Boost.Accumulators

Eric Niebler

Table of Contents

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 cacheing 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 `accumulator_set<>`. The type of the sample is the *sample type*.
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 *weight type*.
Feature	An abstract primitive computational entity. When defining an `accumulator_set<>`, users specify the features in which they are interested, and the `accumulator_set<>` figures out which *accumulators* 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, propogating 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 `accumulator_set<>`.

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
`accumulator_set<>`	This is the most important type in the Accumulators Framework. It is a collection of accumulators. A datum pushed into an `accumulator_set<>` 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.
`depends_on<>`	Used to specify which other features a feature depends on.
`feature_of<>`	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.
`as_feature<>`	Used to create an alias for a feature. For example, if there are two features, fast_X and accurate_X, they can be mapped to X(fast) and X(accurate) with `as_feature<>`. This is just syntactic sugar.
`features<>`	An MPL sequence. We can use `features<>` as the second template parameter when declaring an `accumulator_set<>`.
`external<>`	Used when declaring an `accumulator_set<>`. If the weight type is specified with `external<>`, then the weight accumulators are assumed to reside in a separate accumulator set which will be passed in with a named parameter.
`extractor<>`	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 `accumulator_set<>` 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 covariate 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 {                            // Putting your accumulators in the
namespace accumulators {                     // impl namespace has some
namespace impl {                             // advantages. See below.

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

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

    template<typename Args>                  // The accumulate function is the function
    void operator ()(Args const & args)      // 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,
        return this->sum;                    // so we use "dont_care" as the argument type.
    }
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 explictitly 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 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 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 {                // By convention, we put extractors
namespace extract {                     // 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 {                    // By convention, we put extractors
namespace extract {                         // 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::average(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::average()` do something sensible with a `std::vector<>`, for instance, we'll need to partially specialize the `boost::numeric::functional::average<>` 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 average;
}}}
```

If you have some user-defined type `MyDouble` for which you would like to customimze the behavior of `numeric::average()`, you would specialize `numeric::functional::average<>` 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 average<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::average()` will use your specialization of `numeric::functional::average<>` 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
Acc::result_type	*implementation defined*	The type returned by Acc::result().
Acc acc(args)	none	Construct from an argument pack.
Acc acc(acc2)	none	Post: acc.result(args) is equivalent to acc2.result(args)
acc(args)	*unspecified*	
acc.on_drop(args)	*unspecified*	
acc.result(args)	Acc::result_type	

Feature Concept

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

Table 3. Featue Requirements

Expression	Return type	Assertion / Note / Pre- / Post-condition
F::dependencies	*unspecified*	An MPL sequence of other features on which which F depends.
F::is_weight_accumulator	mpl::true_ or mpl::false_	mpl::true_ if the accumulator for this feature should be made external when the weight type for the accumulator set is external<S>, mpl::false_ otherwise.
F::impl	*unspecified*	An MPL Lambda Expression 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 statistial 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::average<sample-type, std::size_t>::res↵
ult_type
  , numeric::functional::average<variate-type, std::size_t>::res↵
ult_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	*none*
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::average<sample-
type, std::size_t>::result_type
          , numeric::functional::average<sample-
type, std::size_t>::result_type
        >
    >::iterator
>
```

Depends On	`count`
	`min`
	`max`

Variants	*none*
Initialization Parameters	`density::cache_size` `density::num_bins`
Accumulator Parameters	*none*
Extractor Parameters	*none*

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::average<sample-type, std::size_t>::result_type`
Depends On	`count` `variance`
Variants	`error_of<immediate_mean>`
Initialization Parameters	*none*
Accumulator Parameters	*none*
Extractor Parameters	*none*
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^2` 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::average<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(quadrat↵
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::average<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::average<sample-type, std::size_t>::result_type` For weights, `numeric::functional::average<weight-type, std::size_t>::result_type` For variates, `numeric::functional::average<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	`mean_of_weights`
	`mean_of_variates<`*`variate-type`*`, `*`variate-tag`*`>`
	`immediate_mean`
	`immediate_mean_of_weights`
	`immediate_mean_of_variates<`*`variate-type`*`, `*`variate-tag`*`>`

Initialization Parameters	*none*
Accumulator Parameters	*none*
Extractor Parameters	*none*
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(0.33333333333333333, 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(1.33333333333333333, 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(0.33333333333333333, 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(1.33333333333333333, 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::average<`*`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::average<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::average<sample-type, std::size_t>::result_type , numeric::functional::average<sample-type, std::size_t>::result_type > >::iterator >`
Depends On	`count`
Variants	*none*
Initialization Parameters	`tag::p_square_cumulative_distribution::num_cells`
Accumulator Parameters	*none*
Extractor Parameters	*none*
Accumulator Complexity	TODO
Extractor Complexity	O(N) where N is `num_cells`

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<> > nor↵
mal(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) )), histo↵
gram[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::average<`*`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::average<sample-type, std::size_t>::res↵
ult_type // u_bar
  , numeric::functional::average<sample-type, std::size_t>::res↵
ult_type // beta_bar
  , numeric::functional::average<sample-type, std::size_t>::res↵
ult_type // xi_hat
>
```

Depends On	`count` In addtion, `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::average<`*`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<> > nor↵
mal(rng, mean_sigma);
boost::variate_generator<boost::lagged_fibonacci607&, boost::exponential_distribution<> > expo↵
nential(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 inplementation 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::average<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	`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_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	`rolling_window_plus1`
Variants	*none*
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_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.

Result Type	`numeric::functional::average<`*`sample-type`*`, std::size_t>::result_type`
Depends On	`rolling_sum` `rolling_count`
Variants	*none*
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

- `rolling_mean_impl`

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 sampless 3. For implementation details, see `skewness_impl`.

Result Type	`numeric::functional::average<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	*`sample-type`* for summing samples *`weight-type`* for summing weights *`variate-type`* for summing variates
Depends On	*none*

Variants	`tag::sum`
	`tag::sum_of_weights`
	`tag::sum_of_variates<`*`variate-type`*`, `*`variate-tag`*`>`
	`tag::sum_kahan` (a.k.a. `tag::sum(kahan)`)
	`tag::sum_of_weights_kahan` (a.k.a. `tag::sum_of_weights(kahan)`)
	`tag::sum_of_variates_kahan<`*`variate-type`*`, `*`variate-tag`*`>`

Initialization Parameters	*none*
Accumulator Parameters	`weight` for summing weights *`variate-tag`* for summing variates
Extractor Parameters	*none*
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 `abstract_tail`

Initialization Parameters	`tag::tail<left-or-right>::cache_size`
Accumulator Parameters	*none*
Extractor Parameters	*none*
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 choherent 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::average<sample-type, std::size_t>::result_type`

Depends On	`count`
	`quantile`
	`non_coherent_tail_mean<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_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-choherent 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::average<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<> > nor↵
mal(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	`tail<left-or-right>`
Variants	`abstract_tail_variate`
Initialization Parameters	`tag::tail<left-or-right>::cache_size`
Accumulator Parameters	*none*
Extractor Parameters	*none*
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::average<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::covariate1>(relat↵
ive)>, tag::tail<right> >
accumulator_t1;

typedef accumulator_set<double, stats<
    tag::tail_variate_means<right, variate_set_type, tag::covariate1>(abso↵
lute)>, tag::tail<right> >
accumulator_t2;

typedef accumulator_set<double, stats<
   tag::tail_variate_means<left, variate_set_type, tag::covariate1>(relative)>, tag::tail<left> >
accumulator_t3;

typedef accumulator_set<double, stats<
   tag::tail_variate_means<left, variate_set_type, tag::covariate1>(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., covariate1 = cov1);
acc1( 50., covariate1 = cov2);
acc1(200., covariate1 = cov3);
acc1( 80., covariate1 = cov4);
acc1( 20., covariate1 = cov5);

acc2(100., covariate1 = cov1);
acc2( 50., covariate1 = cov2);
acc2(200., covariate1 = cov3);
acc2( 80., covariate1 = cov4);
acc2( 20., covariate1 = cov5);

acc3(100., covariate1 = cov1);
acc3( 50., covariate1 = cov2);
acc3(200., covariate1 = cov3);
acc3( 80., covariate1 = cov4);
acc3( 20., covariate1 = cov5);

acc4(100., covariate1 = cov1);
acc4( 50., covariate1 = cov2);
acc4(200., covariate1 = cov3);
acc4( 80., covariate1 = cov4);
acc4( 20., covariate1 = 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).be↵
gin() + 2), 19./70. ); // (17 +  2) /  70 = 19/70
BOOST_CHECK_EQUAL( *(relative_tail_variate_means(acc3, quantile_probability = 0.3).be↵
gin() + 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::average<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::average<sample-type, std::size_t>::result_type >::result_type , numeric::functional::multiplies< weight-type , numeric::functional::average<variate-type, std::size_t>::result_type >::result_type >`
Depends On	`count` `sum_of_weights`

	`weighted_mean` `weighted_mean_of_variates<variate-type, variate-tag>`
Variants	`abstract_weighted_covariance`
Initialization Parameters	*none*
Accumulator Parameters	`weight` *`variate-tag`*
Extractor Parameters	*none*
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::average<weight-
type, std::size_t>::result_type
          , numeric::functional::average<weight-
type, std::size_t>::result_type
        >
    >::iterator
>
```

Depends On	`count`

	`sum_of_weights`
	`min`
	`max`

Variants	*none*
Initialization Parameters	`tag::weighted_density::cache_size` `tag::weighted_density::num_bins`
Accumulator Parameters	`weight`
Extractor Parameters	*none*
Accumulator Complexity	TODO
Extractor Complexity	O(N), when N is `weighted_density::num_bins`

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	`count` `sum_of_weights`
Variants	*none*
Initialization Parameters	`tag::weighted_extended_p_square::probabilities`
Accumulator Parameters	`weight`
Extractor Parameters	*none*
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<> > nor↵
mal1(rng, mean_sigma1);
boost::variate_generator<boost::lagged_fibonacci607&, boost::normal_distribution<> > nor↵
mal2(rng, mean_sigma2);

std::vector<double> probs_uniform, probs_normal1, probs_normal2, probs_normal_exact1, probs_nor↵
mal_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::average< 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, numeric::functional::average<numeric::functional::multiplies<*sample-type*, *weight-type*>::result_type, *weight-type*>::result_type For variates, numeric::functional::average<numeric::functional::multiplies<*variate-type*, *weight-type*>::result_type, *weight-type*>::result_type
Depends On	sum_of_weights The lazy mean of samples depends on weighted_sum The lazy mean of variates depends on weighted_sum_of_variates<>
Variants	weighted_mean_of_variates<*variate-type*, *variate-tag*> immediate_weighted_mean immediate_weighted_mean_of_variates<*variate-type*, *variate-tag*>
Initialization Parameters	*none*
Accumulator Parameters	*none*
Extractor Parameters	*none*
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::average<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 distribu↵
tion 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_nar↵
row(rng, mean_sigma_narrow);

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

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_nar↵
row - 1./sigma/sigma )));
    acc_dens(sample, weight = std::exp(0.5 * (sample - mu) * (sample - mu) * ( 1./sigma_nar↵
row/sigma_narrow - 1./sigma/sigma )));
    acc_cdist(sample, weight = std::exp(0.5 * (sample - mu) * (sample - mu) * ( 1./sigma_nar↵
row/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::average<
    numeric::functional::multiplies<sample-type, weight-type>::res↵
ult_type
  , weight_type
>::result_type
```

Depends On	`count` `sum_of_weights`
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_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^2` algorithm for weighted samples. For more implementation details, see `weighted_p_square_cumulative_distribution_impl`

Result Type

```
iterator_range<
    std::vector<
        std::pair<
            numeric::functional::aver↲
age<weighted_sample, std::size_t>::result_type
          , numeric::functional::aver↲
age<weighted_sample, std::size_t>::result_type
        >
    >::iterator
>
```

	where `weighted_sample` is `numeric::functional::multiplies<`*`sample-type`*`,` *`weight-type`*`>::result_type`
Depends On	`count` `sum_or_weights`
Variants	*none*
Initialization Parameters	`tag::weighted_p_square_cumulative_distribution::num_cells`
Accumulator Parameters	`weight`
Extractor Parameters	*none*
Accumulator Complexity	TODO
Extractor Complexity	O(N) where N is `num_cells`

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 > ac↲
cumulator_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 applaying 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::average< 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<> > nor↵
mal4(rng, mean_sigma4);
boost::variate_generator<boost::lagged_fibonacci607&, boost::normal_distribution<> > nor↵
mal5(rng, mean_sigma5);
boost::variate_generator<boost::lagged_fibonacci607&, boost::normal_distribution<> > nor↵
mal6(rng, mean_sigma6);
boost::variate_generator<boost::lagged_fibonacci607&, boost::normal_distribution<> > nor↵
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::average<
    numeric::functional::multiplies<sample-type, weight-type>::res↵
ult_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 sampless 3. 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.

For implementation details, see `weighted_skewness_impl`.

Result Type	`numeric::functional::average< 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	`numeric::functional::multiplies<`*`sample-type`*`, `*`weight-type`*`>::result_type` for summing weighted samples `numeric::functional::multiplies<`*`variate-type`*`, `*`weight-type`*`>::result_type` for summing weighted variates
Depends On	*none*

Variants	`tag::weighted_sum`
	`tag::weighted_sum_of_variates<`*`variate-type`*`, `*`variate-tag`*`>`
	`tag::weighted_sum_kahan` (a.k.a. tag::weighted_sum(kahan))
	`tag::weighted_sum_of_variates_kahan<`*`variate-type`*`, `*`variate-tag`*`>`

Initialization Parameters	*none*
Accumulator Parameters	`weight` *`variate-tag`* for summing variates
Extractor Parameters	*none*
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::covari↵
ate1> >, 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-choherent 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::average< 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, ep↵
silon );
BOOST_CHECK_CLOSE( non_coherent_weighted_tail_mean(acc0, quantile_probability = 0.975), 0.9875, ep↵
silon );
BOOST_CHECK_CLOSE( non_coherent_weighted_tail_mean(acc0, quantile_probability = 0.99), 0.995, ep↵
silon );
BOOST_CHECK_CLOSE( non_coherent_weighted_tail_mean(acc0, quantile_probability = 0.999), 0.9995, ep↵
silon );
BOOST_CHECK_CLOSE( non_coherent_weighted_tail_mean(acc1, quantile_probability = 0.05), 0.025, ep↵
silon );
BOOST_CHECK_CLOSE( non_coherent_weighted_tail_mean(acc1, quantile_probability = 0.025), 0.0125, ep↵
silon );
BOOST_CHECK_CLOSE( non_coherent_weighted_tail_mean(acc1, quantile_probability = 0.01), 0.005, ep↵
silon );
BOOST_CHECK_CLOSE( non_coherent_weighted_tail_mean(acc1, quantile_probabil↵
ity = 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_sigma1(mu1,1);
boost::normal_distribution<> mean_sigma2(mu2,1);
boost::variate_generator<boost::lagged_fibonacci607&, boost::normal_distribution<> > nor↵
mal1(rng, mean_sigma1);
boost::variate_generator<boost::lagged_fibonacci607&, boost::normal_distribution<> > nor↵
mal2(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 = normal1();
    double sample2 = normal2();
    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::average< 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::co↲
variate1>(relative)>, double >
    acc1( right_tail_cache_size = c );
accumulator_set<double, stats<tag::weighted_tail_variate_means<right, variate_set_type, tag::co↲
variate1>(absolute)>, double >
    acc2( right_tail_cache_size = c );
accumulator_set<double, stats<tag::weighted_tail_variate_means<left, variate_set_type, tag::cov↲
ariate1>(relative)>, double >
    acc3( left_tail_cache_size = c );
accumulator_set<double, stats<tag::weighted_tail_variate_means<left, variate_set_type, tag::cov↲
ariate1>(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, covariate1 = cov1);
acc1( 50., weight = 0.9, covariate1 = cov2);
acc1(200., weight = 1.0, covariate1 = cov3);
acc1( 80., weight = 1.1, covariate1 = cov4);
acc1( 20., weight = 1.2, covariate1 = cov5);

acc2(100., weight = 0.8, covariate1 = cov1);
acc2( 50., weight = 0.9, covariate1 = cov2);
acc2(200., weight = 1.0, covariate1 = cov3);
acc2( 80., weight = 1.1, covariate1 = cov4);
acc2( 20., weight = 1.2, covariate1 = cov5);

acc3(100., weight = 0.8, covariate1 = cov1);
acc3( 50., weight = 0.9, covariate1 = cov2);
acc3(200., weight = 1.0, covariate1 = cov3);
acc3( 80., weight = 1.1, covariate1 = cov4);
acc3( 20., weight = 1.2, covariate1 = cov5);

acc4(100., weight = 0.8, covariate1 = cov1);
acc4( 50., weight = 0.9, covariate1 = cov2);
acc4(200., weight = 1.0, covariate1 = cov3);
acc4( 80., weight = 1.1, covariate1 = cov4);
acc4( 20., weight = 1.2, covariate1 = cov5);

// check relative risk contributions
BOOST_CHECK_EQUAL( *(relative_weighted_tail_variate_means(acc1, quantile_probability = 0.7).be↲
gin()    ), (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).be↲
gin() + 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).be↲
gin() + 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).be↲
gin() + 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).be↵
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).be↵
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).be↵
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).be↵
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).be↵
gin() + 1), (0.8*20 + 1.0*64)/1.8 );
BOOST_CHECK_EQUAL( *(weighted_tail_variate_means(acc2, quantile_probability = 0.7).be↵
gin() + 2), (0.8*30 + 1.0*40)/1.8 );
BOOST_CHECK_EQUAL( *(weighted_tail_variate_means(acc2, quantile_probability = 0.7).be↵
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).be↵
gin() + 1), (0.9*4 + 1.2*2)/2.1 );
BOOST_CHECK_EQUAL( *(weighted_tail_variate_means(acc4, quantile_probability = 0.3).be↵
gin() + 2), (0.9*17 + 1.2*2)/2.1 );
BOOST_CHECK_EQUAL( *(weighted_tail_variate_means(acc4, quantile_probability = 0.3).be↵
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).be↵
gin()    ), 1.0*46/(1.0*200) );
BOOST_CHECK_EQUAL( *(relative_weighted_tail_variate_means(acc1, quantile_probability = 0.9).be↵
gin() + 1), 1.0*64/(1.0*200) );
BOOST_CHECK_EQUAL( *(relative_weighted_tail_variate_means(acc1, quantile_probability = 0.9).be↵
gin() + 2), 1.0*40/(1.0*200) );
BOOST_CHECK_EQUAL( *(relative_weighted_tail_variate_means(acc1, quantile_probability = 0.9).be↵
gin() + 3), 1.0*50/(1.0*200) );
BOOST_CHECK_EQUAL( *(relative_weighted_tail_variate_means(acc3, quantile_probability = 0.1).be↵
gin()    ), 1.2*2/(1.2*20) );
BOOST_CHECK_EQUAL( *(relative_weighted_tail_variate_means(acc3, quantile_probability = 0.1).be↵
gin() + 1), 1.2*2/(1.2*20) );
BOOST_CHECK_EQUAL( *(relative_weighted_tail_variate_means(acc3, quantile_probability = 0.1).be↵
gin() + 2), 1.2*2/(1.2*20) );
BOOST_CHECK_EQUAL( *(relative_weighted_tail_variate_means(acc3, quantile_probability = 0.1).be↵
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).be↵
gin() + 1), 1.0*64/1.0 );
BOOST_CHECK_EQUAL( *(weighted_tail_variate_means(acc2, quantile_probability = 0.9).be↵
gin() + 2), 1.0*40/1.0 );
BOOST_CHECK_EQUAL( *(weighted_tail_variate_means(acc2, quantile_probability = 0.9).be↵
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).be↵
gin() + 1), 1.2*2/1.2 );
BOOST_CHECK_EQUAL( *(weighted_tail_variate_means(acc4, quantile_probability = 0.1).be↵
gin() + 2), 1.2*2/1.2 );
BOOST_CHECK_EQUAL( *(weighted_tail_variate_means(acc4, quantile_probability = 0.1).be↵
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::average< 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<> > nor↲
mal(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 Accumulator> struct droppable_accumulator_base;
    template<typename Accumulator> struct droppable_accumulator;
    template<typename Accumulator> struct with_cached_result;
    template<typename Sample, typename Features, typename Weight = void>
      struct accumulator_set;
    template<typename Feature> struct extractor;
    template<typename Feature> struct feature_of;
    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 Fea↵
ture, mpl::na) > 
      struct depends_on;
    template<BOOST_PP_ENUM_PARAMS_WITH_A_DEFAULT(BOOST_ACCUMULATORS_MAX_FEATURES, typename Fea↵
ture, mpl::na) > 
      struct features;
    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 ValueType, typename Tag> struct value;
      template<typename Tag> struct value_tag;
      template<typename Referent, typename Tag> struct reference;
      template<typename Tag> struct reference_tag;
      template<typename Type, typename Tag = void, 
               typename AccumulatorSet = void> 
        struct external;
      template<typename Feature> struct droppable;
    }
  }
}
```

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 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 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 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 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 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/destruct
  template<typename Args> droppable_accumulator(Args const &);
  droppable_accumulator(droppable_accumulator const &);
};
```

Description

`droppable_accumulator` public construct/copy/destruct

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

2.
```
droppable_accumulator(droppable_accumulator const & that);
```

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;
```

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.

acccumulator_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 extractor

boost::accumulators::extractor

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

Extracts the result associated with Feature from the specified accumulator_set.

`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 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

Synopsis

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

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

Description

depends_on

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 Fea↵
ture, mpl::na) >
struct features : public mpl::vector< Feature1, Feature2,...> {
};
```

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 dont_care;
    struct accumulator_base;
  }
}
```

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 &);
```

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);
```

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::fea↵
ture_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<tag::value< ValueType, Tag >>

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/destruct
  count_impl(dont_care);

  // public member functions
  void operator()(dont_care);
  result_type result(dont_care) const;
};
```

Description

`count_impl` public construct/copy/destruct

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 feature_of<tag::covariance< VariateType, VariateTag >>;
    template<typename VariateType, typename VariateTag>
      struct as_weighted_feature<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 EnableIf = void>
        struct outer_product_base;
      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>;
    }
    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 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 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::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::accumulat↵
ors::feature_of< tag::covariance< VariateType, VariateTag > >
{
};
```

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 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 >::res↵
ult_type > >
{
  // types
  typedef ublas::matrix< typename functional::multiplies< typename Left::value_type, type↵
name 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 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<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 >
{
};
```

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 > {
};
```

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::exten↲
ded_p_square::probabilities named paramter
};
```

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 feature_of<tag::extended_p_square_quantile>;
    template<> struct feature_of<tag::extended_p_square_quantile_quadratic>;
    template<> struct as_weighted_feature<tag::extended_p_square_quantile>;
    template<> struct feature_of<tag::weighted_extended_p_square_quantile>;
    template<>
      struct as_weighted_feature<tag::extended_p_square_quantile_quadratic>;
    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_exten↵
ded_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 > im↵
pl;
};
```

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 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::fea↵
ture_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::fea↵
ture_of< tag::quantile > {
};
```

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 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 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::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::accumulat↵
ors::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 >, mo↵
ment< 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::kurtos↵
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(lazy)>;
    template<> struct as_feature<tag::mean(immediate)>;
    template<> struct as_feature<tag::mean_of_weights(lazy)>;
    template<> struct as_feature<tag::mean_of_weights(immediate)>;
    template<typename VariateType, typename VariateTag>
      struct as_feature<tag::mean_of_variates< VariateType, VariateTag >(lazy)>;
    template<typename VariateType, typename VariateTag>
      struct as_feature<tag::mean_of_variates< VariateType, VariateTag >(immediate)>;
    template<> struct feature_of<tag::immediate_mean>;
    template<> struct feature_of<tag::immediate_mean_of_weights>;
    template<typename VariateType, typename VariateTag>
      struct feature_of<tag::immediate_mean_of_variates< VariateType, VariateTag >>;
    template<> struct as_weighted_feature<tag::mean>;
    template<> struct feature_of<tag::weighted_mean>;
    template<> struct as_weighted_feature<tag::immediate_mean>;
    template<> struct feature_of<tag::immediate_weighted_mean>;
    template<typename VariateType, typename VariateTag>
      struct as_weighted_feature<tag::mean_of_variates< VariateType, VariateTag >>;
    template<typename VariateType, typename VariateTag>
      struct feature_of<tag::weighted_mean_of_variates< VariateType, VariateTag >>;
    template<typename VariateType, typename VariateTag>
      struct as_weighted_feature<tag::immediate_mean_of_variates< VariateType, VariateTag >>;
    template<typename VariateType, typename VariateTag>
      struct feature_of<tag::immediate_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 mean;
      struct immediate_mean;
      struct mean_of_weights;
      struct immediate_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 mean

boost::accumulators::tag::mean

Synopsis

```
// In header: <boost/accumulators/statistics/mean.hpp>


struct mean : public boost::accumulators::depends_on< count, sum > {
};
```

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 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 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 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 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_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_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 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 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 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 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 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 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 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 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 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 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 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 > >
{
};
```

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 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 >> : pub↵
lic boost::accumulators::feature_of< tag::immediate_mean_of_variates< VariateType, VariateTag ↵
> >
{
};
```

Header <boost/accumulators/statistics/median.hpp>

```
namespace boost {
  namespace accumulators {
    template<> struct as_feature<tag::median(with_p_square_quantile)>;
    template<> struct as_feature<tag::median(with_density)>;
    template<>
      struct as_feature<tag::median(with_p_square_cumulative_distribution)>;
    template<> struct feature_of<tag::with_density_median>;
    template<>
      struct feature_of<tag::with_p_square_cumulative_distribution_median>;
    template<> struct as_weighted_feature<tag::median>;
    template<> struct feature_of<tag::weighted_median>;
    template<> struct as_weighted_feature<tag::with_density_median>;
    template<> struct feature_of<tag::with_density_weighted_median>;
    template<>
      struct as_weighted_feature<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_dis↵
tribution_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_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_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 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_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 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 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 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 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 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::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::accumulat↵
ors::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_value)>;
    template<typename LeftRight>
      struct as_feature<tag::peaks_over_threshold< LeftRight >(with_threshold_probability)>;
    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 as_weighted_feature<tag::peaks_over_threshold< LeftRight >>;
    template<typename LeftRight>
      struct feature_of<tag::weighted_peaks_over_threshold< LeftRight >>;
    template<typename LeftRight>
      struct as_weighted_feature<tag::peaks_over_threshold_prob< 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::average< 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 ξ and β 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.

$$\widehat{F}(x) = [1-F_n(u)]G_{\xi,\beta,u}(x) + F_n(u).$$

It can be shown that $\widehat{F}(x)$ is a generalized Pareto distribution $G_{\xi,\bar{\beta},\bar{u}}(x)$ with $\bar{\beta} = \beta[1-F_n(u)]^{\xi}$ and $\bar{u} = u - \bar{\beta}\left\{[1-F_n(u)]^{-\xi}-1\right\}/\xi$. By inverting $\widehat{F}(x)$, one obtains an estimator for the α-quantile,

Equation 5.

$$\hat{q}_\alpha = \bar{u} + \frac{\bar{\beta}}{\xi}\left[(1-\alpha)^{-\xi} - 1\right],$$

and similarly an estimator for the (coherent) tail mean,

Equation 6.

$$\widehat{CTM}_\alpha = \hat{q}_\alpha - \frac{\bar{\beta}}{\xi - 1}(1-\alpha)^{-\xi},$$

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_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_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 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::fea↲
ture_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::fea↵
ture_of< tag::abstract_peaks_over_threshold >
{
};
```

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 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::accumulat↵
ors::feature_of< tag::peaks_over_threshold< LeftRight > >
{
};
```

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::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::accumu↵
lators::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_value)>;
    template<typename LeftRight>
      struct as_feature<tag::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_feature<tag::weighted_pot_quantile< LeftRight >(with_threshold_probability)>;
    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 as_weighted_feature<tag::pot_quantile< LeftRight >>;
    template<typename LeftRight>
      struct feature_of<tag::weighted_pot_quantile< LeftRight >>;
    template<typename LeftRight>
      struct as_weighted_feature<tag::pot_quantile_prob< 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_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::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::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_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 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::fea↲
ture_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::fea↲
ture_of< tag::quantile > {
};
```

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 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 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::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::fea↵
ture_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_value)>;
    template<typename LeftRight>
      struct as_feature<tag::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_feature<tag::weighted_pot_tail_mean< LeftRight >(with_threshold_probability)>;
    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 as_weighted_feature<tag::pot_tail_mean< LeftRight >>;
    template<typename LeftRight>
      struct feature_of<tag::weighted_pot_tail_mean< LeftRight >>;
    template<typename LeftRight>
      struct as_weighted_feature<tag::pot_tail_mean_prob< 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_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::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::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_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 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::fea↵
ture_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 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 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 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::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 {
    namespace extract {
      extractor< tag::rolling_mean > const rolling_mean;
    }
    namespace impl {
    }
    namespace tag {
      struct 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 rolling_mean

boost::accumulators::tag::rolling_mean

Synopsis

```
// In header: <boost/accumulators/statistics/rolling_mean.hpp>


struct 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
};
```

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_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_plus1;
      struct rolling_window;
    }
  }
}
```

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_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
};
```

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
};
```

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::skew↵
ness > {
};
```

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<> struct feature_of<tag::weighted_sum>;
    template<typename VariateType, typename VariateTag>
      struct feature_of<tag::sum_of_variates< VariateType, VariateTag >>;
    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 sum;
      struct sum_of_weights;
      struct abstract_sum_of_variates;
    }
  }
}
```

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 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 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 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 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 > {
};
```

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 >
{
};
```

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::weighted_sum_kahan>;
    template<> struct feature_of<tag::sum_kahan>;
    template<> struct feature_of<tag::sum_of_weights_kahan>;
    template<typename VariateType, typename VariateTag>
      struct feature_of<tag::sum_of_variates_kahan< VariateType, VariateTag >>;
    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::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 > {
};
```

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 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 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 >
{
};
```

Header <boost/accumulators/statistics/tail.hpp>

```
namespace boost {
  namespace accumulators {
    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>;
    template<typename LeftRight> struct feature_of<tag::tail< LeftRight >>;
    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 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 {
};
```

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 >
{
};
```

Header <boost/accumulators/statistics/tail_mean.hpp>

```
namespace boost {
  namespace accumulators {
    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 as_weighted_feature<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 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::fea↵
ture_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::fea↵
ture_of< tag::abstract_non_coherent_tail_mean >
{
};
```

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::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::accumulat↵
ors::feature_of< tag::non_coherent_tail_mean< LeftRight > >
{
};
```

Header <boost/accumulators/statistics/tail_quantile.hpp>

```
namespace boost {
  namespace accumulators {
    template<typename LeftRight>
      struct feature_of<tag::tail_quantile< LeftRight >>;
    template<typename LeftRight>
      struct as_weighted_feature<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 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 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::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 feature_of<tag::absolute_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 as_weighted_feature<tag::absolute_tail_variate_means< LeftRight, VariateType, Vari↵
ateTag >>;
    template<typename LeftRight, typename VariateType, typename VariateTag>
     struct feature_of<tag::absolute_weighted_tail_variate_means< LeftRight, VariateType, Vari↵
ateTag >>;
    template<typename LeftRight, typename VariateType, typename VariateTag>
     struct as_weighted_feature<tag::relative_tail_variate_means< LeftRight, VariateType, Vari↵
ateTag >>;
    template<typename LeftRight, typename VariateType, typename VariateTag>
     struct feature_of<tag::relative_weighted_tail_variate_means< LeftRight, VariateType, Vari↵
ateTag >>;
    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 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 >> : pub↵
lic boost::accumulators::feature_of< tag::abstract_absolute_tail_variate_means >
{
};
```

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 >> : pub↵
lic boost::accumulators::feature_of< tag::abstract_relative_tail_variate_means >
{
};
```

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, Variat↵
eTag >> {
  // types
  typedef tag::absolute_weighted_tail_variate_means< LeftRight, VariateType, VariateTag > type;
};
```

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, Variat↵
eTag >> : public boost::accumulators::feature_of< tag::absolute_tail_variate_means< LeftRight, ↵
VariateType, VariateTag > >
{
};
```

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, Variat↵
eTag >> {
  // types
  typedef tag::relative_weighted_tail_variate_means< LeftRight, VariateType, VariateTag > type;
};
```

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, Variat↵
eTag >> : 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(lazy)>;
    template<> struct as_feature<tag::variance(immediate)>;
    template<> struct feature_of<tag::lazy_variance>;
    template<> struct as_weighted_feature<tag::variance>;
    template<> struct feature_of<tag::weighted_variance>;
    template<> struct as_weighted_feature<tag::lazy_variance>;
    template<> struct feature_of<tag::lazy_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(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_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 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 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::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 > {
};
```

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 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 >
{
};
```

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_mo↵
ment< 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(lazy)>;
    template<> struct as_feature<tag::weighted_mean(immediate)>;
    template<typename VariateType, typename VariateTag> 
      struct as_feature<tag::weighted_mean_of_variates< VariateType, VariateTag >(lazy)>;
    template<typename VariateType, typename VariateTag> 
      struct as_feature<tag::weighted_mean_of_variates< VariateType, VariateTag >(immediate)>;
    namespace extract {
      extractor< tag::mean > const weighted_mean;
    }
    namespace impl {
    }
    namespace tag {
      struct weighted_mean;
      struct immediate_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 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 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 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 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 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;
};
```

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;
};
```

Header <boost/accumulators/statistics/weighted_median.hpp>

```
namespace boost {
  namespace accumulators {
    template<> struct as_feature<tag::weighted_median(with_p_square_quantile)>;
    template<> struct as_feature<tag::weighted_median(with_density)>;
    template<>
      struct as_feature<tag::weighted_median(with_p_square_cumulative_distribution)>;
    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_me↵
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_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;
};
```

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;
};
```

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_cumu↲
lative_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_dis↵
tribution;
```

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_distribu↵
tion_num_cells
{
  // types
  typedef accumulators::impl::weighted_p_square_cumulative_distribution_impl< mpl::_1, mpl::_2 > im↵
pl;
};
```

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_medi↵
an > 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_me↵
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_value)>;
    template<typename LeftRight>
      struct as_feature<tag::weighted_peaks_over_threshold< LeftRight >(with_threshold_probabil↵
ity)>;
    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_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;
};
```

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;
};
```

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_mo↵
ment< 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 weighted_sum;
      struct abstract_weighted_sum_of_variates;
    }
  }
}
```

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 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 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 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 >> : pub↵
lic 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 >(ab↵
solute)>;
    template<typename LeftRight, typename VariateType, typename VariateTag>
      struct as_feature<tag::weighted_tail_variate_means< LeftRight, VariateType, VariateTag >(re↵
lative)>;
    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_vari↵
ate_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 >(abso↵
lute)> {
  // 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 >(relat↵
ive)> {
  // 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(lazy)>;
    template<> struct as_feature<tag::weighted_variance(immediate)>;
    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(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;
};
```

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;
};
```

Header <boost/accumulators/statistics/with_error.hpp>

Header <boost/accumulators/statistics_fwd.hpp>

```
namespace boost {
  namespace accumulators {
    template<BOOST_PP_ENUM_PARAMS_WITH_A_DEFAULT(BOOST_ACCUMULATORS_MAX_FEATURES, typename Fea↵
ture, mpl::na) > 
      struct stats;
    template<BOOST_PP_ENUM_PARAMS_WITH_A_DEFAULT(BOOST_ACCUMULATORS_MAX_FEATURES, typename Fea↵
ture, mpl::na) > 
      struct with_error;
    struct lazy;
    struct immediate;
    struct right;
    struct left;
    struct absolute;
    struct relative;
    struct with_density;
    struct with_p_square_cumulative_distribution;
    struct with_p_square_quantile;
    struct with_threshold_value;
    struct with_threshold_probability;
    struct weighted;
    struct unweighted;
    struct linear;
    struct quadratic;
    struct regular;
    struct for_median;
    struct kahan;
    namespace extract {
      extractor< tag::quantile > const quantile;
      extractor< tag::tail_mean > const tail_mean;
    }
    namespace 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> struct kurtosis_impl;
      template<typename Sample> struct max_impl;
      template<typename Sample> struct median_impl;
      template<typename Sample> struct with_density_median_impl;
      template<typename Sample> 
        struct with_p_square_cumulative_distribution_median_impl;
      template<typename Sample> struct min_impl;
      template<typename Sample, typename SumFeature = tag::sum> struct mean_impl;
      template<typename Sample, typename Tag = tag::sample> 
        struct immediate_mean_impl;
      template<typename N, typename Sample> struct moment_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 p_square_cumulative_distribution_impl;
      template<typename Sample, typename Impl> struct p_square_quantile_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 coherent_tail_mean_impl;
template<typename Sample, typename LeftRight> 
  struct non_coherent_tail_mean_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> struct lazy_variance_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_kurtosis_impl;
template<typename Sample> struct weighted_median_impl;
template<typename Sample> struct with_density_weighted_median_impl;
template<typename Sample, typename Weight> 
  struct with_p_square_cumulative_distribution_weighted_median_impl;
template<typename Sample, typename Weight, typename Tag> 
  struct weighted_mean_impl;
template<typename Sample, typename Weight, typename Tag> 
  struct immediate_weighted_mean_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_extended_p_square_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> 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 non_coherent_weighted_tail_mean_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> 
  struct lazy_weighted_variance_impl;
template<typename Sample, typename Weight, typename MeanFeature, 
         typename Tag> 
  struct weighted_variance_impl;
template<typename Sample> struct rolling_window_plus1_impl;
template<typename Sample> struct rolling_window_impl;
template<typename Sample> struct rolling_sum_impl;
template<typename Sample> struct rolling_count_impl;
template<typename Sample> struct rolling_mean_impl;
```

```
    }
    namespace tag {
      struct quantile;
      struct tail_mean;
      template<typename VariateType, typename VariateTag> struct covariance;
      template<typename Feature> struct error_of;
      template<typename VariateType, typename VariateTag> struct mean_of_variates;
      template<typename VariateType, typename VariateTag> 
        struct immediate_mean_of_variates;
      template<int N> struct moment;
      template<typename LeftRight> struct peaks_over_threshold;
      template<typename LeftRight> struct peaks_over_threshold_prob;
      template<typename LeftRight> struct pot_tail_mean;
      template<typename LeftRight> struct pot_tail_mean_prob;
      template<typename LeftRight> struct pot_quantile;
      template<typename LeftRight> struct pot_quantile_prob;
      template<typename VariateType, typename VariateTag> struct sum_of_variates;
      template<typename VariateType, typename VariateTag> 
        struct sum_of_variates_kahan;
      template<typename LeftRight> struct tail;
      template<typename LeftRight> struct coherent_tail_mean;
      template<typename LeftRight> struct non_coherent_tail_mean;
      template<typename LeftRight> struct tail_quantile;
      template<typename VariateType, typename VariateTag, typename LeftRight> 
        struct tail_variate;
      template<typename LeftRight> struct tail_weights;
      template<typename VariateType, typename VariateTag, typename LeftRight> 
        struct right_tail_variate;
      template<typename VariateType, typename VariateTag, typename LeftRight> 
        struct left_tail_variate;
      template<typename LeftRight, typename VariateType, typename VariateTag> 
        struct tail_variate_means;
      template<typename LeftRight, typename VariateType, typename VariateTag> 
        struct absolute_tail_variate_means;
      template<typename LeftRight, typename VariateType, typename VariateTag> 
        struct relative_tail_variate_means;
      template<typename VariateType, typename VariateTag> 
        struct weighted_covariance;
      template<typename VariateType, typename VariateTag> 
        struct weighted_mean_of_variates;
      template<typename VariateType, typename VariateTag> 
        struct immediate_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 LeftRight> struct weighted_tail_quantile;
      template<typename LeftRight> struct non_coherent_weighted_tail_mean;
      template<typename LeftRight, typename VariateType, typename VariateTag> 
        struct weighted_tail_variate_means;
      template<typename LeftRight, typename VariateType, typename VariateTag> 
        struct absolute_weighted_tail_variate_means;
      template<typename LeftRight, typename VariateType, typename VariateTag> 
        struct relative_weighted_tail_variate_means;
      template<typename VariateType, typename VariateTag> 
        struct weighted_sum_of_variates;
    }
  }
}
```

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 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 $\mathrm{Cov}(X,X')$, where X is a sample and X' is a variate, is given by:

Equation 7.

$$\hat{c}_n = \frac{n-1}{n}\hat{c}_{n-1} + \frac{1}{n-1}(X_n - \hat{\mu}_n)(X'_n - \hat{\mu}'_n), \quad n \ge 2, \quad \hat{c}_1 = 0,$$

$\hat{\mu}_n$ 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 $\xi_{p_1}, \ldots, \xi_{p_m}$ 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-parablic 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 generalizess the P^2 algorithm of

R. Jain and I. Chlamtac, The P^2 algorithmus 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 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/destruct
  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 8.

$$\hat{g}_2 = \frac{\widehat{m}_n^{(4)} - 4\widehat{m}_n^{(3)}\hat{\mu}_n + 6\widehat{m}_n^{(2)}\hat{\mu}_n^2 - 3\hat{\mu}_n^4}{\left(\widehat{m}_n^{(2)} - \hat{\mu}_n^2\right)^2} - 3,$$

where $\widehat{m}_n^{(i)}$ are the i -th moment and $\hat{\mu}_n$ the mean (first moment) of the n samples.

`kurtosis_impl` public construct/copy/destruct

1.
```
kurtosis_impl(dont_care);
```

`kurtosis_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 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 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_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/destruct
  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/destruct

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 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 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 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 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 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 9.

$$\hat{q}_\alpha = \bar{u} + \frac{\bar{\beta}}{\xi}\left[(1-\alpha)^{-\xi} - 1\right]$$

for a right or left extreme quantile, $\bar{u}$, $\bar{\beta}$ and ξ 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 10.

$$\widehat{CTM}_\alpha = \hat{q}_\alpha - \frac{\bar{\beta}}{\xi - 1}(1-\alpha)^{-\xi},$$

where $\bar{u}$, $\bar{\beta}$ and ξ 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 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 algorithmus 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 algorithmus fordynamic 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 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 sampless 3. The skewness can also be expressed by the simple moments:

Equation 11.

$$\hat{g}_1 = \frac{\widehat{m}_n^{(3)} - 3\widehat{m}_n^{(2)}\hat{\mu}_n + 2\hat{\mu}_n^3}{\left(\widehat{m}_n^{(2)} - \hat{\mu}_n^2\right)^{3/2}}$$

where $\widehat{m}_n^{(i)}$ 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 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 12.

$$\widehat{CTM}_{n,\alpha}^{\mathrm{right}}(X) = \widehat{NCTM}_{n,\alpha}^{\mathrm{right}}(X) + \frac{1}{\lceil n(1-\alpha)\rceil}\hat{q}_{n,\alpha}(X)\left(1-\alpha-\frac{1}{n}\lceil n(1-\alpha)\rceil\right)$$

Equation 13.

$$\widehat{CTM}_{n,\alpha}^{\mathrm{left}}(X) = \widehat{NCTM}_{n,\alpha}^{\mathrm{left}}(X) + \frac{1}{\lceil n\alpha\rceil}\hat{q}_{n,\alpha}(X)\left(\alpha-\frac{1}{n}\lceil n\alpha\rceil\right)$$

`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 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 α the quantile level:

Equation 14.

$$\widehat{NCTM}^{\text{right}}_{n,\alpha}(X) = \frac{1}{\lceil n(1-\alpha) \rceil} \sum_{i=\lceil \alpha n \rceil}^{n} X_{i:n}$$

Equation 15.

$$\widehat{NCTM}^{\text{left}}_{n,\alpha}(X) = \frac{1}{\lceil n\alpha \rceil} \sum_{i=1}^{\lceil \alpha n \rceil} X_{i:n}$$

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 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 α based on order statistics requires the chaching 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 16.

$$\hat{q}_{n,\alpha} = X_{\lceil \alpha n \rceil : n}$$

`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;
```

Struct 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 17.

$$\widehat{ATM}^{\mathrm{right}}_{n,\alpha}(X,j) = \frac{1}{\lceil n(1-\alpha)\rceil}\sum_{i=\lceil \alpha n\rceil}^{n}\xi_{j,i}$$

Equation 18.

$$\widehat{ATM}^{\mathrm{left}}_{n,\alpha}(X,j) = \frac{1}{\lceil n\alpha\rceil}\sum_{i=1}^{\lceil n\alpha\rceil}\xi_{j,i}$$

Equation 19.

$$\widehat{RTM}^{\mathrm{right}}_{n,\alpha}(X,j) = \frac{\sum_{i=\lceil n\alpha\rceil}^{n}\xi_{j,i}}{\lceil n(1-\alpha)\rceil\widehat{NCTM}^{\mathrm{right}}_{n,\alpha}(X)}$$

Equation 20.

$$\widehat{RTM}^{\mathrm{left}}_{n,\alpha}(X,j) = \frac{\sum_{i=1}^{\lceil n\alpha\rceil}\xi_{j,i}}{\lceil n\alpha\rceil\widehat{NCTM}^{\mathrm{left}}_{n,\alpha}(X)}$$

`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 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/destruct
  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 21.

$$\sigma_n^2 = M_n^{(2)} - \mu_n^2.$$

where

Equation 22.

$$\mu_n = \frac{1}{n} \sum_{i=1}^{n} x_i.$$

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 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 σ_n^2 according to the formula

Equation 23.

$$\sigma_n^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_n)^2 = \frac{n-1}{n}\sigma_{n-1}^2 + \frac{1}{n-1}(x_n - \mu_n)^2.$$

where

Equation 24.

$$\mu_n = \frac{1}{n}\sum_{i=1}^{n} x_i.$$

is the estimate of the sample mean and n is the number of samples.

Note that the sample variance is not defined for $n <= 1$.

A simplification can be obtained by the approximate recursion

Equation 25.

$$\sigma_n^2 \approx \frac{n-1}{n}\sigma_{n-1}^2 + \frac{1}{n}(x_n - \mu_n)^2.$$

because the difference

Equation 26.

$$\left(\frac{1}{n-1} - \frac{1}{n}\right)(x_n - \mu_n)^2 = \frac{1}{n(n-1)}(x_n - \mu_n)^2.$$

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 $\mathrm{Cov}(X,X')$, where X is a sample and X' a variate, is given by:

Equation 27.

$$\hat{c}_n = \frac{\bar{w}_n - w_n}{\bar{w}_n}\hat{c}_{n-1} + \frac{w_n}{\bar{w}_n - w_n}(X_n - \hat{\mu}_n)(X'_n - \hat{\mu}'_n), \quad n \ge 2, \quad \hat{c}_1 = 0,$$

$\hat{\mu}_n$ 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_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/destruct
  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 28.

$$\hat{g}_2 = \frac{\widehat{m}_n^{(4)} - 4\widehat{m}_n^{(3)}\hat{\mu}_n + 6\widehat{m}_n^{(2)}\hat{\mu}_n^2 - 3\hat{\mu}_n^4}{\left(\widehat{m}_n^{(2)} - \hat{\mu}_n^2\right)^2} - 3,$$

where $\widehat{m}_n^{(i)}$ 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/destruct

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_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 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_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 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/destruct
  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 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 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_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 $\xi_{p_1}, \ldots, \xi_{p_m}$ 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-parablic 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 generalizess the P^2 algorithm of

R. Jain and I. Chlamtac, The P^2 algorithmus 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_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 algorithmus 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 algorithmus 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_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 simple moments:

Equation 29.

$$\hat{g}_1 = \frac{\widehat{m}_n^{(3)} - 3\widehat{m}_n^{(2)}\hat{\mu}_n + 2\hat{\mu}_n^3}{\left(\widehat{m}_n^{(2)} - \hat{\mu}_n^2\right)^{3/2}}$$

where $\widehat{m}_n^{(i)}$ 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 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 30.

$$\lambda = \inf\left\{ l \left| \frac{1}{\bar{w}_n} \sum_{i=1}^{l} w_i \geq \alpha \right. \right\}$$

smallest samples (left tail) or the weighted mean of the

Equation 31.

$$n + 1 - \rho = n + 1 - \sup\left\{ r \left| \frac{1}{\bar{w}_n} \sum_{i=r}^{n} w_i \geq (1 - \alpha) \right. \right\}$$

largest samples (right tail) above a quantile $\hat{q}_\alpha$ of level α, n being the total number of sample and $\bar{w}_n$ the sum of all n weights:

Equation 32.

$$\widehat{NCTM}_{n,\alpha}^{\mathrm{left}}(X) = \frac{\sum_{i=1}^{\lambda} w_i X_{i:n}}{\sum_{i=1}^{\lambda} w_i},$$

Equation 33.

$$\widehat{NCTM}_{n,\alpha}^{\mathrm{right}}(X) = \frac{\sum_{i=\rho}^{n} w_i X_{i:n}}{\sum_{i=\rho}^{n} w_i}.$$

`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 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 α 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 34.

$$\lambda = \inf\left\{ l \left| \frac{1}{\bar{w}_n} \sum_{i=1}^{l} w_i \geq \alpha \right. \right\}$$

and

Equation 35.

$$\rho = \sup\left\{ r \left| \frac{1}{\bar{w}_n} \sum_{i=r}^{n} w_i \geq (1-\alpha) \right. \right\},$$

n being the number of samples and $\bar{w}_n$ the sum of all weights.

`weighted_tail_quantile_impl` public construct/copy/destruct

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/destruct
  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 36.

$$\lambda = \inf\left\{ l \left| \frac{1}{\bar{w}_n} \sum_{i=1}^{l} w_i \geq \alpha \right. \right\}$$

smallest samples (left tail) or the weighted mean of the

Equation 37.

$$n + 1 - \rho = n + 1 - \sup\left\{ r \left| \frac{1}{\bar{w}_n} \sum_{i=r}^{n} w_i \geq (1 - \alpha) \right. \right\}$$

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 38.

$$\widehat{ATM}_{n,\alpha}^{\mathrm{right}}(X, j) = \frac{1}{\sum_{i=\rho}^{n} w_i} \sum_{i=\rho}^{n} w_i \xi_{j,i}$$

Equation 39.

$$\widehat{ATM}_{n,\alpha}^{\mathrm{left}}(X, j) = \frac{1}{\sum_{i=1}^{\lambda}} \sum_{i=1}^{\lambda} w_i \xi_{j,i}$$

Equation 40.

$$\widehat{RTM}_{n,\alpha}^{\mathrm{right}}(X, j) = \frac{\sum_{i=\rho}^{n} w_i \xi_{j,i}}{\sum_{i=\rho}^{n} w_i \widehat{NCTM}_{n,\alpha}^{\mathrm{right}}(X)}$$

Equation 41.

$$\widehat{RTM}_{n,\alpha}^{\mathrm{left}}(X, j) = \frac{\sum_{i=1}^{\lambda} w_i \xi_{j,i}}{\sum_{i=1}^{\lambda} w_i \widehat{NCTM}_{n,\alpha}^{\mathrm{left}}(X)}$$

`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 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 42.

$$\hat{\sigma}_n^2 = \hat{m}_n^{(2)} - \hat{\mu}_n^2,$$

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 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 43.

$$\hat{\sigma}_n^2 = \frac{\bar{w}_n - w_n}{\bar{w}_n}\hat{\sigma}_{n-1}^2 + \frac{w_n}{\bar{w}_n - w_n}\left(X_n - \hat{\mu}_n\right)^2, \quad n \geq 2, \quad \hat{\sigma}_0^2 = 0.$$

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 smaples. Note that the sample variance is not defined for $n <= 1$.

`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 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 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_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 & 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 : public accumulator_base {
  // construct/copy/destruct
  rolling_mean_impl(dont_care);

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

Description

`rolling_mean_impl` public construct/copy/destruct

1.
```
rolling_mean_impl(dont_care);
```

`rolling_mean_impl` public member functions

1. ```
   template<typename Args> result_type result(Args const & args) const;
   ```

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 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 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< Vari↵
ateType, 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 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< Vari↵
ateType, VariateTag > >
{
};
```

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 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 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_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 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 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 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_coher↵
ent_tail_mean< LeftRight > >
{
};
```

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 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_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 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 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 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 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_coher↵
ent_tail_mean< LeftRight >, tail_variate< VariateType, VariateTag, LeftRight > >
{
};
```

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_coher↵
ent_tail_mean< LeftRight >, tail_variate< VariateType, VariateTag, 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 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 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_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 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 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 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_coher↵
ent_weighted_tail_mean< LeftRight >, tail_variate< VariateType, VariateTag, LeftRight >, ↵
tail_weights< 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_coher↵
ent_weighted_tail_mean< LeftRight >, tail_variate< VariateType, VariateTag, LeftRight >, ↵
tail_weights< 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 stats

boost::accumulators::stats

Synopsis

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

template<BOOST_PP_ENUM_PARAMS_WITH_A_DEFAULT(BOOST_ACCUMULATORS_MAX_FEATURES, typename Fea↵
ture, mpl::na) >
struct stats : public mpl::vector< Stat1, Stat2,...> {
};
```

Description

An MPL sequence of statistics.

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 Fea↵
ture, mpl::na) >
struct with_error {
};
```

Struct lazy

boost::accumulators::lazy

Synopsis

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


struct lazy {
};
```

Struct immediate

boost::accumulators::immediate

Synopsis

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


struct immediate {
};
```

Struct right

boost::accumulators::right

Synopsis

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


struct right {
};
```

Struct left

boost::accumulators::left

Synopsis

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


struct left {
};
```

Struct absolute

boost::accumulators::absolute

Synopsis

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


struct absolute {
};
```

Struct relative

boost::accumulators::relative

Synopsis

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


struct relative {
};
```

Struct with_density

boost::accumulators::with_density

Synopsis

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


struct with_density {
};
```

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_value

boost::accumulators::with_threshold_value

Synopsis

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


struct with_threshold_value {
};
```

Struct with_threshold_probability

boost::accumulators::with_threshold_probability

Synopsis

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


struct with_threshold_probability {
};
```

Struct weighted

boost::accumulators::weighted

Synopsis

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


struct weighted {
};
```

Struct unweighted

boost::accumulators::unweighted

Synopsis

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


struct unweighted {
};
```

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 for_median

boost::accumulators::for_median

Synopsis

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


struct for_median {
};
```

Struct kahan

boost::accumulators::kahan

Synopsis

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


struct kahan {
};
```

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 zero;
    template<typename T> struct one_or_default;
    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 Left, typename Right> struct left_ref;
      template<typename Left, typename Right, typename EnableIf = void>
        struct plus_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 EnableIf = void>
        struct minus_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 EnableIf = void>
        struct multiplies_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 EnableIf = void>
        struct divides_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 EnableIf = void>
        struct modulus_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 EnableIf = void>
        struct greater_base;
      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_equal_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 less_base;
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_equal_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 equal_to_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 not_equal_to_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 assign_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 plus_assign_base;
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 minus_assign_base;
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 multiplies_assign_base;
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 divides_assign_base;
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 modulus_assign_base;
template<typename Left, typename Right, 
         typename LeftTag = typename tag<Left>::type, 
         typename RightTag = typename tag<Right>::type> 
  struct modulus_assign;
template<typename Arg, typename EnableIf = void> struct unary_plus_base;
template<typename Arg, typename Tag = typename tag<Arg>::type> 
  struct unary_plus;
template<typename Arg, typename EnableIf = void> struct unary_minus_base;
```

```
    template<typename Arg, typename Tag = typename tag<Arg>::type>
      struct unary_minus;
    template<typename Arg, typename EnableIf = void> struct complement_base;
    template<typename Arg, typename Tag = typename tag<Arg>::type>
      struct complement;
    template<typename Arg, typename EnableIf = void> struct logical_not_base;
    template<typename Arg, typename Tag = typename tag<Arg>::type>
      struct logical_not;
    template<typename Left, typename Right, typename EnableIf>
      struct min_assign_base;
    template<typename Left, typename Right, typename EnableIf>
      struct max_assign_base;
    template<typename Left, typename Right, typename EnableIf>
      struct average_base;

    template<typename Left, typename Right>
     struct average_base<Left, Right, typename enable_if< are_integral< Left, Right > >::type>;

    template<typename To, typename From, typename EnableIf> struct promote_base;

    template<typename ToFrom> struct promote_base<ToFrom, ToFrom, void>;

    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 EnableIf> struct as_max_base;
    template<typename Arg, typename EnableIf> struct as_zero_base;
    template<typename Arg, typename EnableIf> struct as_one_base;
    template<typename To, typename From, typename ToTag, typename FromTag>
      struct promote;
    template<typename Left, typename Right, typename LeftTag,
             typename RightTag>
      struct min_assign;
    template<typename Left, typename Right, typename LeftTag,
             typename RightTag>
      struct max_assign;
    template<typename Left, typename Right, typename LeftTag,
             typename RightTag>
      struct average;
    template<typename Arg, typename Tag> struct as_min;
    template<typename Arg, typename Tag> struct as_max;
    template<typename Arg, typename Tag> struct as_zero;
    template<typename Arg, typename Tag> struct as_one;
  }
  namespace op {
    struct plus;
    struct minus;
    struct multiplies;
    struct divides;
    struct modulus;
    struct greater;
    struct greater_equal;
    struct less;
    struct less_equal;
    struct equal_to;
    struct not_equal_to;
    struct assign;
    struct plus_assign;
    struct minus_assign;
    struct multiplies_assign;
    struct divides_assign;
```

```
      struct modulus_assign;
      struct unary_plus;
      struct unary_minus;
      struct complement;
      struct logical_not;
      template<typename To> struct promote;
      struct min_assign;
      struct max_assign;
      struct average;
      struct as_min;
      struct as_max;
      struct as_zero;
      struct as_one;
    }
  }
}
```

Struct template left_ref

boost::numeric::functional::left_ref

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right>
struct left_ref {
  // types
  typedef Left & type;
};
```

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 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 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< ↵
Right >())>
{

  // 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 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 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 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 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, typeof(lvalue< Left >()/lvalue< ↵
Right >())>
{

  // 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 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 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, typeof(lvalue< Left >()%lvalue< ↵
Right >())>
{

  // 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 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 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

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_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 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 less_base

boost::numeric::functional::less_base

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right, typename EnableIf = void> 
struct less_base : public std::binary_function< Left, Right, typeof(lvalue< Left >()< lvalue< ↵
Right >())>
{

  // public member functions
  result_type operator()(Left &, Right &) const;
};
```

Description

`less_base` public member functions

1.
```
result_type operator()(Left & left, Right & right) const;
```

Returns: left < right

Struct template less

boost::numeric::functional::less

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 : 
  public boost::numeric::functional::less_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, typeof(lvalue< Left ↲
>()<=lvalue< Right >())>
{

  // 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 less_equal

boost::numeric::functional::less_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 less_equal :
  public boost::numeric::functional::less_equal_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, typeof(lvalue< Left >()==lvalue< ↵
Right >())>
{

  // 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 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 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, typeof(lvalue< Left ↵
>()!=lvalue< Right >())>
{

  // 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 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 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, typeof(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 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 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_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 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, typeof(lvalue< Left >()-
=lvalue< Right >())>
{

  // 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_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 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_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 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, typeof(lvalue< Left ↵
>()/=lvalue< Right >())>
{

  // 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_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 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, typeof(lvalue< Left ↵
>()%=lvalue< Right >())>
{

  // 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_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 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 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_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_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 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 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 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, typeof(!lvalue< Arg >())>
{

  // 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 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 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 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 average_base

boost::numeric::functional::average_base

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right, typename EnableIf>
struct average_base :
  public boost::numeric::functional::divides< Left, Right >
{
};
```

Struct template average_base<Left, Right, typename enable_if< are_integral< Left, Right > >::type>

boost::numeric::functional::average_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 average_base<Left, Right, typename enable_if< are_integral< Left, Right > >::type> :
  public boost::numeric::functional::divides< double const, double const  >
{
};
```

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 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_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_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 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 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 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 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 average

boost::numeric::functional::average

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename Left, typename Right, typename LeftTag, typename RightTag>
struct average :
  public boost::numeric::functional::average_base< Left, Right, void >
{
};
```

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_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_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_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 plus

boost::numeric::op::plus

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>


struct plus {
};
```

Struct minus

boost::numeric::op::minus

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>


struct minus {
};
```

Struct multiplies

boost::numeric::op::multiplies

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>


struct multiplies {
};
```

Struct divides

boost::numeric::op::divides

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>


struct divides {
};
```

Struct modulus

boost::numeric::op::modulus

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>


struct modulus {
};
```

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 equal_to

boost::numeric::op::equal_to

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>


struct equal_to {
};
```

Struct not_equal_to

boost::numeric::op::not_equal_to

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>


struct not_equal_to {
};
```

Struct assign

boost::numeric::op::assign

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>


struct assign {
};
```

Struct plus_assign

boost::numeric::op::plus_assign

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>


struct plus_assign {
};
```

Struct minus_assign

boost::numeric::op::minus_assign

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>


struct minus_assign {
};
```

Struct multiplies_assign

boost::numeric::op::multiplies_assign

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>


struct multiplies_assign {
};
```

Struct divides_assign

boost::numeric::op::divides_assign

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>


struct divides_assign {
};
```

Struct modulus_assign

boost::numeric::op::modulus_assign

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>


struct modulus_assign {
};
```

Struct unary_plus

boost::numeric::op::unary_plus

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>


struct unary_plus {
};
```

Struct unary_minus

boost::numeric::op::unary_minus

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>


struct unary_minus {
};
```

Struct complement

boost::numeric::op::complement

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>


struct complement {
};
```

Struct logical_not

boost::numeric::op::logical_not

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>


struct logical_not {
};
```

Struct template promote

boost::numeric::op::promote

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename To>
struct promote {
};
```

Struct min_assign

boost::numeric::op::min_assign

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>


struct min_assign {
};
```

Struct max_assign

boost::numeric::op::max_assign

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>


struct max_assign {
};
```

Struct average

boost::numeric::op::average

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>


struct average {
};
```

Struct as_min

boost::numeric::op::as_min

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>


struct as_min {
};
```

Struct as_max

boost::numeric::op::as_max

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>


struct as_max {
};
```

Struct as_zero

boost::numeric::op::as_zero

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>


struct as_zero {
};
```

Struct as_one

boost::numeric::op::as_one

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>


struct as_one {
};
```

Struct template default_

boost::numeric::default_

Synopsis

```
// In header: <boost/accumulators/numeric/functional.hpp>

template<typename T>
struct default_ {
  // types
  typedef T value_type;

  // public static functions

  // public member functions
  operator T const &() const;

  // public data members
  static T const value;
};
```

Description

`default_` public static functions

`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 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 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_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::com↲
plex< 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::com↲
plex< 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 tag<std::valarray< T >>;
      template<typename Left, typename Right> 
        struct min_assign<Left, Right, std_valarray_tag, std_valarray_tag>;
      template<typename Left, typename Right> 
        struct max_assign<Left, Right, std_valarray_tag, std_valarray_tag>;
      template<typename Left, typename Right, typename RightTag> 
        struct average<Left, Right, std_valarray_tag, RightTag>;
      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 From> struct promote<bool, From, void, std_valarray_tag>;
      template<typename From> 
        struct promote<bool const, From, void, std_valarray_tag>;
      template<typename T> struct as_min<T, std_valarray_tag>;
      template<typename T> struct as_max<T, std_valarray_tag>;
      template<typename T> struct as_zero<T, std_valarray_tag>;
      template<typename T> struct as_one<T, std_valarray_tag>;
    }
    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 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;
};
```

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 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 average<Left, Right, std_valarray_tag, RightTag>

boost::numeric::functional::average<Left, Right, std_valarray_tag, RightTag>

Synopsis

```
// In header: <boost/accumulators/numeric/functional/valarray.hpp>

template<typename Left, typename Right, typename RightTag>
struct average<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 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_func↵
tion< 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_func↵
tion< ToFrom, ToFrom > {

  // public member functions
  ToFrom & operator()(ToFrom &) const;
};
```

Description

`promote` public member functions

1.
```
ToFrom & operator()(ToFrom & tofrom) const;
```

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<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::pro↵
mote< bool, From, void, std_valarray_tag >
{
};
```

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_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_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 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;
```

Header <boost/accumulators/numeric/functional/vector.hpp>

```
namespace boost {
  namespace numeric {
    namespace functional {
      template<typename T, typename Al> struct tag<std::vector< T, Al >>;
      template<typename Left, typename Right>
        struct min_assign<Left, Right, std_vector_tag, std_vector_tag>;
      template<typename Left, typename Right>
        struct max_assign<Left, Right, std_vector_tag, std_vector_tag>;
      template<typename Left, typename Right>
        struct average<Left, Right, std_vector_tag, void>;
      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> struct as_min<T, std_vector_tag>;
      template<typename T> struct as_max<T, std_vector_tag>;
      template<typename T> struct as_zero<T, std_vector_tag>;
      template<typename T> struct as_one<T, std_vector_tag>;
    }
    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 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 Al>
struct tag<std::vector< T, Al >> {
  // types
  typedef std_vector_tag type;
};
```

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↵
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 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↵
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 average<Left, Right, std_vector_tag, void>

boost::numeric::functional::average<Left, Right, std_vector_tag, void>

Synopsis

```
// In header: <boost/accumulators/numeric/functional/vector.hpp>

template<typename Left, typename Right> 
struct average<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 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< To↵
From, ToFrom > {

  // public member functions
  ToFrom & operator()(ToFrom &) const;
};
```

Description

`promote` public member functions

1.
```
ToFrom & operator()(ToFrom & tofrom) 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_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_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 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;
```