnableIf> struct fdiv_base;
    }
}

```



```

template<typename Left, typename Right>
    struct fdiv_base<Left, Right, typename enable_if< are_integral< Left, Right > >::type>;

template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
    struct greater;
template<typename Left, typename Right, typename EnableIf = void>
    struct greater_base;
template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
    struct greater_equal;
template<typename Left, typename Right, typename EnableIf = void>
    struct greater_equal_base;
template<typename Left, typename Right> struct left_ref;
template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
    struct less;
template<typename Left, typename Right, typename EnableIf = void>
    struct less_base;
template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
    struct less_equal;
template<typename Left, typename Right, typename EnableIf = void>
    struct less_equal_base;
template<typename Arg, typename Tag = typename tag<Arg>::type>
    struct logical_not;
template<typename Arg, typename EnableIf = void> struct logical_not_base;
template<typename Left, typename Right, typename LeftTag,
        typename RightTag>
    struct max_assign;
template<typename Left, typename Right, typename EnableIf>
    struct max_assign_base;
template<typename Left, typename Right, typename LeftTag,
        typename RightTag>
    struct min_assign;
template<typename Left, typename Right, typename EnableIf>
    struct min_assign_base;
template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
    struct minus;
template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
    struct minus_assign;
template<typename Left, typename Right, typename EnableIf = void>
    struct minus_assign_base;
template<typename Left, typename Right, typename EnableIf = void>
    struct minus_base;
template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
    struct modulus;
template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
    struct modulus_assign;
template<typename Left, typename Right, typename EnableIf = void>
    struct modulus_assign_base;

```

```

template<typename Left, typename Right, typename EnableIf = void>
    struct modulus_base;
template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
    struct multiplies;
template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
    struct multiplies_assign;
template<typename Left, typename Right, typename EnableIf = void>
    struct multiplies_assign_base;
template<typename Left, typename Right, typename EnableIf = void>
    struct multiplies_base;
template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
    struct not_equal_to;
template<typename Left, typename Right, typename EnableIf = void>
    struct not_equal_to_base;
template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
    struct plus;
template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
    struct plus_assign;
template<typename Left, typename Right, typename EnableIf = void>
    struct plus_assign_base;
template<typename Left, typename Right, typename EnableIf = void>
    struct plus_base;
template<typename To, typename From, typename ToTag, typename FromTag>
    struct promote;
template<typename To, typename From, typename EnableIf> struct promote_base;

template<typename ToFrom> struct promote_base<ToFrom, ToFrom, void>;

template<typename Arg, typename Tag = typename tag<Arg>::type>
    struct unary_minus;
template<typename Arg, typename EnableIf = void> struct unary_minus_base;
template<typename Arg, typename Tag = typename tag<Arg>::type>
    struct unary_plus;
template<typename Arg, typename EnableIf = void> struct unary_plus_base;
}
namespace op {
    struct as_max;
    struct as_min;
    struct as_one;
    struct as_zero;
    struct assign;
    struct complement;
    struct divides;
    struct divides_assign;
    struct equal_to;
    struct fdiv;
    struct greater;
    struct greater_equal;
    struct less;
    struct less_equal;
    struct logical_not;
    struct max_assign;
    struct min_assign;

```

```
    struct minus;
    struct minus_assign;
    struct modulus;
    struct modulus_assign;
    struct multiplies;
    struct multiplies_assign;
    struct not_equal_to;
    struct plus;
    struct plus_assign;
    template<typename To> struct promote;
    struct unary_minus;
    struct unary_plus;
}
}
```

## Struct template `as_max`

`boost::numeric::functional::as_max`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Arg, typename Tag>
struct as_max : public boost::numeric::functional::as_max_base< Arg, void > {
};
```

## Struct template `as_max_base`

`boost::numeric::functional::as_max_base`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Arg, typename EnableIf>
struct as_max_base :
    public std::unary_function< Arg, remove_const< Arg >::type >
{
    // public member functions
    BOOST_STATIC_ASSERT(std::numeric_limits< typename remove_const< Arg >::type >::is_specialized);
    remove_const< Arg >::type operator()(Arg &) const;
};
```

## Description

### `as_max_base` public member functions

1. `BOOST_STATIC_ASSERT(std::numeric_limits< typename remove_const< Arg >::type >::is_specialized);`
2. `remove_const< Arg >::type operator()(Arg &) const;`

## Struct template `as_min`

`boost::numeric::functional::as_min`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Arg, typename Tag>
struct as_min : public boost::numeric::functional::as_min_base< Arg, void > {
};
```

## Struct template `as_min_base`

`boost::numeric::functional::as_min_base`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Arg, typename EnableIf>
struct as_min_base :
    public std::unary_function< Arg, remove_const< Arg >::type >
{
    // public member functions
    BOOST_STATIC_ASSERT(std::numeric_limits< typename remove_const< Arg >::type >::is_specialized);
    remove_const< Arg >::type operator()(Arg &) const;
};
```

## Description

### `as_min_base` public member functions

1. `BOOST_STATIC_ASSERT(std::numeric_limits< typename remove_const< Arg >::type >::is_specialized);`
2. `remove_const< Arg >::type operator()(Arg &) const;`

## Struct template `as_min_base<Arg, typename enable_if< is_floating_point< Arg > >::type>`

`boost::numeric::functional::as_min_base<Arg, typename enable_if< is_floating_point< Arg > >::type>`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Arg>
struct as_min_base<Arg, typename enable_if< is_floating_point< Arg >>::type> :
    public std::unary_function< Arg, remove_const< Arg >::type> >
{
    // public member functions
    BOOST_STATIC_ASSERT(std::numeric_limits< typename remove_const< Arg >::type>::is_specialized);
    remove_const< Arg >::type operator()(Arg &) const;
};
```

### Description

#### as\_min\_base public member functions

1. 

```
BOOST_STATIC_ASSERT(std::numeric_limits< typename remove_const< Arg >::type>::is_specialized);
```
2. 

```
remove_const< Arg >::type operator()(Arg &) const;
```

### Struct template as\_one

boost::numeric::functional::as\_one

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Arg, typename Tag>
struct as_one : public boost::numeric::functional::as_one_base< Arg, void> > {
};
```

### Struct template as\_one\_base

boost::numeric::functional::as\_one\_base

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Arg, typename EnableIf>
struct as_one_base :
    public std::unary_function< Arg, remove_const< Arg >::type> >
{
    // public member functions
    remove_const< Arg >::type operator()(Arg &) const;
};
```

## Description

**as\_one\_base** public member functions

1. 

```
remove_const< Arg >::type operator()(Arg &) const;
```

## Struct template as\_zero

boost::numeric::functional::as\_zero

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Arg, typename Tag>
struct as_zero : public boost::numeric::functional::as_zero_base< Arg, void > {
};
```

## Struct template as\_zero\_base

boost::numeric::functional::as\_zero\_base

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Arg, typename EnableIf>
struct as_zero_base :
    public std::unary_function< Arg, remove_const< Arg >::type >
{
    // public member functions
    remove_const< Arg >::type operator()(Arg &) const;
};
```

## Description

**as\_zero\_base** public member functions

1. 

```
remove_const< Arg >::type operator()(Arg &) const;
```

## Struct template assign

boost::numeric::functional::assign

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
struct assign :
    public boost::numeric::functional::assign_base< Left, Right, void >
{
};
```

### Struct template assign\_base

boost::numeric::functional::assign\_base

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right, typename EnableIf = void>
struct assign_base : public std::binary_function< Left, Right,
    typename std::result_of<Left >()=lvalue< Left >()=lvalue<
    Right >()>
{
    // public member functions
    result_type operator()(Left &, Right &) const;
};
```

### Description

#### assign\_base public member functions

1. `result_type operator()(Left & left, Right & right) const;`

Returns:      left = right

### Struct template complement

boost::numeric::functional::complement

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Arg, typename Tag = typename tag<Arg>::type>
struct complement :
    public boost::numeric::functional::complement_base< Arg, void >
{
};
```

### Struct template complement\_base

boost::numeric::functional::complement\_base

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Arg, typename EnableIf = void>
struct complement_base :
    public std::unary_function< Arg, typeof(~lvalue< Arg >())>
{
    // public member functions
    result_type operator()(Arg &) const;
};
```

### Description

**complement\_base public member functions**

1. `result_type operator()(Arg & arg) const;`

Returns:      ~ arg

### Struct template divides

boost::numeric::functional::divides

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right,
         typename LeftTag = typename tag<Left>::type,
         typename RightTag = typename tag<Right>::type>
struct divides :
    public boost::numeric::functional::divides_base< Left, Right, void >
{
};
```

### Struct template divides\_assign

boost::numeric::functional::divides\_assign

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right,
         typename LeftTag = typename tag<Left>::type,
         typename RightTag = typename tag<Right>::type>
struct divides_assign :
    public boost::numeric::functional::divides_assign_base< Left, Right, void >
{
};
```



## Struct template divides\_assign\_base

boost::numeric::functional::divides\_assign\_base

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right, typename EnableIf = void>
struct divides_assign_base : public std::binary_function< Left, Right,
    typename std::enable_if< !std::is_lvalue< Left >() ||
    !std::is_lvalue< Right >() >::type> >()>
{
    // public member functions
    result_type operator()(Left &, Right &) const;
};
```

### Description

**divides\_assign\_base** public member functions

1. `result_type operator()(Left & left, Right & right) const;`

Returns:      left /= right

## Struct template divides\_base

boost::numeric::functional::divides\_base

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right, typename EnableIf = void>
struct divides_base : public std::binary_function< Left, Right,
    typename std::enable_if< !std::is_lvalue< Left >() ||
    !std::is_lvalue< Right >() >::type> >()>
{
    // public member functions
    result_type operator()(Left &, Right &) const;
};
```

### Description

**divides\_base** public member functions

1. `result_type operator()(Left & left, Right & right) const;`

Returns:      left / right

## Struct template equal\_to

boost::numeric::functional::equal\_to

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
struct equal_to :
    public boost::numeric::functional::equal_to_base< Left, Right, void >
{
};
```

### Struct template equal\_to\_base

boost::numeric::functional::equal\_to\_base

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right, typename EnableIf = void>
struct equal_to_base : public std::binary_function< Left, Right,
    typename std::enable_if< !std::is_lvalue_reference<Left> &&
    !std::is_lvalue_reference<Right> &&
    !std::is_const<Left> && !std::is_const<Right> &&
    !std::is_volatile<Left> && !std::is_volatile<Right> &&
    void*>::type>
{
    // public member functions
    result_type operator()(Left &, Right &) const;
};
```

### Description

#### equal\_to\_base public member functions

1. `result_type operator()(Left & left, Right & right) const;`

Returns:      left == right

### Struct template fdiv

boost::numeric::functional::fdiv

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right, typename LeftTag, typename RightTag>
struct fdiv :
    public boost::numeric::functional::fdiv_base< Left, Right, void >
{
};
```

### Struct template fdiv\_base

boost::numeric::functional::fdiv\_base

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right, typename EnableIf>
struct fdiv_base : public boost::numeric::functional::divides< Left, Right > {
};
```

### Struct template fdiv\_base<Left, Right, typename enable\_if< are\_integral< Left, Right > >::type>

boost::numeric::functional::fdiv\_base<Left, Right, typename enable\_if< are\_integral< Left, Right > >::type>

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right>
struct fdiv_base<Left, Right, typename enable_if< are_integral< Left, Right > >::type> :
    public boost::numeric::functional::divides< double const, double const >
{
};
```

### Struct template greater

boost::numeric::functional::greater

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
struct greater :
    public boost::numeric::functional::greater_base< Left, Right, void >
{
};
```

### Struct template greater\_base

boost::numeric::functional::greater\_base

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right, typename EnableIf = void>
struct greater_base : public std::binary_function< Left, Right, typeof(lvalue< Left >() > lvalue<
Right >())>
{

    // public member functions
    result_type operator()(Left &, Right &) const;
};
```

## Description

### **greater\_base** public member functions

1. 

```
result_type operator()(Left & left, Right & right) const;
```

Returns:     left > right

## Struct template **greater\_equal**

boost::numeric::functional::greater\_equal

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right,
         typename LeftTag = typename tag<Left>::type,
         typename RightTag = typename tag<Right>::type>
struct greater_equal :
    public boost::numeric::functional::greater_equal_base< Left, Right, void >
{
};
```

## Struct template **greater\_equal\_base**

boost::numeric::functional::greater\_equal\_base

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right, typename EnableIf = void>
struct greater_equal_base : public std::binary_function< Left, Right,
    typeof(lvalue< Left >() < lvalue< Right >())>
{
    // public member functions
    result_type operator()(Left &, Right &) const;
};
```

## Description

### **greater\_equal\_base** public member functions

1. 

```
result_type operator()(Left & left, Right & right) const;
```

Returns:     left >= right

## Struct template **left\_ref**

boost::numeric::functional::left\_ref



## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
struct less_equal :
    public boost::numeric::functional::less_equal_base< Left, Right, void >
{
};
```

### Struct template less\_equal\_base

boost::numeric::functional::less\_equal\_base

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right, typename EnableIf = void>
struct less_equal_base : public std::binary_function< Left, Right,
    typename std::enable_if< !std::is_lvalue_reference< Left > &
    !std::is_lvalue_reference< Right >, void>::type>
{
    // public member functions
    result_type operator()(Left &, Right &) const;
};
```

### Description

#### less\_equal\_base public member functions

1. `result_type operator()(Left & left, Right & right) const;`

Returns:      left <= right

### Struct template logical\_not

boost::numeric::functional::logical\_not

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Arg, typename Tag = typename tag<Arg>::type>
struct logical_not :
    public boost::numeric::functional::logical_not_base< Arg, void >
{
};
```

### Struct template logical\_not\_base

boost::numeric::functional::logical\_not\_base

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Arg, typename EnableIf = void>
struct logical_not_base :
    public std::unary_function< Arg, typename std::enable_if< !is_lvalue< Arg >(), void >>
{
    // public member functions
    result_type operator()(Arg &) const;
};
```

### Description

#### `logical_not_base` public member functions

1. `result_type operator()(Arg & arg) const;`

Returns: `!arg`

### Struct template `max_assign`

`boost::numeric::functional::max_assign`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right, typename LeftTag, typename RightTag>
struct max_assign :
    public boost::numeric::functional::max_assign_base< Left, Right, void >
{
};
```

### Struct template `max_assign_base`

`boost::numeric::functional::max_assign_base`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right, typename EnableIf>
struct max_assign_base : public std::binary_function< Left, Right, void > {

    // public member functions
    void operator()(Left &, Right &) const;
};
```

## Description

**max\_assign\_base** public member functions

1. 

```
void operator()(Left & left, Right & right) const;
```

## Struct template min\_assign

boost::numeric::functional::min\_assign

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right, typename LeftTag, typename RightTag>
struct min_assign :
    public boost::numeric::functional::min_assign_base< Left, Right, void >
{
};
```

## Struct template min\_assign\_base

boost::numeric::functional::min\_assign\_base

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right, typename EnableIf>
struct min_assign_base : public std::binary_function< Left, Right, void > {

    // public member functions
    void operator()(Left &, Right &) const;
};
```

## Description

**min\_assign\_base** public member functions

1. 

```
void operator()(Left & left, Right & right) const;
```

## Struct template minus

boost::numeric::functional::minus



## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
struct minus :
    public boost::numeric::functional::minus_base< Left, Right, void >
{
};
```

### Struct template minus\_assign

boost::numeric::functional::minus\_assign

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
struct minus_assign :
    public boost::numeric::functional::minus_assign_base< Left, Right, void >
{
};
```

### Struct template minus\_assign\_base

boost::numeric::functional::minus\_assign\_base

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right, typename EnableIf = void>
struct minus_assign_base : public std::binary_function< Left, Right,
    typename std::result_of<Left &()>::type>
{
    // public member functions
    result_type operator()(Left &, Right &) const;
};
```

### Description

#### minus\_assign\_base public member functions

1. `result_type operator()(Left & left, Right & right) const;`

Returns:     left -= right

## Struct template minus\_base

boost::numeric::functional::minus\_base

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right, typename EnableIf = void>
struct minus_base : public std::binary_function< Left, Right, typeof(lvalue< Left >()-lvalue< Left >())>
{
    // public member functions
    result_type operator()(Left &, Right &) const;
};
```

### Description

#### minus\_base public member functions

1. 

```
result_type operator()(Left & left, Right & right) const;
```

Returns:      left - right

## Struct template modulus

boost::numeric::functional::modulus

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
struct modulus :
    public boost::numeric::functional::modulus_base< Left, Right, void >
{
};
```

## Struct template modulus\_assign

boost::numeric::functional::modulus\_assign

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
struct modulus_assign :
    public boost::numeric::functional::modulus_assign_base< Left, Right, void >
{
};
```

### Struct template modulus\_assign\_base

boost::numeric::functional::modulus\_assign\_base

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right, typename EnableIf = void>
struct modulus_assign_base : public std::binary_function< Left, Right,
    typename std::result_of<LeftTag::operator%(RightTag::type const&,
    LeftTag::type const&)>::type>
{
    // public member functions
    result_type operator()(Left &, Right &) const;
};
```

### Description

#### modulus\_assign\_base public member functions

1. `result_type operator()(Left & left, Right & right) const;`

Returns:      left %= right

### Struct template modulus\_base

boost::numeric::functional::modulus\_base

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right, typename EnableIf = void>
struct modulus_base : public std::binary_function< Left, Right,
    typename std::result_of<LeftTag::operator%(RightTag::type const&,
    LeftTag::type const&)>::type>
{
    // public member functions
    result_type operator()(Left &, Right &) const;
};
```

## Description

**modulus\_base** public member functions

```
1. result_type operator()(Left & left, Right & right) const;
```

Returns:      left % right

## Struct template multiplies

boost::numeric::functional::multiplies

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
struct multiplies :
    public boost::numeric::functional::multiplies_base< Left, Right, void >
{
};
```

## Struct template multiplies\_assign

boost::numeric::functional::multiplies\_assign

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
struct multiplies_assign : public boost::numeric::functional::multiplies_assign_base< Left, ↵
    Right, void >
{
};
```

## Struct template multiplies\_assign\_base

boost::numeric::functional::multiplies\_assign\_base

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right, typename EnableIf = void>
struct multiplies_assign_base : public std::binary_function< Left, Right, typeof(lvalue< Left >() * lvalue< Right >())>
{
    // public member functions
    result_type operator()(Left &, Right &) const;
};
```

### Description

#### **multiplies\_assign\_base** public member functions

1. `result_type operator()(Left & left, Right & right) const;`

Returns:      left \*= right

### Struct template **multiplies\_base**

boost::numeric::functional::multiplies\_base

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right, typename EnableIf = void>
struct multiplies_base : public std::binary_function< Left, Right, typeof(lvalue< Left >() * lvalue< Right >())>
{
    // public member functions
    result_type operator()(Left &, Right &) const;
};
```

### Description

#### **multiplies\_base** public member functions

1. `result_type operator()(Left & left, Right & right) const;`

Returns:      left \* right

### Struct template **not\_equal\_to**

boost::numeric::functional::not\_equal\_to

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
struct not_equal_to :
    public boost::numeric::functional::not_equal_to_base< Left, Right, void >
{
};
```

### Struct template not\_equal\_to\_base

boost::numeric::functional::not\_equal\_to\_base

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right, typename EnableIf = void>
struct not_equal_to_base : public std::binary_function< Left, Right,
    typename std::enable_if< !std::is_lvalue< Left >() ||
    !std::is_lvalue< Right >() >::type>
{
    // public member functions
    result_type operator()(Left &, Right &) const;
};
```

### Description

#### not\_equal\_to\_base public member functions

1. `result_type operator()(Left & left, Right & right) const;`

Returns:      left != right

### Struct template plus

boost::numeric::functional::plus

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
struct plus :
    public boost::numeric::functional::plus_base< Left, Right, void >
{
};
```

## Struct template `plus_assign`

`boost::numeric::functional::plus_assign`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right,
        typename LeftTag = typename tag<Left>::type,
        typename RightTag = typename tag<Right>::type>
struct plus_assign :
    public boost::numeric::functional::plus_assign_base< Left, Right, void >
{
};
```

## Struct template `plus_assign_base`

`boost::numeric::functional::plus_assign_base`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right, typename EnableIf = void>
struct plus_assign_base : public std::binary_function< Left, Right,
    typeof(lvalue< Left >() += lvalue< Right >())>
{
    // public member functions
    result_type operator()(Left &, Right &) const;
};
```

## Description

**`plus_assign_base` public member functions**

1. `result_type operator()(Left & left, Right & right) const;`

Returns:     `left += right`

## Struct template `plus_base`

`boost::numeric::functional::plus_base`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right, typename EnableIf = void>
struct plus_base : public std::binary_function< Left, Right, typeof(lvalue< Left >()+lvalue<
Right >())>
{
    // public member functions
    result_type operator()(Left &, Right &) const;
};
```

### Description

#### plus\_base public member functions

1. `result_type operator()(Left & left, Right & right) const;`

Returns:      left + right

### Struct template promote

boost::numeric::functional::promote

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename To, typename From, typename ToTag, typename FromTag>
struct promote :
    public boost::numeric::functional::promote_base< To, From, void >
{
};
```

### Struct template promote\_base

boost::numeric::functional::promote\_base

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename To, typename From, typename EnableIf>
struct promote_base : public std::unary_function< From, To > {

    // public member functions
    To operator()(From &) const;
};
```



## Description

**promote\_base** public member functions

1. 

```
To operator()(From & from) const;
```

## Struct template **promote\_base<ToFrom, ToFrom, void>**

boost::numeric::functional::promote\_base<ToFrom, ToFrom, void>

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename ToFrom>
struct promote_base<ToFrom, ToFrom, void> : public std::unary_function< ToFrom, ToFrom > {

    // public member functions
    ToFrom & operator()(ToFrom &);
};
```

## Description

**promote\_base** public member functions

1. 

```
ToFrom & operator()(ToFrom & tofrom);
```

## Struct template **unary\_minus**

boost::numeric::functional::unary\_minus

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Arg, typename Tag = typename tag<Arg>::type>
struct unary_minus :
    public boost::numeric::functional::unary_minus_base< Arg, void >
{
};
```

## Struct template **unary\_minus\_base**

boost::numeric::functional::unary\_minus\_base

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Arg, typename EnableIf = void>
struct unary_minus_base :
    public std::unary_function< Arg, typeof(-lvalue< Arg >())>
{
    // public member functions
    result_type operator()(Arg &) const;
};
```

### Description

#### **unary\_minus\_base** public member functions

1. `result_type operator()(Arg & arg) const;`

Returns:      - arg

### Struct template unary\_plus

boost::numeric::functional::unary\_plus

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Arg, typename Tag = typename tag<Arg>::type>
struct unary_plus :
    public boost::numeric::functional::unary_plus_base< Arg, void >
{
};
```

### Struct template unary\_plus\_base

boost::numeric::functional::unary\_plus\_base

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Arg, typename EnableIf = void>
struct unary_plus_base :
    public std::unary_function< Arg, typeof(+lvalue< Arg >())>
{
    // public member functions
    result_type operator()(Arg &) const;
};
```

## Description

**unary\_plus\_base** public member functions

```
1. result_type operator()(Arg & arg) const;
```

Returns:      + arg

## Struct as\_max

boost::numeric::op::as\_max

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct as_max {
};
```

## Struct as\_min

boost::numeric::op::as\_min

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct as_min {
};
```

## Struct as\_one

boost::numeric::op::as\_one

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct as_one {
};
```

## Struct as\_zero

boost::numeric::op::as\_zero

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct as_zero {
};
```

### Struct assign

boost::numeric::op::assign

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct assign {
};
```

### Struct complement

boost::numeric::op::complement

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct complement {
};
```

### Struct divides

boost::numeric::op::divides

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct divides {
};
```

### Struct divides\_assign

boost::numeric::op::divides\_assign

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct divides_assign {
};
```

### Struct equal\_to

boost::numeric::op::equal\_to

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct equal_to {
};
```

### Struct fdiv

boost::numeric::op::fdiv

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct fdiv {
};
```

### Struct greater

boost::numeric::op::greater

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct greater {
};
```

### Struct greater\_equal

boost::numeric::op::greater\_equal

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct greater_equal {
};
```

### Struct less

boost::numeric::op::less

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct less {
};
```

### Struct less\_equal

boost::numeric::op::less\_equal

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct less_equal {
};
```

### Struct logical\_not

boost::numeric::op::logical\_not

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct logical_not {
};
```

### Struct max\_assign

boost::numeric::op::max\_assign

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct max_assign {
};
```

### Struct min\_assign

boost::numeric::op::min\_assign

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct min_assign {
};
```

### Struct minus

boost::numeric::op::minus

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct minus {
};
```

### Struct minus\_assign

boost::numeric::op::minus\_assign

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct minus_assign {
};
```

### Struct modulus

boost::numeric::op::modulus

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct modulus {
};
```

### Struct modulus\_assign

boost::numeric::op::modulus\_assign

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct modulus_assign {
};
```

### Struct multiplies

boost::numeric::op::multiplies

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct multiplies {
};
```

### Struct multiplies\_assign

boost::numeric::op::multiplies\_assign

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct multiplies_assign {
};
```

### Struct not\_equal\_to

boost::numeric::op::not\_equal\_to



## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct not_equal_to {
};
```

### Struct plus

boost::numeric::op::plus

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct plus {
};
```

### Struct plus\_assign

boost::numeric::op::plus\_assign

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct plus_assign {
};
```

### Struct template promote

boost::numeric::op::promote

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename To>
struct promote {
};
```

### Struct unary\_minus

boost::numeric::op::unary\_minus

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct unary_minus {
};
```

### Struct unary\_plus

boost::numeric::op::unary\_plus

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

struct unary_plus {
};
```

### Struct template default\_

boost::numeric::default\_

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename T>
struct default_ {
    // types
    typedef default_ type;
    typedef T          value_type;

    // public member functions
    operator T const &() const;

    // public data members
    static T const value;
};
```

### Description

**default\_ public member functions**

1. 

```
operator T const &() const;
```

### Struct template one

boost::numeric::one

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename T>
struct one {
    // types
    typedef one type;
    typedef T value_type;

    // public member functions
    operator T const &() const;

    // public data members
    static T const value;
};
```

### Description

one public member functions

1. 

```
operator T const &() const;
```

### Struct template one\_or\_default

boost::numeric::one\_or\_default

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename T>
struct one_or_default :
    public mpl::if_::type< is_empty< T >, default_< T >, one< T > >
{
};
```

### Struct template zero

boost::numeric::zero

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename T>
struct zero {
    // types
    typedef zero type;
    typedef T value_type;

    // public member functions
    operator T const &() const;

    // public data members
    static T const value;
};
```

### Description

**zero** public member functions

1. 

```
operator T const &() const;
```

## Struct template zero\_or\_default

boost::numeric::zero\_or\_default

## Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename T>
struct zero_or_default :
    public mpl::if_::type< is_empty< T >, default_< T >, zero< T > >
{
};
```

## Header <boost/accumulators/numeric/functional/complex.hpp>

```
namespace boost {
    namespace numeric {
        namespace operators {
            template<typename T, typename U>
                disable_if< mpl::or_< is_same< T, U >, is_same< std::complex< T >, U > >, std::complex< T > >::type
                operator*(std::complex< T > ri, U const & u);
            template<typename T, typename U>
                disable_if< mpl::or_< is_same< T, U >, is_same< std::complex< T >, U > >, std::complex< T > >::type
                operator/(std::complex< T > ri, U const & u);
        }
    }
}
```

## Header `<boost/accumulators/numeric/functional/valarray.hpp>`

```
namespace boost {
  namespace numeric {
    namespace functional {
      template<typename T> struct as_max<T, std_valarray_tag>;
      template<typename T> struct as_min<T, std_valarray_tag>;
      template<typename T> struct as_one<T, std_valarray_tag>;
      template<typename T> struct as_zero<T, std_valarray_tag>;
      template<typename Left, typename Right, typename RightTag>
        struct fdiv<Left, Right, std_valarray_tag, RightTag>;
      template<typename Left, typename Right>
        struct max_assign<Left, Right, std_valarray_tag, std_valarray_tag>;
      template<typename Left, typename Right>
        struct min_assign<Left, Right, std_valarray_tag, std_valarray_tag>;
      template<typename From>
        struct promote<bool const, From, void, std_valarray_tag>;
      template<typename From> struct promote<bool, From, void, std_valarray_tag>;
      template<typename To, typename From>
        struct promote<To, From, std_valarray_tag, std_valarray_tag>;
      template<typename ToFrom>
        struct promote<ToFrom, ToFrom, std_valarray_tag, std_valarray_tag>;
      template<typename T> struct tag<std::valarray< T >>;
    }
    namespace operators {
      template<typename Left, typename Right>
        unspecified operator/(std::valarray< Left > const & left,
                             Right const & right);
      template<typename Left, typename Right>
        unspecified operator*(std::valarray< Left > const & left,
                              Right const & right);
      template<typename Left, typename Right>
        unspecified operator+(std::valarray< Left > const & left,
                              std::valarray< Right > const & right);
    }
  }
}
```

### Struct template `as_max<T, std_valarray_tag>`

`boost::numeric::functional::as_max<T, std_valarray_tag>`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional/valarray.hpp>

template<typename T>
struct as_max<T, std_valarray_tag> : public std::unary_function< T, remove_const< T >::type > {

  // public member functions
  remove_const< T >::type operator()(T &) const;
};
```

### Description

#### `as_max` public member functions

1. `remove_const< T >::type operator()(T & arr) const;`

## Struct template `as_min<T, std_valarray_tag>`

`boost::numeric::functional::as_min<T, std_valarray_tag>`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional/valarray.hpp>

template<typename T>
struct as_min<T, std_valarray_tag> : public std::unary_function< T, remove_const< T >::type > {

    // public member functions
    remove_const< T >::type operator()(T &) const;
};
```

### Description

#### `as_min` public member functions

1. `remove_const< T >::type operator()(T & arr) const;`

## Struct template `as_one<T, std_valarray_tag>`

`boost::numeric::functional::as_one<T, std_valarray_tag>`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional/valarray.hpp>

template<typename T>
struct as_one<T, std_valarray_tag> : public std::unary_function< T, remove_const< T >::type > {

    // public member functions
    remove_const< T >::type operator()(T &) const;
};
```

### Description

#### `as_one` public member functions

1. `remove_const< T >::type operator()(T & arr) const;`

## Struct template `as_zero<T, std_valarray_tag>`

`boost::numeric::functional::as_zero<T, std_valarray_tag>`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional/valarray.hpp>

template<typename T>
struct as_zero<T, std_valarray_tag> : public std::unary_function< T, remove_const< T >::type > {

    // public member functions
    remove_const< T >::type operator()(T &) const;
};
```

### Description

#### **as\_zero** public member functions

1. `remove_const< T >::type operator()(T & arr) const;`

## Struct template fdiv<Left, Right, std\_valarray\_tag, RightTag>

`boost::numeric::functional::fdiv<Left, Right, std_valarray_tag, RightTag>`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional/valarray.hpp>

template<typename Left, typename Right, typename RightTag>
struct fdiv<Left, Right, std_valarray_tag, RightTag> : public mpl::if_::type< are_integral<
Left::value_type, Right >, divides< Left, double const >, divides< Left, Right > >
{
};
```

## Struct template max\_assign<Left, Right, std\_valarray\_tag, std\_valarray\_tag>

`boost::numeric::functional::max_assign<Left, Right, std_valarray_tag, std_valarray_tag>`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional/valarray.hpp>

template<typename Left, typename Right>
struct max_assign<Left, Right, std_valarray_tag, std_valarray_tag> : public std::binary_func-
tion< Left, Right, void > {

    // public member functions
    void operator()(Left &, Right &) const;
};
```

### Description

#### **max\_assign** public member functions

1. `void operator()(Left & left, Right & right) const;`

## Struct template `min_assign<Left, Right, std_valarray_tag, std_valarray_tag>`

`boost::numeric::functional::min_assign<Left, Right, std_valarray_tag, std_valarray_tag>`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional/valarray.hpp>

template<typename Left, typename Right>
struct min_assign<Left, Right, std_valarray_tag, std_valarray_tag> : public std::binary_func-
tion< Left, Right, void > {

    // public member functions
    void operator()(Left &, Right &) const;
};
```

### Description

**`min_assign` public member functions**

1. `void operator()(Left & left, Right & right) const;`

## Struct template `promote<bool const, From, void, std_valarray_tag>`

`boost::numeric::functional::promote<bool const, From, void, std_valarray_tag>`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional/valarray.hpp>

template<typename From>
struct promote<bool const, From, void, std_valarray_tag> : public boost::numeric::functional::prom-
ote< bool, From, void, std_valarray_tag >
{
};
```

## Struct template `promote<bool, From, void, std_valarray_tag>`

`boost::numeric::functional::promote<bool, From, void, std_valarray_tag>`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional/valarray.hpp>

template<typename From>
struct promote<bool, From, void, std_valarray_tag> : public std::unary_function< From, bool > {

    // public member functions
    bool operator()(From &) const;
};
```



## Description

**promote public member functions**

```
1. bool operator()(From & arr) const;
```

## Struct template `promote<To, From, std_valarray_tag, std_valarray_tag>`

`boost::numeric::functional::promote<To, From, std_valarray_tag, std_valarray_tag>`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional/valarray.hpp>

template<typename To, typename From>
struct promote<To, From, std_valarray_tag, std_valarray_tag> : public std::unary_function< From, To > {

    // public member functions
    To operator()(From &) const;
};
```

## Description

**promote public member functions**

```
1. To operator()(From & arr) const;
```

## Struct template `promote<ToFrom, ToFrom, std_valarray_tag, std_valarray_tag>`

`boost::numeric::functional::promote<ToFrom, ToFrom, std_valarray_tag, std_valarray_tag>`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional/valarray.hpp>

template<typename ToFrom>
struct promote<ToFrom, ToFrom, std_valarray_tag, std_valarray_tag> : public std::unary_function< ToFrom, ToFrom > {

    // public member functions
    ToFrom & operator()(ToFrom &) const;
};
```

## Description

**promote public member functions**

```
1. ToFrom & operator()(ToFrom & tofrom) const;
```

## Struct template tag<std::valarray< T >>

boost::numeric::functional::tag<std::valarray< T >>

## Synopsis

```
// In header: <boost/accumulators/numeric/functional/valarray.hpp>

template<typename T>
struct tag<std::valarray< T >> {
    // types
    typedef std_valarray_tag type;
};
```

## Header <boost/accumulators/numeric/functional/vector.hpp>

```
namespace boost {
    namespace numeric {
        namespace functional {
            template<typename T> struct as_max<T, std_vector_tag>;
            template<typename T> struct as_min<T, std_vector_tag>;
            template<typename T> struct as_one<T, std_vector_tag>;
            template<typename T> struct as_zero<T, std_vector_tag>;
            template<typename Left, typename Right>
                struct fdiv<Left, Right, std_vector_tag, void>;
            template<typename Left, typename Right>
                struct max_assign<Left, Right, std_vector_tag, std_vector_tag>;
            template<typename Left, typename Right>
                struct min_assign<Left, Right, std_vector_tag, std_vector_tag>;
            template<typename To, typename From>
                struct promote<To, From, std_vector_tag, std_vector_tag>;
            template<typename ToFrom>
                struct promote<ToFrom, ToFrom, std_vector_tag, std_vector_tag>;
            template<typename T, typename A1> struct tag<std::vector< T, A1 >>;
        }
        namespace operators {
            template<typename Left, typename Right>
                unspecified operator/(std::vector< Left > const & left,
                                    Right const & right);
            template<typename Left, typename Right>
                std::vector< typename functional::divides< Left, Right >::result_type >
                operator/(std::vector< Left > const & left,
                        std::vector< Right > const & right);
            template<typename Left, typename Right>
                unspecified operator*(std::vector< Left > const & left,
                                    Right const & right);
            template<typename Left, typename Right>
                unspecified operator*(Left const & left,
                                    std::vector< Right > const & right);
            template<typename Left, typename Right>
                std::vector< typename functional::multiplies< Left, Right >::result_type >
                operator*(std::vector< Left > const & left,
                        std::vector< Right > const & right);
            template<typename Left, typename Right>
                std::vector< typename functional::plus< Left, Right >::result_type >
                operator+(std::vector< Left > const & left,
                        std::vector< Right > const & right);
            template<typename Left, typename Right>
                std::vector< typename functional::minus< Left, Right >::result_type >
                operator-(std::vector< Left > const & left,
```

```
        std::vector< Right > const & right);  
template<typename Left>  
    std::vector< Left > &  
    operator+=(std::vector< Left > & left,  
               std::vector< Left > const & right);  
template<typename Arg>  
    std::vector< typename functional::unary_minus< Arg >::result_type >  
    operator-(std::vector< Arg > const & arg);  
    }  
}  
}
```

## Struct template `as_max<T, std_vector_tag>`

`boost::numeric::functional::as_max<T, std_vector_tag>`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional/vector.hpp>  
  
template<typename T>  
struct as_max<T, std_vector_tag> : public std::unary_function< T, remove_const< T >::type > {  
  
    // public member functions  
    remove_const< T >::type operator()(T &) const;  
};
```

### Description

#### `as_max` public member functions

1. `remove_const< T >::type operator()(T & arr) const;`

## Struct template `as_min<T, std_vector_tag>`

`boost::numeric::functional::as_min<T, std_vector_tag>`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional/vector.hpp>  
  
template<typename T>  
struct as_min<T, std_vector_tag> : public std::unary_function< T, remove_const< T >::type > {  
  
    // public member functions  
    remove_const< T >::type operator()(T &) const;  
};
```

### Description

#### `as_min` public member functions

1. `remove_const< T >::type operator()(T & arr) const;`

## Struct template `as_one<T, std_vector_tag>`

`boost::numeric::functional::as_one<T, std_vector_tag>`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional/vector.hpp>

template<typename T>
struct as_one<T, std_vector_tag> : public std::unary_function< T, remove_const< T >::type > {

    // public member functions
    remove_const< T >::type operator()(T &) const;
};
```

### Description

#### `as_one` public member functions

1. `remove_const< T >::type operator()(T & arr) const;`

## Struct template `as_zero<T, std_vector_tag>`

`boost::numeric::functional::as_zero<T, std_vector_tag>`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional/vector.hpp>

template<typename T>
struct as_zero<T, std_vector_tag> : public std::unary_function< T, remove_const< T >::type > {

    // public member functions
    remove_const< T >::type operator()(T &) const;
};
```

### Description

#### `as_zero` public member functions

1. `remove_const< T >::type operator()(T & arr) const;`

## Struct template `fdiv<Left, Right, std_vector_tag, void>`

`boost::numeric::functional::fdiv<Left, Right, std_vector_tag, void>`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional/vector.hpp>

template<typename Left, typename Right>
struct fdiv<Left, Right, std_vector_tag, void> : public mpl::if_::type< are_integral<
Left::value_type, Right >, divides< Left, double const >, divides< Left, Right > >
{
};
```

### Struct template `max_assign<Left, Right, std_vector_tag, std_vector_tag>`

`boost::numeric::functional::max_assign<Left, Right, std_vector_tag, std_vector_tag>`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional/vector.hpp>

template<typename Left, typename Right>
struct max_assign<Left, Right, std_vector_tag, std_vector_tag> : public std::binary_func<
Left, Right, void > {

    // public member functions
    void operator()(Left & left, Right & right) const;
};
```

### Description

**max\_assign** public member functions

1. `void operator()(Left & left, Right & right) const;`

### Struct template `min_assign<Left, Right, std_vector_tag, std_vector_tag>`

`boost::numeric::functional::min_assign<Left, Right, std_vector_tag, std_vector_tag>`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional/vector.hpp>

template<typename Left, typename Right>
struct min_assign<Left, Right, std_vector_tag, std_vector_tag> : public std::binary_func<
Left, Right, void > {

    // public member functions
    void operator()(Left & left, Right & right) const;
};
```

### Description

**min\_assign** public member functions

1. `void operator()(Left & left, Right & right) const;`

## Struct template `promote<To, From, std_vector_tag, std_vector_tag>`

`boost::numeric::functional::promote<To, From, std_vector_tag, std_vector_tag>`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional/vector.hpp>

template<typename To, typename From>
struct promote<To, From, std_vector_tag, std_vector_tag> : public std::unary_function< From, To > {

    // public member functions
    To operator()(From &) const;
};
```

### Description

`promote` public member functions

1. `To operator()(From & arr) const;`

## Struct template `promote<ToFrom, ToFrom, std_vector_tag, std_vector_tag>`

`boost::numeric::functional::promote<ToFrom, ToFrom, std_vector_tag, std_vector_tag>`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional/vector.hpp>

template<typename ToFrom>
struct promote<ToFrom, ToFrom, std_vector_tag, std_vector_tag> : public std::unary_function< ToFrom, ToFrom > {

    // public member functions
    ToFrom & operator()(ToFrom &) const;
};
```

### Description

`promote` public member functions

1. `ToFrom & operator()(ToFrom & tofrom) const;`

## Struct template `tag<std::vector< T, Al >>`

`boost::numeric::functional::tag<std::vector< T, Al >>`

## Synopsis

```
// In header: <boost/accumulators/numeric/functional/vector.hpp>

template<typename T, typename A1>
struct tag<std::vector< T, A1 >> {
    // types
    typedef std_vector_tag type;
};
```