5/18/2016

Boost::accumulators

Bugsquashing Seminar 18.05.2016

Fabian Isensee

Willkommen im DKFZ!

dkfz.
DEUTSCHES KREBSFORSCHUNGSZENTRUM
IN DER HELMHOLTZ-GEMEINSCHAFT

50 Jahre – Forschen für ein Leben ohne Krebs

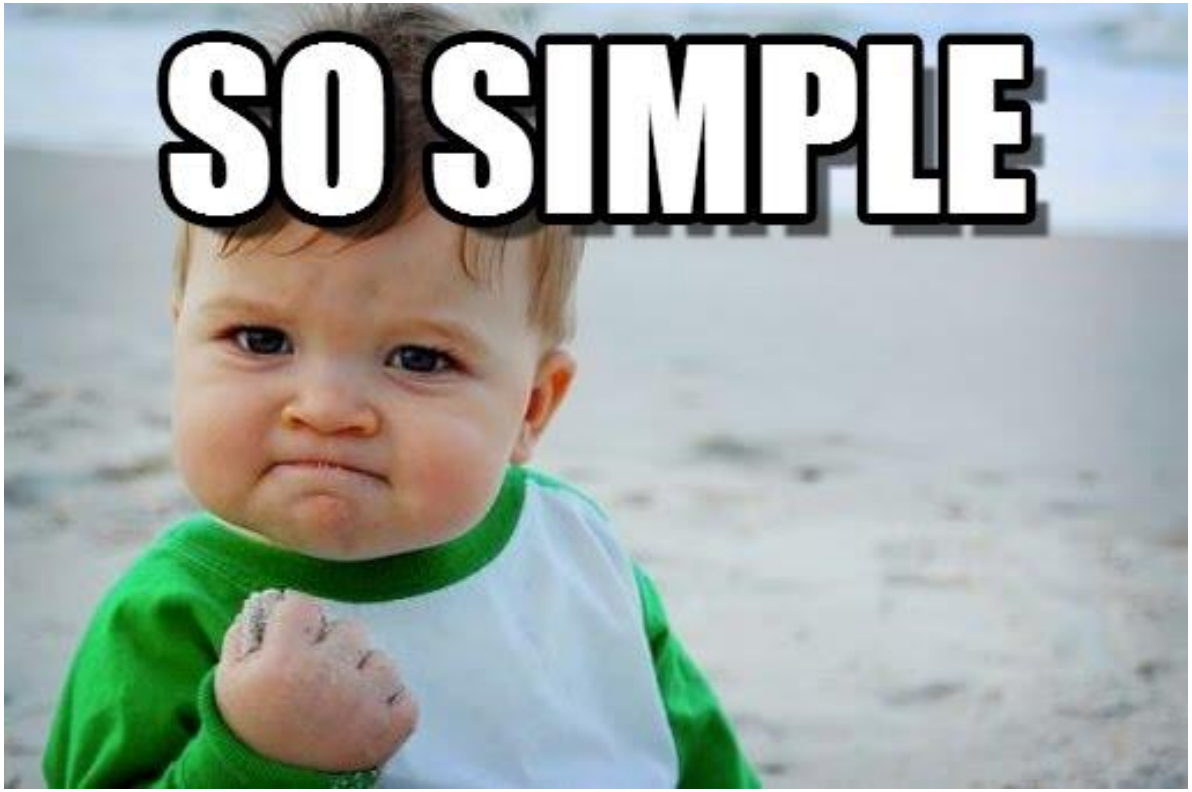
boost::accumulators – Introduction

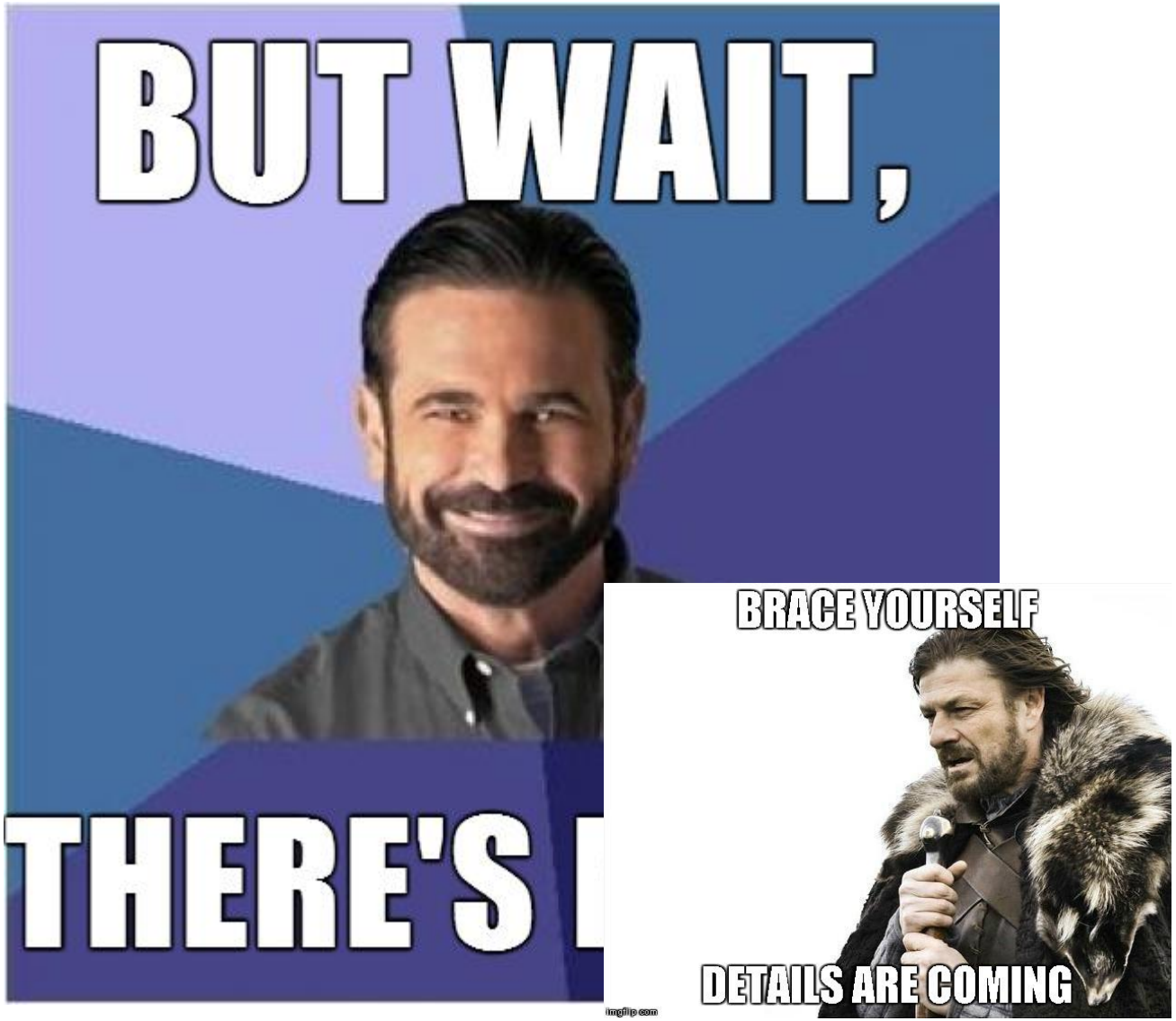
- Incremental computation of statistics
- Large variety of implemented measures
- Easy to use
- boost is MITK dependency

boost::accumulators – Simple Example

```cpp
#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(1.8);
    acc(1.3, weight = 2.5);
    acc(7.1, weight = 0.5);

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

    return 0;
}
```

sample type statistics


SO SIMPLE


BUT WAIT,
THERE'S
BRACE YOURSELF
DETAILS ARE COMING
imgflip.com

boost::accumulators – Advanced Example

```
using namespace boost::accumulators;


int main()
{
    // initialize accumulator
    accumulator_set <double, features<
        tag::mean,
        tag::min,
        tag::max,
        tag::tail<left>,
        tag::lazy_variance
        > > acc(tag::tail<left>::cache_size = 100);

    // initialize vector with 1 mio samples randomly drawn from [0, 100]
    long nElements = 1000000;
    double range = 100.0;
    std::vector<double> vec(nElements, 0.0);
    for (unsigned int i=0; i < nElements; i++)
    {
        vec[i] = (rand()/(double)RAND_MAX) * range;
    }

    // bad:
    acc = std::for_each(vec.begin(), vec.end(), acc);
    // good:
    std::for_each(vec.begin(), vec.end(), std::ref(acc));

    // extract results
    std::cout << mean(acc) << std::endl;
    std::cout << extract_result<tag::min>(acc) << std::endl;
    extractor< tag::max > max_;
    std::cout << max_(acc) << std::endl;
    std::cout << (max)(acc) << std::endl;
    std::cout << lazy_variance(acc) << std::endl;
}
```

boost::accumulators – Defining an Accumulator

```
using namespace boost::accumulators;


int main()
{
    // initialize accumulator
    accumulator_set <double, features<
        tag::mean,
        tag::min,
        tag::max,
        tag::tail<left>,
        tag::lazy_variance
        > > acc(tag::tail<left>::cache_size = 100);

    double range = 100.0;
    std::vector<double> vec(nElements, 0.0);
    for (unsigned int i=0; i < nElements; i++)
    {
        vec[i] = (rand()/(double)RAND_MAX) * range;
    }

    // bad:
    acc = std::for_each(vec.begin(), vec.end(), acc);
    // good:
    std::for_each(vec.begin(), vec.end(), std::ref(acc));

    // extract results
    std::cout << mean(acc) << std::endl;
    std::cout << extract_result<tag::min>(acc) << std::endl;
    extractor< tag::max > max_;
    std::cout << max_(acc) << std::endl;
    std::cout << (max)(acc) << std::endl;
    std::cout << lazy_variance(acc) << std::endl;
}
```

sample type statistics

boost::accumulators – Defining an Accumulator

```
// initialize accumulator
accumulator_set <double, features<
    tag::mean,
    tag::min,
    tag::max,
    tag::tail<left>,
    tag::lazy_variance
    > > acc(tag::tail<left>::cache_size = 100);
```

- Statistics are defined as template parameters
- boost resolves dependency tree (compile time!) between stats for efficient computation
  - `lazy_variance` depends on `tag::moment<2>` and `tag::mean`
  - `mean` depends on `tag::sum` (implicitly added)

boost::accumulators – Adding Samples

```
using namespace boost::accumulators;


int main()
{
    // initialize accumulator
    accumulator_set <double, features<
        tag::mean,
        tag::min,
        tag::max,
        tag::tail<left>,
        tag::lazy_variance
        > > acc(tag::tail<left>::cache_size = 100);

    // initialize vector with 1 mio samples randomly drawn from [0, 100]
    long nElements = 1000000;
    double range = 100.0;
    std::vector<double> vec(nElements, 0.0);
    for (unsigned int i=0; i < nElements; i++)
    {
        vec[i] = (rand()/(double)RAND_MAX) * range;
    }

    // bad:
    acc = std::for_each(vec.begin(), vec.end(), acc);
    // good:
    std::for_each(vec.begin(), vec.end(), std::ref(acc));

    // extract results
    std::cout << mean(acc) << std::endl;
    std::cout << extract_result<tag::min>(acc) << std::endl;
    extractor< tag::max > max_;
    std::cout << max_(acc) << std::endl;
    std::cout << (max)(acc) << std::endl;
    std::cout << lazy_variance(acc) << std::endl;
}
```

boost::accumulators – Adding Samples

```
acc(1.2);
```

```
std::for_each(vec.begin(), vec.end(), std::ref(acc));
```

- boost updates inner state whenever a sample is added:
  - `min`, `max`, `sum`, `tail`
- Lazy evaluation of `mean` and `lazy_variance` on request

boost::accumulators – Adding Samples

- Bad: `acc` is copied. Inefficient if stats such as `tag::tail<left>` are included

```
// add values to accumulator
// bad:
acc = std::for_each(vec.begin(), vec.end(), acc);
```

- Good: use reference of acc

```
// good:
std::for_each(vec.begin(), vec.end(), std::ref(acc));
```

boost::accumulators – Extracting Results

```
using namespace boost::accumulators;


int main()
{
    // initialize accumulator
    accumulator_set <double, features<
        tag::mean,
        tag::min,
        tag::max,
        tag::tail<left>,
        tag::lazy_variance
        > > acc(tag::tail<left>::cache_size = 100);

    // initialize vector with 1 mio samples randomly drawn from [0, 100]
    long nElements = 1000000;
    double range = 100.0;
    std::vector<double> vec(nElements, 0.0);
    for (unsigned int i=0; i < nElements; i++)
    {
        vec[i] = (rand()/(double)RAND_MAX) * range;
    }

    // bad:
    acc = std::for_each(vec.begin(), vec.end(), acc);
    // good:
    std::for_each(vec.begin(), vec.end(), std::ref(acc));

    // extract results
    std::cout << mean(acc) << std::endl;
    std::cout << extract_result<tag::min>(acc) << std::endl;
    extractor< tag::max > max_;
    std::cout << max_(acc) << std::endl;
    std::cout << (max)(acc) << std::endl;
    std::cout << lazy_variance(acc) << std::endl;
}
```

boost::accumulators – Extracting Results

- Simple and sufficient for most use cases:

```
// extract results
std::cout << mean(acc) << std::endl;
```

- Being explicit:

```
std::cout << extract_result<tag::min>(acc) << std::endl;
```

- Windows <3:

```
extractor< tag::max > max_;
std::cout << max_(acc) << std::endl;
```

or

```
std::cout << (max)(acc) << std::endl;
```

(`min` and `max` are preprocessor macros defined in WinDef.h)

boost::accumulators – DIY

Writing your own accumulator

- Define accumulator itself (derive from `accumulator_base`):
  - `constructor`
  - `operator()`
  - `result()`
- Define new feature
  - Inherits from `depends_on<>` for dependency tree
- Define new extractor
  - `extractor< tag::my_feature > const my_feature = {}`

#IfboostWasItk

This one is for the typedef fanatics:

```
typedef double SampleType;
typedef tag::tail<left> TailType;
typedef features<tag::mean, tag::min, tag::max, TailType, tag::lazy_variance> FeaturesType;
typedef accumulator_set<SampleType, FeaturesType> AccumulatorType;

// initialize accumulator
AccumulatorType acc(TailType::cache_size = 100);
```

(

```
// initialize accumulator
accumulator_set < double, features <
    tag::mean,
    tag::min,
    tag::max,
    tag::tail<left>,
    tag::lazy_variance
    > > acc(tag::tail<left>::cache_size = 100);
```

)

boost::accumulators – Summary

boost::accumulators are:

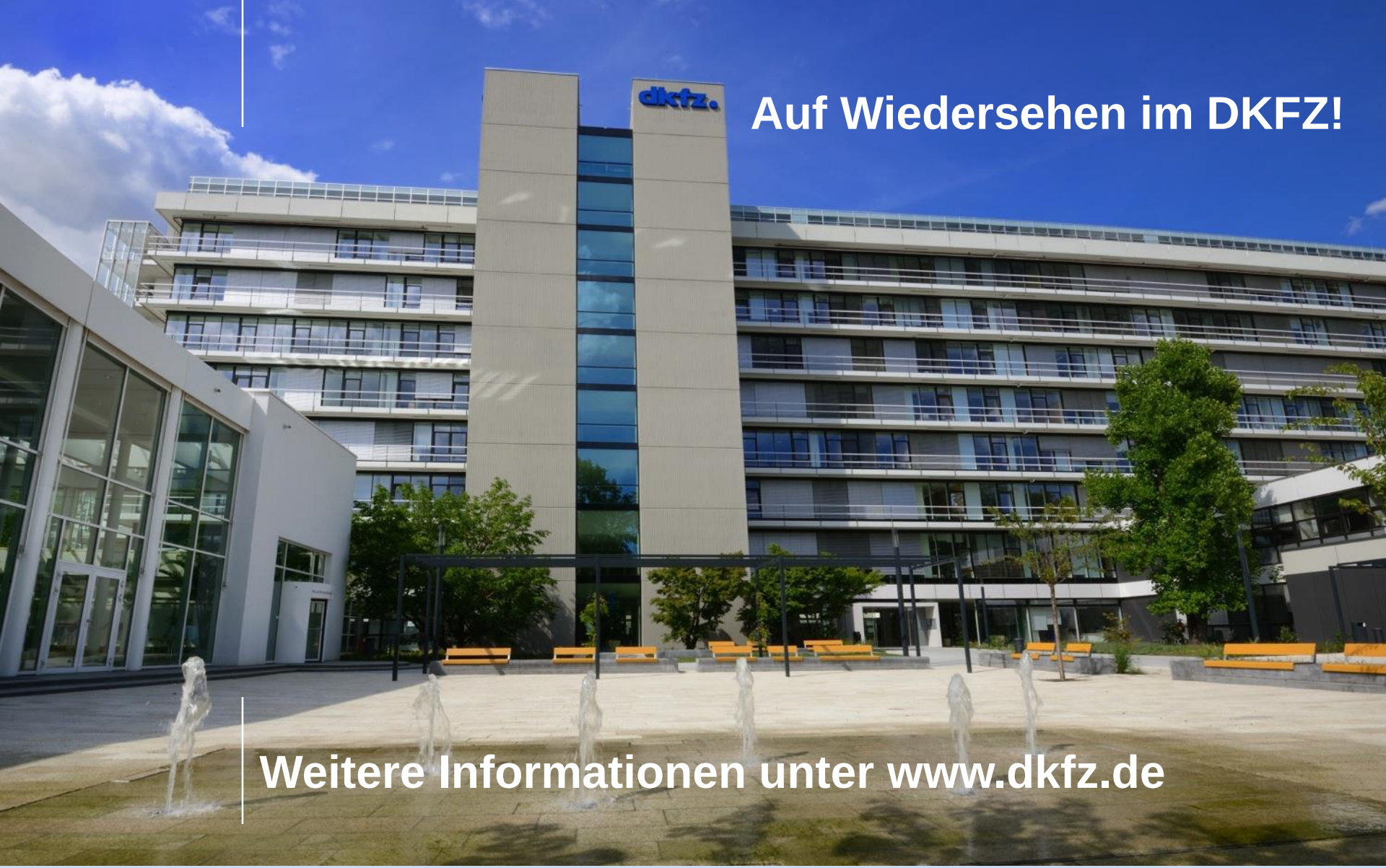
- Easy to use
- Powerful
- Fast
- Expandible

=> No reason not to use them for your statistics!

boost::accumulators – Further Reading

http://www.boost.org/doc/libs/1_60_0/doc/html/accumulators/user_s_guide.html

(boost documentation – great stuff!)



Auf Wiedersehen im DKFZ!

Weitere Informationen unter www.dkfz.de

dkfz.
DEUTSCHES
KREBSFORSCHUNGSZENTRUM
IN DER HELMHOLTZ-GEMEINSCHAFT

50 Jahre – Forschen für
ein Leben ohne Krebs