
peakingduck Documentation

Thomas Stainer

Apr 01, 2020

CONTENTS:

1	About	1
2	Status	3
3	Building	5
3.1	Installation	5
3.2	1D Data Processing	5
3.3	Peak Finding	5
3.4	Extending in C++	5
3.5	Extending with Python	5
3.6	peakingduck Python Package	5
3.6.1	peakingduck.core module	6
3.6.2	peakingduck.io module	12
3.6.3	peakingduck.plotting module	12
3.6.4	peakingduck.util module	13
3.7	peakingduck Header Only Library	14
3.7.1	peakingduck::constants namespace	15
3.7.2	peakingduck::core namespace	15
3.7.3	peakingduck::io namespace	24
3.7.4	peakingduck::util namespace	24
4	Indices and tables	29
	Python Module Index	31
	Index	33

ABOUT

Peaking identification is crucial for gamma spectroscopy and nuclear analysis. Conventional methods (although included) are not great at finding peaks in areas of low statistics and often fail for multiplet identification (overlapping peaks). A new method involving deep learning methods has been developed to improve both precision and recall of peaks. This library contains some traditional algorithms for peak identification and some neural networks used for more modern approaches. The intention is to provide further analysis in the future (peak fitting, background subtraction, etc) but for the minute just focuses on peak identification. This can also be extended (in theory) to any 1 dimension data set containing labelled peaks.

**CHAPTER
TWO**

STATUS

Very much a work in progress, but is expected to have the first version (0.0.1) ready for release in 2 months.

BUILDING

It is header only C++ so nothing to build (only unit tests) if using it in C++. If you want python bindings enabled then it needs building (default will build them), as below.

```
git clone --recursive -j8 https://github.com/fispact/peakingduck
cd peakingduck
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release -DBUILD_PY_BINDINGS=ON ..
make -j4```
```

Note: Project uses cmake (> 3.2) to build peaking duck.

3.1 Installation

3.2 1D Data Processing

3.3 Peak Finding

3.4 Extending in C++

3.5 Extending with Python

3.6 peakingduck Python Package

Peaking duck library using pybind11

ToDo: write

A list:

- Entry 1
- Entry 2

3.6.1 peakingduck.core module

class peakingduck.core.**ChunkedSimplePeakFinder** (*threshold=0.05, nchunks=10*)

Bases: PEAKINGDUCK.core.IPeakFinder

Breaks the spectrum up into nchunks applying the threshold relative to that chunk

find (*self: PEAKINGDUCK.core.IPeakFinder, arg0: PEAKINGDUCK.core.NumericalData*) →
List[PEAKINGDUCK.core.PeakInfo]
Identifies potential peaks in the data

class peakingduck.core.**ChunkedThresholdPeakFilter**

Bases: PEAKINGDUCK.core.IProcess

Simple threshold local/chunked peak filter

class peakingduck.core.**GlobalThresholdPeakFilter**

Bases: PEAKINGDUCK.core.IProcess

Simple threshold global peak filter

class peakingduck.core.**Histogram**

Bases: pybind11_builtins.pybind11_object

Represents a basic 1D histogram

Energies vs values.

property X

property Y

class peakingduck.core.**HistogramChannelBased**

Bases: pybind11_builtins.pybind11_object

Represents a basic 1D histogram

Channels vs values.

property X

property Y

class peakingduck.core.**IPeakFinder**

Bases: pybind11_builtins.pybind11_object

Interface for peak finding algorithms

Operates on numerical data (filtered or unfiltered). Never mutates the input (always const process).

Returns A list of peaks

Return type PeakList

find (*self: PEAKINGDUCK.core.IPeakFinder, arg0: PEAKINGDUCK.core.NumericalData*) →
List[PEAKINGDUCK.core.PeakInfo]
Identifies potential peaks in the data

class peakingduck.core.**IProcess**

Bases: pybind11_builtins.pybind11_object

Interface for all process algorithms

Operates on numerical data. Never mutates the input (always const process).

Returns A new numerical array.

go (*self*: *PEAKINGDUCK.core.IProcess*, *arg0*: *PEAKINGDUCK.core.NumericalData*) → *PEAKINGDUCK.core.NumericalData*

class *peakingduck.core.IProcessManager*

Bases: *pybind11_builtins.pybind11_object*

A general process manager interface

append (*self*: *PEAKINGDUCK.core.IProcessManager*, *arg0*: *PEAKINGDUCK.core.IProcess*) → *PEAKINGDUCK.core.IProcessManager*

reset (*self*: *PEAKINGDUCK.core.IProcessManager*) → *None*

run (*self*: *PEAKINGDUCK.core.IProcessManager*, *arg0*: *PEAKINGDUCK.core.NumericalData*) → *PEAKINGDUCK.core.NumericalData*

class *peakingduck.core.IntegerData*

Bases: *pybind11_builtins.pybind11_object*

Represents a 1-dimensional data structure of ints (basically a 1D Eigen array)

Dynamic array - most use cases will be determined at runtime (I am assuming). We don't want anyone to know we are using Eigen beyond this file, since (in theory) it should make it easier to change library if need be. We only really need the array datastructure from Eigen and not much else and instead of reinventing the wheel, we wrap Eigen array.

We wrap this with private inheritance on the Eigen type but there are a lot of methods to expose, easy to add when/if we need them.

Eigen array is pretty good, it has things like `sqrt`, `exp` on array coefficients, but we need to extend this to other functions, so we use CRTP to do this.

For all of this, you may ask why not just use Eigen and use an alias? Well for one, we don't need all of Eigen just the array, and not all of the array type (we require a simpler interface). Additionally, at some point we may wish to use another data structure as `std::array` for example. In this case we just change the `NumericalData` class to wrap that instead. If we change the alias this could break existing interfaces and APIs, causing big changes later on. Since this datastructure is fundamental to everything we need to make sure that we have this sorted properly first!

from_list (*self*: *PEAKINGDUCK.core.IntegerData*, *arg0*: *List[int]*) → *None*

maxCoeff (*self*: *PEAKINGDUCK.core.IntegerData*) → *int*

minCoeff (*self*: *PEAKINGDUCK.core.IntegerData*) → *int*

ramp (*self*: *PEAKINGDUCK.core.IntegerData*, *arg0*: *int*) → *PEAKINGDUCK.core.IntegerData*

A simple function for filtering values above a certain threshold (\geq). This is useful to remove entries that are negative for example.

Returns A new array.

rampInPlace (*self*: *PEAKINGDUCK.core.IntegerData*, *arg0*: *int*) → *PEAKINGDUCK.core.IntegerData*

A simple function for filtering values above a certain threshold (\geq). This is useful to remove entries that are negative for example.

Mutates underlying data.

reverse (*self*: *PEAKINGDUCK.core.IntegerData*) → *numpy.ndarray[int32[m, 1]]*

reverseInPlace (*self*: *PEAKINGDUCK.core.IntegerData*) → *None*

slice (*self*: *PEAKINGDUCK.core.IntegerData*, *arg0*: *int*, *arg1*: *int*) → *PEAKINGDUCK.core.IntegerData*

sum (*self*: *PEAKINGDUCK.core.IntegerData*) → *int*

`to_list` (*self*: *PEAKINGDUCK.core.IntegerData*) → List[int]

class `peakingduck.core.MovingAveragePeakFilter`

Bases: `PEAKINGDUCK.core.IProcess`

Simple moving average peak filter

class `peakingduck.core.MovingAverageSmoother`

Bases: `PEAKINGDUCK.core.IProcess`

Simple moving average smoother

Can we support window size given at compile time too? It would certainly help with unit tests.

class `peakingduck.core.NumericalData`

Bases: `pybind11_builtins.pybind11_object`

Represents a 1-dimensional data structure of floats (basically a 1D Eigen array)

Dynamic array - most use cases will be determined at runtime (I am assuming). We don't want anyone to know we are using Eigen beyond this file, since (in theory) it should make it easier to change library if need be. We only really need the array datastructure from Eigen and not much else and instead of reinventing the wheel, we wrap Eigen array.

We wrap this with private inheritance on the Eigen type but there are a lot of methods to expose, easy to add when/if we need them.

Eigen array is pretty good, it has things like `sqrt`, `exp` on array coefficients, but we need to extend this to other functions, so we use CRTP to do this.

For all of this, you may ask why not just use Eigen and use an alias? Well for one, we don't need all of Eigen just the array, and not all of the array type (we require a simpler interface). Additionally, at some point we may wish to use another data structure as `std::array` for example. In this case we just change the `NumericalData` class to wrap that instead. If we change the alias this could break existing interfaces and APIs, causing big changes later on. Since this datastructure is fundamental to everything we need to make sure that we have this sorted properly first!

LLS (*self*: *PEAKINGDUCK.core.NumericalData*) → `PEAKINGDUCK.core.NumericalData`

$\log(\log(\sqrt{\text{value} + 1} + 1) + 1)$

Returns A new array.

LLSInPlace (*self*: *PEAKINGDUCK.core.NumericalData*) → `PEAKINGDUCK.core.NumericalData`

$\log(\log(\sqrt{\text{value} + 1} + 1) + 1)$

Changes the underlying array.

exp (*self*: *PEAKINGDUCK.core.NumericalData*) → `numpy.ndarray[float64[m, 1]]`

from_list (*self*: *PEAKINGDUCK.core.NumericalData*, *arg0*: List[float]) → None

inverseLLS (*self*: *PEAKINGDUCK.core.NumericalData*) → `PEAKINGDUCK.core.NumericalData`

$\exp(\exp(\sqrt{\text{value} + 1} + 1) + 1)$

Returns A new array.

inverseLLSInPlace (*self*: *PEAKINGDUCK.core.NumericalData*) → `PEAKINGDUCK.core.NumericalData`

$\exp(\exp(\sqrt{\text{value} + 1} + 1) + 1)$

Changes the underlying array.

log (*self*: *PEAKINGDUCK.core.NumericalData*) → `numpy.ndarray[float64[m, 1]]`

maxCoeff (*self*: *PEAKINGDUCK.core.NumericalData*) → float

mean (*self*: *PEAKINGDUCK.core.NumericalData*) → float

midpoint (*self*: *PEAKINGDUCK.core.NumericalData*, *arg0*: *int*) → *PEAKINGDUCK.core.NumericalData*

For each element calculate the midpoint value from the adjacent elements at a given order.

Take the i-order point and the i+order point and determine the average = (array[i-j]+array[i+j])/2.0. End points are not counted (stay as original) - max(0, i-j) and min(i+j, len(array))

Examples

Given an array: [1, 4, 6, 2, 4, 2, 5]

- we have the midpoints for order 0: [1, 4, 6, 2, 4, 2, 5]
- we have the midpoints for order 1: [1, 3.5, 3, 5, 2, 4.5, 5]
- we have the midpoints for order 2: [1, 4, 2.5, 3, 5.5, 2, 5]
- we have the midpoints for order 3: [1, 4, 6, 3, 4, 2, 5]
- we have the midpoints for order 4+: [1, 4, 6, 2, 4, 2, 5]

Returns A new array.

midpointInPlace (*self*: *PEAKINGDUCK.core.NumericalData*, *arg0*: *int*) → *PEAKINGDUCK.core.NumericalData*

For each element calculate the midpoint value from the adjacent elements at a given order.

Mutates underlying array.

See also:

`peakingduck.core.NumericalData.midpoint()`

minCoeff (*self*: *PEAKINGDUCK.core.NumericalData*) → float

pow (*self*: *PEAKINGDUCK.core.NumericalData*, *arg0*: *float*) → `numpy.ndarray[float64[m, 1]]`

ramp (*self*: *PEAKINGDUCK.core.NumericalData*, *arg0*: *float*) → *PEAKINGDUCK.core.NumericalData*

A simple function for filtering values above a certain threshold (>=). This is useful to remove entries that are negative for example.

Returns A new array.

rampInPlace (*self*: *PEAKINGDUCK.core.NumericalData*, *arg0*: *float*) → *PEAKINGDUCK.core.NumericalData*

A simple function for filtering values above a certain threshold (>=). This is useful to remove entries that are negative for example.

Mutates underlying data.

reverse (*self*: *PEAKINGDUCK.core.NumericalData*) → `numpy.ndarray[float64[m, 1]]`

reverseInPlace (*self*: *PEAKINGDUCK.core.NumericalData*) → None

slice (*self*: *PEAKINGDUCK.core.NumericalData*, *arg0*: *int*, *arg1*: *int*) → *PEAKINGDUCK.core.NumericalData*

snip (**args*, ***kwargs*)

Overloaded function.

1. `snip(self: PEAKINGDUCK.core.NumericalData, arg0: List[int]) -> PEAKINGDUCK.core.NumericalData`

Sensitive Nonlinear Iterative Peak (SNIP) algorithm for estimating backgrounds

ref needed here:

Allows any form of iterations given an iterator

Returns: A new array.

2. `snip(self: PEAKINGDUCK.core.NumericalData, iterations: int = 20) -> PEAKINGDUCK.core.NumericalData`

Sensitive Nonlinear Iterative Peak (SNIP) algorithm for estimating backgrounds ref needed here.

Does via increasing window only (ToDo: need to allow decreasing window)

Deprecate this!

Returns: A new array.

`sqrt (self: PEAKINGDUCK.core.NumericalData) -> numpy.ndarray[float64[m, 1]]`

`square (self: PEAKINGDUCK.core.NumericalData) -> numpy.ndarray[float64[m, 1]]`

`stddev (self: PEAKINGDUCK.core.NumericalData, ddof: int = 0) -> float`

`sum (self: PEAKINGDUCK.core.NumericalData) -> float`

`to_list (self: PEAKINGDUCK.core.NumericalData) -> List[float]`

class `peakingduck.core.PeakInfo`

Bases: `pybind11_builtins.pybind11_object`

Simple struct for holding peak info.

value

value, i.e. count or flux.

index

the corresponding index/channel of the data.

property index

property value

class `peakingduck.core.PySimpleProcessManager (processes=None)`

Bases: `PEAKINGDUCK.core.IProcessManager`

Instead of using the C++ SimpleProcessManager we make a python native version to avoid ref counting issues

Easy to extend also.

`append (self: PEAKINGDUCK.core.IProcessManager, arg0: PEAKINGDUCK.core.IProcess) -> PEAKINGDUCK.core.IProcessManager`

`run (self: PEAKINGDUCK.core.IProcessManager, arg0: PEAKINGDUCK.core.NumericalData) -> PEAKINGDUCK.core.NumericalData`

class `peakingduck.core.SavitzkyGolaySmoother (windowsize, order=2)`

Bases: `PEAKINGDUCK.core.IProcess`

`go (self: PEAKINGDUCK.core.IProcess, arg0: PEAKINGDUCK.core.NumericalData) -> PEAKINGDUCK.core.NumericalData`

class `peakingduck.core.ScipyPeakFinder (threshold=2.0, smoothsize=1001)`

Bases: `PEAKINGDUCK.core.IPeakFinder`

Wrapper for scipy peak finder

TODO: pass smoother to constructor not just window size

find (*self*: *PEAKINGDUCK.core.IPeakFinder*, *arg0*: *PEAKINGDUCK.core.NumericalData*) → List[*PEAKINGDUCK.core.PeakInfo*]
Identifies potential peaks in the data

class *peakingduck.core.SimplePeakFinder*

Bases: *PEAKINGDUCK.core.IPeakFinder*

find (*self*: *PEAKINGDUCK.core.SimplePeakFinder*, *arg0*: *PEAKINGDUCK.core.NumericalData*) → List[*PEAKINGDUCK.core.PeakInfo*]

class *peakingduck.core.SimpleProcessManager*

Bases: *PEAKINGDUCK.core.IProcessManager*

A simple process manager

append (*self*: *PEAKINGDUCK.core.SimpleProcessManager*, *arg0*: *PEAKINGDUCK.core.IProcess*) → *PEAKINGDUCK.core.IProcessManager*

reset (*self*: *PEAKINGDUCK.core.SimpleProcessManager*) → None

run (*self*: *PEAKINGDUCK.core.SimpleProcessManager*, *arg0*: *PEAKINGDUCK.core.NumericalData*) → *PEAKINGDUCK.core.NumericalData*

class *peakingduck.core.SpectrumChannelBased*

Bases: *PEAKINGDUCK.core.HistogramChannelBased*

Represents a basic 1D histogram

Channels vs values.

estimateBackground (*self*: *PEAKINGDUCK.core.SpectrumChannelBased*, *arg0*: List[int]) → *PEAKINGDUCK.core.NumericalData*

removeBackground (*self*: *PEAKINGDUCK.core.SpectrumChannelBased*, *arg0*: List[int]) → None

class *peakingduck.core.SpectrumEnergyBased*

Bases: *PEAKINGDUCK.core.Histogram*

Represents a basic 1D histogram

Energies vs values.

estimateBackground (*self*: *PEAKINGDUCK.core.SpectrumEnergyBased*, *arg0*: List[int]) → *PEAKINGDUCK.core.NumericalData*

removeBackground (*self*: *PEAKINGDUCK.core.SpectrumEnergyBased*, *arg0*: List[int]) → None

class *peakingduck.core.WeightedMovingAverageNative* (*windowsize*)

Bases: *PEAKINGDUCK.core.IProcess*

We can extend the C++ classes here This is an example

go (*data*)

if N=1, weights=[1] -> [1/N] if N=2, weights=[1,1] -> [1/2, 1/2] if N=3, weights=[1,2,1] -> [1/4, 2/4, 1/4]
if N=4, weights=[1,2,2,1] -> [1/6, 2/6, 2/6, 1/6] if N=5, weights=[1,2,3,2,1] -> [1/9, 2/9, 3/9, 2/9, 1/9] ...

weights=[1,2,..., ceil(n/2), ..., 2, 1] = [1, ..., ceil(n/2)] + [ceil(n/2), ..., 1] weights = weights/sum(weights)

class *peakingduck.core.WeightedMovingAverageSmoother*

Bases: *PEAKINGDUCK.core.IProcess*

Simple moving average smoother

Uses weights determined, with *windowsize* = N

- if N=1, weights=[1] -> [1/N]
- if N=2, weights=[1,1] -> [1/2, 1/2]
- if N=3, weights=[1,2,1] -> [1/4, 2/4, 1/4]
- if N=4, weights=[1,2,2,1] -> [1/6, 2/6, 2/6, 1/6]
- if N=5, weights=[1,2,3,2,1] -> [1/9, 2/9, 3/9, 2/9, 1/9]
- ...

```
class peakingduck.core.WindowPeakFinder (threshold=2.0, ninner=0, nouter=40, include_point=False, enforce_maximum=False, use_grad=False)
```

Bases: PEAKINGDUCK.core.IPeakFinder

A bespoke window method peak finder

```
find (data, *args, **kwargs)
```

Takes npoints either side of bin for each bin in histogram to get mean and stddev with and without that bin

Doesn't use the gradient yet, but we should enable this

```
peakingduck.core.combine (arg0: PEAKINGDUCK.core.NumericalData, arg1: PEAKINGDUCK.core.NumericalData) → PEAKINGDUCK.core.NumericalData
```

```
peakingduck.core.np
```

```
peakingduck.core.peakwindow ()
```

```
window(values: PEAKINGDUCK.core.NumericalData, centerindex: int, nouter: int = 5, ninner: int = 0, includeindex: bool = True) -> PEAKINGDUCK.core.NumericalData
```

```
peakingduck.core.scipy
```

```
peakingduck.core.signal
```

```
peakingduck.core.window (values: PEAKINGDUCK.core.NumericalData, centerindex: int, nouter: int = 5, ninner: int = 0, includeindex: bool = True) → PEAKINGDUCK.core.NumericalData
```

3.6.2 peakingduck.io module

```
peakingduck.io.from_csv (arg0: PEAKINGDUCK.core.SpectrumEnergyBased, arg1: str) → None
```

Deserialization method for histogram

Assumes delimited text data in column form of:

```
channel, lowerenergy, upperenergy, count
```

3.6.3 peakingduck.plotting module

```
class peakingduck.plotting.LinePlotAdapter
```

Bases: peakingduck.plotting.plotadaptor.PlotAdapter

```
lineplot (x, y, datalabel="", xlabel="", ylabel="", logx=False, logy=False, overlay=True)
```

```
peakingduck.plotting.PLT
```


class `peakingduck.plotting.PlotAdapter`

Bases: `object`

Wraps the Matplotlib plotter

addlegend (*location*)

property engine

The plotter engine

property enginename

The name of the plotting engine

grid (*show=True*)

newcanvas (**args, **kwargs*)

show ()

`peakingduck.plotting.getplotvalues` (*x, y*)

Matplotlib hist is slow. Use a hist like plot with conventional line plot

`peakingduck.plotting.matplotlib`

3.6.4 peakingduck.util module

`peakingduck.util.get_window` (*values: List[float], centerindex: int, nouter: int = 5, ninner: int = 0, includeindex: bool = True*) → `List[float]`

Given a list of values take nouter points either side of the index given and ignore ninner points.

Examples

```
>>> get_window([8, 2, 5, 2, 6, 6, 9, 23, 12], 4, 3, 0, True)
[2, 5, 2, 6, 6, 9, 23]
>>> get_window([8, 2, 5, 2, 6, 6, 9, 23, 12], 4, 3, 0, False)
[2, 5, 2, 6, 9, 23]
>>> get_window([8, 2, 5, 2, 6, 6, 9, 23, 12], 4, 3, 1, True)
[2, 5, 6, 9, 23]
>>> get_window([8, 2, 5, 2, 6, 6, 9, 23, 12], 4, 3, 1, False)
[2, 5, 9, 23]
```

Therefore:

- `ninner >= 0`
- `ninner <= nouter`
- `index >= nouter`
- `index < values.size()`

It will clip at `(0, len(values))`

3.7 peakingduck Header Only Library

Defines the full library. A single header file for whole library

Copyright UK Atomic Energy Authority (UKAEA) - 2019-20

```
class PeakingDuckDBConnectionInvalidException : public peakingduck::PeakingDuckException
    #include <exceptions.hpp> Database connection exception PeakingDuckDBConnectionInvalidException Throw
    when issues with attempting to connect to the database.
```

Public Functions

```
PeakingDuckDBConnectionInvalidException (std::string text)
virtual ~PeakingDuckDBConnectionInvalidException ()
virtual const char *what () const
```

Protected Attributes

```
std::string _what
```

```
class PeakingDuckException : public std::exception
    #include <exceptions.hpp> Base level exception PeakingDuckException Should be extended with subclasses.
    Subclassed by peakingduck::PeakingDuckDBConnectionInvalidException, peaking-
    duck::PeakingDuckFileFormatReadException, peakingduck::PeakingDuckMapKeyNotFoundException
```

Public Functions

```
PeakingDuckException (std::string text)
virtual ~PeakingDuckException ()
virtual const char *what () const
```

Protected Attributes

```
std::string _what
```

```
class PeakingDuckFileFormatReadException : public peakingduck::PeakingDuckException
    #include <exceptions.hpp> File Format exception PeakingDuckFileFormatReadException Throw when issues
    with file formatting (reading)
```

Public Functions

PeakingDuckFileFormatReadException (std::string *text*)

virtual ~PeakingDuckFileFormatReadException ()

virtual const char *what () **const**

class PeakingDuckMapKeyNotFoundException : **public** peakingduck::PeakingDuckException
#include <exceptions.hpp> Key lookup exception *PeakingDuckMapKeyNotFoundException* Throw when issues when cannot find a key by value (std::map)

Public Functions

PeakingDuckMapKeyNotFoundException (std::string *text*)

virtual ~PeakingDuckMapKeyNotFoundException ()

virtual const char *what () **const**

3.7.1 peakingduck::constants namespace

Defines the constants (mathematical and physical) of the library.

Copyright UK Atomic Energy Authority (UKAEA) - 2019-20

constexpr const double peakingduck::constants::PI = units::constants::pi

3.7.2 peakingduck::core namespace

Defines all core parts of the library (algorithms, data structures).

Copyright UK Atomic Energy Authority (UKAEA) - 2019-20

using peakingduck::core::Array1D = Eigen::Array<Scalar, Size, 1>

using peakingduck::core::Array1Di = Array1D<int>

using peakingduck::core::Array1Df = Array1D<float>

using peakingduck::core::Array1Dd = Array1D<double>

using peakingduck::core::DefaultType = double

using peakingduck::core::PeakList = std::vector<PeakInfo<ValueType>>

const int peakingduck::core::ArrayTypeDynamic = Eigen::Dynamic

template<typename **T** = *DefaultType*, int **Size** = *ArrayTypeDynamic*>

NumericalData<**T**, **Size**> peakingduck::core::combine (**const** NumericalData<**T**, *ArrayTypeDynamic*> &one, **const** NumericalData<**T**, *ArrayTypeDynamic*> &two)

Combine (concatenate) arrays into another.

template<typename **T** = *DefaultType*, int **InputSize** = *ArrayTypeDynamic*, int **WindowSize** = *ArrayTypeDynamic*>

```
NumericalData<T, WindowSize> peakingduck::core::window (const NumericalData<T, InputSize>
&data, int centerindex, int nouter =
5, int ninner = 0, bool includeindex =
true)
```

Given a list of values take nouter points either side of the index given and ignore ninner points.

Examples:

1. values = [8, 2, 5, 2, 6, 6, 9, 23, 12] index = 4 nouter = 3 ninner = 0 includeindex = True
=> [2, 5, 2, 6, 6, 9, 23]
2. values = [8, 2, 5, 2, 6, 6, 9, 23, 12] index = 4 nouter = 3 ninner = 0 includeindex = False
=> [2, 5, 2, 6, 9, 23]
3. values = [8, 2, 5, 2, 6, 6, 9, 23, 12] index = 4 nouter = 3 ninner = 1 includeindex = True
=> [2, 5, 6, 9, 23]
4. values = [8, 2, 5, 2, 6, 6, 9, 23, 12] index = 4 nouter = 3 ninner = 1 includeindex = False
=> [2, 5, 9, 23]

Therefore:

- ninner >= 0
- ninner <= nouter
- index >= nouter
- index < values.size()

It will clip at (0, len(values))

```
template<typename T = DefaultType, int Size = ArrayTypeDynamic>
struct ChunkedThresholdPeakFilter : public peakingduck::core::IProcess<T, Size>
    #include <peaking.hpp> Simple threshold local/chunked peak filter.
```

Public Functions

```
ChunkedThresholdPeakFilter (T percentThreshold, size_t chunkSize = 10)
```

```
NumericalData<T, Size> go (const NumericalData<T, Size> &data) const
```

```
template<typename T, template<typename> class crtpType>
```

```
class crtp
```

```
#include <crtp.hpp> Represents the base CRTP class to allow objects to have certain abilities in their interface.
```

Public Functions

```
T &underlying ()
```

```
T const &underlying () const
```

```
template<typename T = DefaultType, int Size = ArrayTypeDynamic>
```

```
struct GlobalThresholdPeakFilter : public peakingduck::core::IProcess<T, Size>
```

```
#include <peaking.hpp> Simple threshold global peak filter.
```

Public Functions

GlobalThresholdPeakFilter (T *percentThreshold*)

NumericalData<T, Size> **go** (const NumericalData<T, Size> &*data*) **const**

template<typename **XScalar**, typename **YScalar**>

class Histogram

#include <spectral.hpp> Represents a basic 1D histogram Energies vs values or Channel vs values.

Subclassed by *peakingduck::core::Spectrum< XScalar, YScalar >*

Public Functions

Histogram ()

Histogram (const NumericalData<XScalar> &*X*, const NumericalData<YScalar> &*Y*)

Histogram (const *Histogram* &*other*)

Histogram (*Histogram* &&*other*)

Histogram &**operator=** (const *Histogram* &*other*)

Histogram &**operator=** (*Histogram* &&*other*)

virtual ~Histogram ()

NumericalData<XScalar> **X** () **const**

NumericalData<YScalar> **Y** () **const**

Protected Attributes

NumericalData<XScalar> **_X**

NumericalData<YScalar> **_Y**

template<typename **ValueType** = *DefaultType*, int **Size** = *ArrayTypeDynamic*>

struct IPeakFinder

#include <peaking.hpp> Interface for peak finding algorithms.

Operates on numerical data (filtered or unfiltered) Never mutates the input (always const process) returns a list of peaks - PeakList

Subclassed by *peakingduck::core::SimplePeakFinder< ValueType, Size >*

Public Functions

virtual ~IPeakFinder ()

virtual PeakList<ValueType> **find** (const NumericalData<ValueType, Size> &*data*) **const** = 0
Identifies potential peaks in the data.

template<typename **T** = *DefaultType*, int **Size** = *ArrayTypeDynamic*>

struct IProcess

#include <process.hpp> Interface for all process algorithms.

Operates on numerical data Never mutates the input (always const process) returns a new numerical array

Subclassed by *peakingduck::core::ChunkedThresholdPeakFilter< T, Size >*, *peakingduck::core::GlobalThresholdPeakFilter< T, Size >*, *peakingduck::core::MovingAveragePeakFilter< T, Size >*, *peakingduck::core::MovingAverageSmoother< T, Size >*, *peakingduck::core::WeightedMovingAverageSmoother< T, Size >*

Public Functions

virtual ~IProcess ()

virtual NumericalData<T, Size> go (const NumericalData<T, Size> &data) const = 0

template<typename T = *DefaultType*, int Size = *ArrayTypeDynamic*>

struct IProcessManager

#include <process.hpp> A general process manager interface.

Subclassed by *peakingduck::core::SimpleProcessManager< T, Size >*

Public Functions

virtual ~IProcessManager ()

virtual IProcessManager &append (const std::shared_ptr<IProcess<T, Size>> &process) = 0

virtual NumericalData<T, Size> run (const NumericalData<T, Size> &data) const = 0

virtual size_t size () const = 0

virtual void reset () = 0

template<typename T = *DefaultType*, int Size = *ArrayTypeDynamic*>

struct MovingAveragePeakFilter : public peakingduck::core::IProcess<T, Size>

#include <peaking.hpp> Simple moving average peak filter.

Public Functions

MovingAveragePeakFilter (int *windowSize*)

NumericalData<T, Size> go (const NumericalData<T, Size> &data) const

template<typename T = *DefaultType*, int Size = *ArrayTypeDynamic*>

struct MovingAverageSmoother : public peakingduck::core::IProcess<T, Size>

#include <smoothing.hpp> Simple moving average smoother.

Can we support windowSize given at compile time too? It would certainly help with unit tests.

Public Functions

MovingAverageSmoother (int *windowSize*)

NumericalData<T, Size> **go** (const NumericalData<T, Size> &*data*) **const**

template<typename **T** = *DefaultType*, int **Size** = *ArrayTypeDynamic*>

struct NumericalData : private *Array1D*<T, Size>, public peakingduck::core::NumericalFunctions<*NumericalData*<T, S
#include <*numerical.hpp*> Represents a 1-dimensional data structure (basically a 1D Eigen array) Dynamic array - most use cases will be determined at runtime (I am assuming). We don't want anyone to know we are using Eigen beyond this file, since (in theory) it should make it easier to change library if need be. We only really need the array datastructure from Eigen and not much else and instead of reinventing the wheel, we wrap Eigen array.

We wrap this with private inheritance on the Eigen type but there are a lot of methods to expose, easy to add when/if we need them.

Eigen array is pretty good, it has things like sqrt, exp on array coefficients, but we need to extend this to other functions, so we use CRTP to do this.

For all of this, you may ask why not just use Eigen and use an alias? Well for one, we don't need all of Eigen just the array, and not all of the array type (we require a simpler interface). Additionally, at some point we may wish to use another data structure as std::array for example. In this case we just change the *NumericalData* class to wrap that instead. If we change the alias this could break existing interfaces and APIs, causing big changes later on. Since this datastructure is fundamental to everything we need to make sure that we have this sorted properly first!

Public Types

template<>

using value_type = T

template<>

using BaseEigenArray = *Array1D*<*value_type*, Size>

Public Functions

template<typename **OtherDerived**>

NumericalData (const Eigen::ArrayBase<*OtherDerived*> &*other*)

NumericalData ()

NumericalData (const std::vector<*value_type*> &*other*)

template<typename **OtherDerived**>

NumericalData &**operator=** (const Eigen::ArrayBase<*OtherDerived*> &*other*)

template<typename **OtherDerived**>

NumericalData &**operator=** (const Eigen::EigenBase<*OtherDerived*> &*other*)

template<typename **OtherDerived**>

NumericalData &**operator=** (const Eigen::ReturnByValue<*OtherDerived*> &*other*)

NumericalData &**operator=** (const std::vector<T> &*other*)

NumericalData **operator+** (const T &*scalar*) **const**

```
NumericalData operator+ (const NumericalData &rhs) const  
const NumericalData &operator+= (const NumericalData &rhs)  
NumericalData operator- (const T &scalar) const  
NumericalData operator- (const NumericalData &rhs) const  
const NumericalData &operator-= (const NumericalData &rhs)  
NumericalData operator* (const T &scalar) const  
NumericalData operator* (const NumericalData &rhs) const  
const NumericalData &operator*= (const NumericalData &rhs)  
NumericalData operator/ (const T &scalar) const  
NumericalData operator/ (const NumericalData &rhs) const  
const NumericalData &operator/= (const NumericalData &rhs)  
void from_vector (const std::vector<value_type> &raw)  
std::vector<value_type> to_vector () const  
NumericalData<value_type> slice (int sindex, int eindex) const  
NumericalData<value_type> operator () (int sindex, int eindex) const
```

```
NumericalData ramp (const value_type &threshold) const
```

A simple function for filtering values above a certain threshold (\geq). This is useful to remove entries that are negative for example.

Returns a new array

```
NumericalData &rampInPlace (const value_type &threshold)
```

A simple function for filtering values above a certain threshold (\geq). This is useful to remove entries that are negative for example.

Mutates underlying array

Friends

```
NumericalData operator+ (const T &scalar, const NumericalData &rhs)
```

```
NumericalData operator- (const T &scalar, const NumericalData &rhs)
```

```
NumericalData operator* (const T &scalar, const NumericalData &rhs)
```

```
NumericalData operator/ (const T &scalar, const NumericalData &rhs)
```

```
template<class Derived>
```

```
struct NumericalFunctions : public peakingduck::core::crtp<Derived, NumericalFunctions>
```

```
    #include <numericalfunctions.hpp> To extend the NumericalData type with certain numerical abilities, we add it to this using CRTP to keep it in the interface but it doesn't require us to change the underlying data structure.
```


Public Functions

decltype(auto) **stddev** (int *ddof* = 0) **const**

Derived **LLS** () **const**

$\log(\log(\sqrt{\text{value} + 1} + 1) + 1)$ Returns a new array

Derived **&LLSInPlace** ()

$\log(\log(\sqrt{\text{value} + 1} + 1) + 1)$ Changes the underlying array

Derived **inverseLLS** () **const**

$\exp(\exp(\sqrt{\text{value} + 1} + 1) + 1)$ Returns a new array

Derived **&inverseLLSInPlace** ()

$\exp(\exp(\sqrt{\text{value} + 1} + 1) + 1)$ Changes the underlying array

Derived **symmetricNeighbourOp** (**const** std::function<void> int, int, **const** Derived&, Derived&

> &operation, int *order* = 1 **const**) For each element calculate a new value from the symmetric neighbour values at a given order. Take the *i*-order point and the *i*+order point and apply a function to that value End points are not counted (stay as original) - $\max(0, i-j)$ and $\min(i+j, \text{len}(\text{array}))$

Returns a new array

Derived **gradient** (int *order* = 1) **const**

For each element calculate the numerical gradient value from the adjacent elements at a given order. Take the *i*-1 point and the *i*+1 point and determine the $\text{grad} = (\text{array}[i+1] - \text{array}[i-1]) / 2.0$. End points are handled differently as first point $\text{grad} = \text{array}[1] - \text{array}[0]$ End points are handled differently as last point $\text{grad} = \text{array}[-1] - \text{array}[-2]$.

For example, given an array: [1. , 2.0, 4.0, 7.0, 11.0, 16.0] The 1-st order gradient would be: [1. , 1.5, 2.5, 3.5, 4.5, 5.] The 2-nd order gradient would be: [0.5, 0.75, 1.0, 1.0, 0.75, 0.5] The 3-rd order gradient would be: [0.25, 0.25, 0.125, -0.125, -0.25, -0.25]

Returns a new array

Derived **&gradientInPlace** (int *order* = 1)

Computes the numerical gradient in place.

See: [NumericalFunctions::gradient](#)

Mutates underlying array

Derived **midpoint** (int *order* = 1) **const**

For each element calculate the midpoint value from the adjacent elements at a given order. Take the *i*-order point and the *i*+order point and determine the $\text{average} = (\text{array}[i-j] + \text{array}[i+j]) / 2.0$. End points are not counted (stay as original) - $\max(0, i-j)$ and $\min(i+j, \text{len}(\text{array}))$

For example, given an array: [1, 4, 6, 2, 4, 2, 5] we have the midpoints for order 0: [1, 4, 6, 2, 4, 2, 5] we have the midpoints for order 1: [1, 3.5, 3, 5, 2, 4.5, 5] we have the midpoints for order 2: [1, 4, 2.5, 3, 5.5, 2, 5] we have the midpoints for order 3: [1, 4, 6, 3, 4, 2, 5] we have the midpoints for order 4+: [1, 4, 6, 2, 4, 2, 5]

Returns a new array

Derived **&midpointInPlace** (int *order* = 1)

For each element calculate the midpoint value from the adjacent elements at a given order.

See: [NumericalFunctions::midpoint](#)

Mutates underlying array

template<class **Iterator**>
Derived **snip** (*Iterator first, Iterator last*) **const**
Sensitive Nonlinear Iterative Peak (SNIP) algorithm for estimating backgrounds ref needed here:
Allows any form of iterations given an iterator
Returns a new array

Derived **snip** (int *niterations*) **const**
Sensitive Nonlinear Iterative Peak (SNIP) algorithm for estimating backgrounds ref needed here:
does via increasing window only (ToDo: need to allow decreasing window)
Deprecate this!
Returns a new array

Derived **&snipInPlace** (int *niterations*)
Sensitive Nonlinear Iterative Peak (SNIP) algorithm for estimating backgrounds.
See: *NumericalFunctions::snip*
Mutates underlying array

template<typename **ValueType** = *DefaultType*>
struct PeakInfo
#include <peaking.hpp> Simple struct for holding peak info.
Stores the: value = value i.e. count, flux index = the corresponding index/channel in the data
Subclassed by *peakingduck::core::PeakInfoWithProp*< *ValueType*, *PropType* >

Public Functions

PeakInfo (size_t *pindex*, *ValueType pvalue*)

Public Members

const size_t **index**
const *ValueType* **value**

template<typename **ValueType** = *DefaultType*, typename **PropType** = *DefaultType*>
struct PeakInfoWithProp : **public** peakingduck::core::PeakInfo<*ValueType*>
#include <peaking.hpp> Struct extends basic *PeakInfo* with a property value.

Stores the: property = property i.e. energy, time value = value i.e. count, flux index = the corresponding index/channel in the data

Public Functions

PeakInfoWithProp (size_t *pindex*, *ValueType pvalue*, *PropType pprop*)

Public Members

const PropType **property**

```
template<typename ValueType = DefaultType, int Size = ArrayTypeDynamic>
struct SimplePeakFinder : public peakingduck::core::IPeakFinder<ValueType, Size>
    #include <peaking.hpp> Interface for peak finding algorithms.
```

Operates on numerical data (filtered or unfiltered) Never mutates the input (always const process) returns a list of peaks - PeakList

Public Functions

SimplePeakFinder (ValueType *percentThreshold*)

virtual ~**SimplePeakFinder** ()

virtual *PeakList*<ValueType> **find** (**const** NumericalData<ValueType, Size> &*data*) **const**
Identifies potential peaks in the data based on a simple global threshold of max coefficient.

```
template<typename T = DefaultType, int Size = ArrayTypeDynamic>
struct SimpleProcessManager : public peakingduck::core::IProcessManager<T, Size>
    #include <process.hpp> A simple process manager.
```

Public Functions

virtual ~**SimpleProcessManager** ()

IProcessManager<T, Size> &**append** (**const** std::shared_ptr<IProcess<T, Size>> &*process*)

NumericalData<T, Size> **run** (**const** NumericalData<T, Size> &*data*) **const**

size_t **size** () **const**

void **reset** ()

```
template<typename XScalar, typename YScalar>
struct Spectrum : public peakingduck::core::Histogram<XScalar, YScalar>
    #include <spectral.hpp> Represents a basic 1D histogram Energies vs values or Channel vs values.
```

Public Functions

```
template<class Iterator>
void removeBackground (Iterator first, Iterator last)
```

```
template<class Iterator>
NumericalData<YScalar> estimateBackground (Iterator first, Iterator last) const
```

```
template<typename T = DefaultType, int Size = ArrayTypeDynamic>
struct WeightedMovingAverageSmoother : public peakingduck::core::IProcess<T, Size>
    #include <smoothing.hpp> Weighted moving average smoother.
```

Uses weights determined, with window size = N if N=1, weights=[1] -> [1/N] if N=2, weights=[1,1] -> [1/2, 1/2] if N=3, weights=[1,2,1] -> [1/4, 2/4, 1/4] if N=4, weights=[1,2,2,1] -> [1/6, 2/6, 2/6, 1/6] if N=5, weights=[1,2,3,2,1] -> [1/9, 2/9, 3/9, 2/9, 1/9] ...

Public Functions

WeightedMovingAverageSmoother (int *windowSize*)

NumericalData<T, Size> **go** (const NumericalData<T, Size> &*data*) **const**

3.7.3 peakingduck::io namespace

Defines all I/O - serialization, etc

Copyright UK Atomic Energy Authority (UKAEA) - 2019-20

constexpr char peakingduck::io::**DEFAULTDELIMITER** = ''

template<typename **XScalar**, typename **YScalar**, char **delimiter**>
static void peakingduck::io::**Deserialize** (std::istream &*stream*, core::Histogram<**XScalar**,
YScalar> &*hist*)

Deserialization method for histogram.

Assumes delimited text data in column form of:

channel,lowerenergy,upperenergy,count

template<typename **XScalar**, typename **YScalar**, char **delimiter** = **DEFAULTDELIMITER**>
std::istream &peakingduck::io::**operator**>> (std::istream &*is*, core::Histogram<**XScalar**, **YScalar**>
&*hist*)

3.7.4 peakingduck::util namespace

Defines all utilities (string manip, file IO, etc) of the library.

Copyright UK Atomic Energy Authority (UKAEA) - 2019-20

bool peakingduck::util::**file_exists_quick** (const std::string &*filename*)
uses POSIX stat to check

a function to check if a file exists on disk (fast approach)

Return True if file exists, false otherwise

Parameters

- *filename*: the name of the file.

bool peakingduck::util::**file_exists** (const std::string &*filename*)
uses C file

A function to check if a file exists on disk (not so quick approach, but still fast)

Return True if file exists, false otherwise

Parameters

- *filename*: the name of the file.

std::string peakingduck::util::**read_stream_into_string** (std::istream &*istream*)
Will return a string from a buffered stream.

A function to read a istream as a string

Return A string containing the whole buffer

Exceptions

- `ios::failure`: if cannot read the buffer

Parameters

- `istream`: the istream buffer

```
template<char delimiter, class Container>
static void peakingduck::util::split (std::istream &stream, Container &cont)
```

Split from a stream using single delimiter per line

```
static void peakingduck::util::ltrim (std::string &s)
```

```
static void peakingduck::util::rtrim (std::string &s)
```

```
static void peakingduck::util::ltrim (std::string &s, char delimiter)
```

```
static void peakingduck::util::rtrim (std::string &s, char delimiter)
```

```
static void peakingduck::util::trim (std::string &s)
```

```
template<typename T>
```

```
static std::vector<T> peakingduck::util::get_window (const std::vector<T> &values, int centerindex, int nouter = 5, int ninner = 0,
                                                    bool includeindex = true)
```

Given a list of values take nouter points either side of the index given and ignore ninner points.

Examples:

1. values = [8, 2, 5, 2, 6, 6, 9, 23, 12] index = 4 nouter = 3 ninner = 0 includeindex = True
=> [2, 5, 2, 6, 6, 9, 23]
2. values = [8, 2, 5, 2, 6, 6, 9, 23, 12] index = 4 nouter = 3 ninner = 0 includeindex = False
=> [2, 5, 2, 6, 9, 23]
3. values = [8, 2, 5, 2, 6, 6, 9, 23, 12] index = 4 nouter = 3 ninner = 1 includeindex = True
=> [2, 5, 6, 9, 23]
4. values = [8, 2, 5, 2, 6, 6, 9, 23, 12] index = 4 nouter = 3 ninner = 1 includeindex = False
=> [2, 5, 9, 23]

Therefore:

- `ninner >= 0`
- `ninner <= nouter`
- `index >= nouter`
- `index < values.size()`

It will clip at (0, len(values))

```
template<typename IntegerType, IntegerType ibegin, IntegerType iend, IntegerType step = 1>
struct range
```

`#include <range.hpp>` A simple range based struct. Assumes begin, end and step known at compile time
Only used for trivial loops to save doing things like: `std::vector<int> indices; for(int i=1, i<5; ++i) indices.push_back(i);`.

Usage as:

```
auto rn = range<size_t,1,5,1>();
for (auto it=rn.begin();it!=rn.end();++it)
    std::cout << *it << ", ";

1, 2, 3, 4,
```

Public Functions

iterator **begin** ()

iterator **end** ()

iterator **begin** () **const**

iterator **end** () **const**

struct **iterator**

Public Types

template<>

using **value_type** = IntegerType

template<>

using **size_type** = std::size_t

template<>

using **difference_type** = IntegerType

template<>

using **pointer** = *value_type* *

template<>

using **reference** = *value_type*&

template<>

using **iterator_category** = std::random_access_iterator_tag

Public Functions

template<>

iterator (**IntegerType** v)

template<>

operator **IntegerType** () **const**

template<>

operator **IntegerType**& ()

template<>

IntegerType **operator*** () **const**

template<>

IntegerType **const** ***operator->** () **const**

template<>

```

iterator &operator++ ()

template<>
iterator &operator++ (int)

template<>
bool operator== (iterator const &other) const

template<>
bool operator!= (iterator const &other) const

```

```
template<typename IntegerType>
```

```
struct rrange
```

#include <range.hpp> A simple range based struct. Assumes begin, end and step not known at compile time Only used for trivial loops to save doing things like: `std::vector<int>` indices; `for(int i=1, i<5; ++i) indices.push_back(i);`.

Usage as:

```

auto rn = range<size_t>(1,5,1);
for (auto it=rn.begin(); it!=rn.end(); ++it)
    std::cout << *it << ", ";

1, 2, 3, 4,

```

Public Functions

```
rrange (IntegerType ibegin, IntegerType iend, IntegerType step = 1)
```

```
iterator begin ()
```

```
iterator end ()
```

```
iterator begin () const
```

```
iterator end () const
```

```
struct iterator
```

Public Types

```

template<>
using value_type = IntegerType

template<>
using size_type = std::size_t

template<>
using difference_type = IntegerType

template<>
using pointer = value_type *

template<>
using reference = value_type&

template<>
using iterator_category = std::random_access_iterator_tag

```

Public Functions

```
template<>  
iterator (rrange<value_type> &range, IntegerType v)
```

```
template<>  
operator IntegerType () const
```

```
template<>  
operator IntegerType& ()
```

```
template<>  
IntegerType operator* () const
```

```
template<>  
IntegerType const *operator-> () const
```

```
template<>  
iterator &operator++ ()
```

```
template<>  
iterator &operator++ (int)
```

```
template<>  
bool operator== (iterator const &other) const
```

```
template<>  
bool operator!= (iterator const &other) const
```


INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

peakingduck, 5

PEAKINGDUCK, 5

peakingduck.core, 6

peakingduck.io, 12

peakingduck.plotting, 12

peakingduck.util, 13

A

addlegend() (*peakingduck.plotting.PlotAdapter method*), 13
 append() (*peakingduck.core.IProcessManager method*), 7
 append() (*peakingduck.core.PySimpleProcessManager method*), 10
 append() (*peakingduck.core.SimpleProcessManager method*), 11

C

ChunkedSimplePeakFinder (*class in peakingduck.core*), 6
 ChunkedThresholdPeakFilter (*class in peakingduck.core*), 6
 combine() (*in module peakingduck.core*), 12

E

engine() (*peakingduck.plotting.PlotAdapter property*), 13
 enginename() (*peakingduck.plotting.PlotAdapter property*), 13
 estimateBackground() (*peakingduck.core.SpectrumChannelBased method*), 11
 estimateBackground() (*peakingduck.core.SpectrumEnergyBased method*), 11
 exp() (*peakingduck.core.NumericalData method*), 8

F

find() (*peakingduck.core.ChunkedSimplePeakFinder method*), 6
 find() (*peakingduck.core.IPeakFinder method*), 6
 find() (*peakingduck.core.ScipyPeakFinder method*), 11
 find() (*peakingduck.core.SimplePeakFinder method*), 11
 find() (*peakingduck.core.WindowPeakFinder method*), 12
 from_csv() (*in module peakingduck.io*), 12

from_list() (*peakingduck.core.IntegerData method*), 7
 from_list() (*peakingduck.core.NumericalData method*), 8

G

get_window() (*in module peakingduck.util*), 13
 getplotvalues() (*in module peakingduck.plotting*), 13
 GlobalThresholdPeakFilter (*class in peakingduck.core*), 6
 go() (*peakingduck.core.IProcess method*), 6
 go() (*peakingduck.core.SavitzkyGolaySmoother method*), 10
 go() (*peakingduck.core.WeightedMovingAverageNative method*), 11
 grid() (*peakingduck.plotting.PlotAdapter method*), 13

H

Histogram (*class in peakingduck.core*), 6
 HistogramChannelBased (*class in peakingduck.core*), 6

I

index (*peakingduck.core.PeakInfo attribute*), 10
 index() (*peakingduck.core.PeakInfo property*), 10
 IntegerData (*class in peakingduck.core*), 7
 inverseLLS() (*peakingduck.core.NumericalData method*), 8
 inverseLLSInPlace() (*peakingduck.core.NumericalData method*), 8
 IPeakFinder (*class in peakingduck.core*), 6
 IProcess (*class in peakingduck.core*), 6
 IProcessManager (*class in peakingduck.core*), 7

L

lineplot() (*peakingduck.plotting.LinePlotAdapter method*), 12
 LinePlotAdapter (*class in peakingduck.plotting*), 12
 LLS() (*peakingduck.core.NumericalData method*), 8
 LLSInPlace() (*peakingduck.core.NumericalData method*), 8

log() (*peakingduck.core.NumericalData method*), 8

M

matplotlib (*in module peakingduck.plotting*), 13

maxCoeff() (*peakingduck.core.IntegerData method*), 7

maxCoeff() (*peakingduck.core.NumericalData method*), 8

mean() (*peakingduck.core.NumericalData method*), 8

midpoint() (*peakingduck.core.NumericalData method*), 9

midpointInPlace() (*peakingduck.core.NumericalData method*), 9

minCoeff() (*peakingduck.core.IntegerData method*), 7

minCoeff() (*peakingduck.core.NumericalData method*), 9

MovingAveragePeakFilter (*class in peakingduck.core*), 8

MovingAverageSmoother (*class in peakingduck.core*), 8

N

newcanvas() (*peakingduck.plotting.PlotAdapter method*), 13

np (*in module peakingduck.core*), 12

NumericalData (*class in peakingduck.core*), 8

O

operator* (*C++ function*), 20

operator+ (*C++ function*), 20

operator/ (*C++ function*), 20

operator- (*C++ function*), 20

P

PeakInfo (*class in peakingduck.core*), 10

PEAKINGDUCK (*module*), 5

peakingduck (*module*), 5

peakingduck.core (*module*), 6

peakingduck.io (*module*), 12

peakingduck.plotting (*module*), 12

peakingduck.util (*module*), 13

peakingduck::constants::PI (*C++ member*), 15

peakingduck::core::Array1D (*C++ type*), 15

peakingduck::core::Array1Dd (*C++ type*), 15

peakingduck::core::Array1Df (*C++ type*), 15

peakingduck::core::Array1Di (*C++ type*), 15

peakingduck::core::ArrayTypeDynamic (*C++ member*), 15

peakingduck::core::ChunkedThresholdPeakFilter (*C++ class*), 16

peakingduck::core::ChunkedThresholdPeakFilter::ChunkedThresholdPeakFilter (*C++ function*), 16

peakingduck::core::ChunkedThresholdPeakFilter::go (*C++ function*), 16

peakingduck::core::combine (*C++ function*), 15

peakingduck::core::crtp (*C++ class*), 16

peakingduck::core::crtp::underlying (*C++ function*), 16

peakingduck::core::DefaultType (*C++ type*), 15

peakingduck::core::GlobalThresholdPeakFilter (*C++ class*), 16

peakingduck::core::GlobalThresholdPeakFilter::GlobalThresholdPeakFilter (*C++ function*), 17

peakingduck::core::GlobalThresholdPeakFilter::go (*C++ function*), 17

peakingduck::core::Histogram (*C++ class*), 17

peakingduck::core::Histogram::_X (*C++ member*), 17

peakingduck::core::Histogram::_Y (*C++ member*), 17

peakingduck::core::Histogram::~Histogram (*C++ function*), 17

peakingduck::core::Histogram::Histogram (*C++ function*), 17

peakingduck::core::Histogram::operator= (*C++ function*), 17

peakingduck::core::Histogram::X (*C++ function*), 17

peakingduck::core::Histogram::Y (*C++ function*), 17

peakingduck::core::IPeakFinder (*C++ class*), 17

peakingduck::core::IPeakFinder::~IPeakFinder (*C++ function*), 17

peakingduck::core::IPeakFinder::find (*C++ function*), 17

peakingduck::core::IProcess (*C++ class*), 17

peakingduck::core::IProcess::~IProcess (*C++ function*), 18

peakingduck::core::IProcess::go (*C++ function*), 18

peakingduck::core::IProcessManager (*C++ class*), 18

peakingduck::core::IProcessManager::~IProcessManager (*C++ function*), 18

peakingduck::core::IProcessManager::append (*C++ function*), 18

peakingduck::core::IProcessManager::reset (*C++ function*), 18

peakingduck::core::IProcessManager::run (*C++ function*), 18

peakingduck::core::IProcessManager::size (*C++ function*), 18

peakingduck::core::MovingAveragePeakFilter (C++ class), 18

peakingduck::core::MovingAveragePeakFilter (C++ function), 18

peakingduck::core::MovingAveragePeakFilter (C++ function), 18

peakingduck::core::MovingAverageSmoother (C++ class), 18

peakingduck::core::MovingAverageSmoother (C++ function), 19

peakingduck::core::MovingAverageSmoother (C++ function), 19

peakingduck::core::NumericalData (C++ class), 19

peakingduck::core::NumericalData::from_vector (C++ function), 20

peakingduck::core::NumericalData::NumericalData (C++ function), 19

peakingduck::core::NumericalData::operator (C++ function), 20

peakingduck::core::NumericalData::operator (C++ function), 20

peakingduck::core::NumericalData::operator (C++ function), 20

peakingduck::core::NumericalData::operator+ (C++ function), 19

peakingduck::core::NumericalData::operator+= (C++ function), 20

peakingduck::core::NumericalData::operator/ (C++ function), 20

peakingduck::core::NumericalData::operator/= (C++ function), 20

peakingduck::core::NumericalData::operator= (C++ function), 19

peakingduck::core::NumericalData::operator- (C++ function), 20

peakingduck::core::NumericalData::operator (C++ function), 20

peakingduck::core::NumericalData::ramp (C++ function), 20

peakingduck::core::NumericalData::rampInPlace (C++ function), 20

peakingduck::core::NumericalData::slice (C++ function), 20

peakingduck::core::NumericalData::to_vector (C++ function), 20

peakingduck::core::NumericalData<T, Size>::BaseEigenArray (C++ type), 19

peakingduck::core::NumericalData<T, Size>::value_type (C++ type), 19

peakingduck::core::NumericalFunctions (C++ class), 20

peakingduck::core::NumericalFunctions::gradientInPlace (C++ function), 21

peakingduck::core::NumericalFunctions::inverseLLS (C++ function), 21

peakingduck::core::NumericalFunctions::inverseLLSInPlace (C++ function), 21

peakingduck::core::NumericalFunctions::LLS (C++ function), 21

peakingduck::core::NumericalFunctions::LLSInPlace (C++ function), 21

peakingduck::core::NumericalFunctions::midpoint (C++ function), 21

peakingduck::core::NumericalFunctions::midpointInPlace (C++ function), 21

peakingduck::core::NumericalFunctions::snip (C++ function), 21, 22

peakingduck::core::NumericalFunctions::snipInPlace (C++ function), 22

peakingduck::core::NumericalFunctions::stddev (C++ function), 21

peakingduck::core::NumericalFunctions::symmetricNoise (C++ function), 21

peakingduck::core::PeakInfo (C++ class), 22

peakingduck::core::PeakInfo::index (C++ member), 22

peakingduck::core::PeakInfo::PeakInfo (C++ function), 22

peakingduck::core::PeakInfo::value (C++ member), 22

peakingduck::core::PeakInfoWithProp (C++ class), 22

peakingduck::core::PeakInfoWithProp::PeakInfoWithProp (C++ function), 22

peakingduck::core::PeakInfoWithProp::property (C++ member), 23

peakingduck::core::PeakList (C++ type), 15

peakingduck::core::SimplePeakFinder (C++ class), 23

peakingduck::core::SimplePeakFinder::~~SimplePeakFinder (C++ function), 23

peakingduck::core::SimplePeakFinder::find (C++ function), 23

peakingduck::core::SimplePeakFinder::SimplePeakFinder (C++ function), 23

peakingduck::core::SimpleProcessManager (C++ class), 23

peakingduck::core::SimpleProcessManager::~~SimpleProcessManager (C++ function), 23

peakingduck::core::SimpleProcessManager::append (C++ function), 23

peakingduck::core::SimpleProcessManager::reset (C++ function), 23

peakingduck::core::SimpleProcessManager::run (C++ function), 23

peakingduck::core::SimpleProcessManager::peakingduck::PeakingDuckMapKeyNotFoundException::what (C++ function), 23 (C++ function), 15

peakingduck::core::Spectrum (C++ class), 23 peakingduck::util::file_exists (C++ function), 24

peakingduck::core::Spectrum::estimateBackground (C++ function), 24 peakingduck::util::file_exists_quick (C++ function), 23

peakingduck::core::Spectrum::removeBackground (C++ function), 24 peakingduck::util::get_window (C++ function), 25

peakingduck::core::WeightedMovingAverageSmoothing (C++ class), 23 peakingduck::util::ltrim (C++ function), 25

peakingduck::core::WeightedMovingAverageSmoothing::what (C++ function), 24 peakingduck::util::range (C++ class), 25

peakingduck::core::WeightedMovingAverageSmoothing::what (C++ function), 24 peakingduck::util::range::begin (C++ function), 26

peakingduck::core::window (C++ function), 15 peakingduck::util::range::end (C++ function), 26

peakingduck::io::DEFAULTDELIMITER (C++ member), 24 peakingduck::util::range::iterator (C++ class), 26

peakingduck::io::Deserialize (C++ function), 24 peakingduck::util::range<IntegerType, ibegin, iend, step>::iterator::difference_type (C++ type), 26

peakingduck::io::operator>> (C++ function), 24 peakingduck::util::range<IntegerType, ibegin, iend, step>::iterator::iterator (C++ function), 26

peakingduck::PeakingDuckDBConnectionInvalidException (C++ class), 14 peakingduck::util::PeakingDuckDBConnectionInvalidException (C++ type), 26

peakingduck::PeakingDuckDBConnectionInvalidException::what (C++ member), 14 peakingduck::util::PeakingDuckDBConnectorJoinInvalidException (C++ type), 26

peakingduck::PeakingDuckDBConnectionInvalidException::what (C++ function), 14 peakingduck::util::what::range<IntegerType, ibegin, iend, step>::iterator::operator IntegerType (C++ function), 26

peakingduck::PeakingDuckDBConnectionInvalidException::what (C++ function), 14 peakingduck::util::range<IntegerType, ibegin, iend, step>::iterator::operator IntegerType& (C++ function), 26

peakingduck::PeakingDuckDBConnectionInvalidException::~~PeakingDuckException (C++ member), 14 peakingduck::util::range<IntegerType, ibegin, iend, step>::iterator::operator!= (C++ function), 27

peakingduck::PeakingDuckDBConnectionInvalidException::~~PeakingDuckException (C++ function), 14 peakingduck::util::range<IntegerType, ibegin, iend, step>::PeakingDuckFileFormatReadException (C++ type), 26

peakingduck::PeakingDuckDBConnectionInvalidException::PeakingDuckException (C++ function), 14 peakingduck::PeakingDuckFileFormatReadException (C++ class), 15

peakingduck::PeakingDuckDBConnectionInvalidException::what (C++ function), 14 peakingduck::PeakingDuckFileFormatReadException::step (C++ function), 15

peakingduck::PeakingDuckDBConnectionInvalidException::what (C++ function), 14 peakingduck::PeakingDuckFileFormatReadException::PeakingDuckFileFormatReadException (C++ function), 15

peakingduck::PeakingDuckDBConnectionInvalidException::what (C++ function), 14 peakingduck::PeakingDuckFileFormatReadException::step::operator++ (C++ function), 26, 27

peakingduck::PeakingDuckException (C++ class), 14 peakingduck::util::range<IntegerType, ibegin, iend, step>::PeakingDuckMapKeyNotFoundException (C++ function), 27

peakingduck::PeakingDuckException::_what (C++ member), 14 peakingduck::PeakingDuckMapKeyNotFoundException (C++ class), 15

peakingduck::PeakingDuckException::~~PeakingDuckException (C++ function), 14 peakingduck::PeakingDuckMapKeyNotFoundException::step (C++ function), 15

peakingduck::PeakingDuckException::PeakingDuckException (C++ function), 14 peakingduck::PeakingDuckMapKeyNotFoundException::PeakingDuckMapKeyNotFoundException (C++ function), 15

peakingduck::PeakingDuckException::what (C++ function), 14 peakingduck::PeakingDuckMapKeyNotFoundException::PeakingDuckMapKeyNotFoundException (C++ function), 15

`step>::iterator::operator->`
 (C++ function), 26
`peakingduck::util::range<IntegerType,`
`ibegin, iend,`
`step>::iterator::pointer` (C++
 type), 26
`peakingduck::util::range<IntegerType,`
`ibegin, iend,`
`step>::iterator::reference` (C++
 type), 26
`peakingduck::util::range<IntegerType,`
`ibegin, iend,`
`step>::iterator::size_type` (C++
 type), 26
`peakingduck::util::range<IntegerType,`
`ibegin, iend,`
`step>::iterator::value_type`
 (C++ type), 26
`peakingduck::util::read_stream_into_string`
 (C++ function), 24
`peakingduck::util::rrange` (C++ class), 27
`peakingduck::util::rrange::begin` (C++
 function), 27
`peakingduck::util::rrange::end` (C++ func-
 tion), 27
`peakingduck::util::rrange::iterator`
 (C++ class), 27
`peakingduck::util::rrange::rrange` (C++
 function), 27
`peakingduck::util::rrange<IntegerType>::iterator`
 (C++ type), 27
`peakingduck::util::rrange<IntegerType>::iterator::difference_type`
 (C++ type), 28
`peakingduck::util::rrange<IntegerType>::iterator::iterator`
 (C++ type), 28
`peakingduck::util::rrange<IntegerType>::iterator::operator`
 (C++ type), 27
`peakingduck::util::rrange<IntegerType>::iterator::operator`
`IntegerType` (C++ function), 28
`peakingduck::util::rrange<IntegerType>::iterator::operator`
`IntegerType&` (C++ function), 28
`peakingduck::util::rrange<IntegerType>::iterator::operator!=`
 (C++ function), 28
`peakingduck::util::rrange<IntegerType>::iterator::operator*`
 (C++ function), 28
`peakingduck::util::rrange<IntegerType>::iterator::operator++`
 (C++ function), 28
`peakingduck::util::rrange<IntegerType>::iterator::operator==`
 (C++ function), 28
`peakingduck::util::rrange<IntegerType>::iterator::operator->`
 (C++ function), 28
`peakingduck::util::rrange<IntegerType>::iterator::pointer`
 (C++ type), 27
`peakingduck::util::rrange<IntegerType>::iterator::reference`
 (C++ type), 27
`peakingduck::util::rrange<IntegerType>::iterator::size_type`
 (C++ type), 27
`peakingduck::util::rrange<IntegerType>::iterator::value_type`
 (C++ type), 27
`peakingduck::util::rrange<IntegerType>::iterator::value_type`
 (C++ type), 27
`peakingduck::util::rtrim` (C++ function), 25
`peakingduck::util::split` (C++ function), 25
`peakingduck::util::trim` (C++ function), 25
`peakwindow()` (in module `peakingduck.core`), 12
`PlotAdapter` (class in `peakingduck.plotting`), 12
`PLT` (in module `peakingduck.plotting`), 12
`pow()` (`peakingduck.core.NumericalData` method), 9
`PySimpleProcessManager` (class in `peaking-`
`duck.core`), 10

R

`ramp()` (`peakingduck.core.IntegerData` method), 7
`ramp()` (`peakingduck.core.NumericalData` method), 9
`rampInPlace()` (`peakingduck.core.IntegerData`
 method), 7
`rampInPlace()` (`peakingduck.core.NumericalData`
 method), 9
`removeBackground()` (`peaking-`
`duck.core.SpectrumChannelBased` method),
 11
`removeBackground()` (`peaking-`
`duck.core.SpectrumEnergyBased` method),
 11
`reset()` (`peakingduck.core.IProcessManager` method),
 7
`reset()` (`peakingduck.core.SimpleProcessManager`
 method), 7
`reverse()` (`peakingduck.core.IntegerData` method), 7
`reverse()` (`peakingduck.core.NumericalData`
 method), 9
`reverseInPlace()` (`peakingduck.core.IntegerData`
 method), 7
`reverseInPlace()` (`peakingduck.core.NumericalData`
 method), 9
`run()` (`peakingduck.core.IProcessManager` method), 7
`run()` (`peakingduck.core.PySimpleProcessManager`
 method), 9
`run()` (`peakingduck.core.SimpleProcessManager`
 method), 9

S

`SavitzkyGolaySmoother` (class in `peaking-`
`duck.core`), 10
`scipy` (in module `peakingduck.core`), 12
`ScipyPeakFinder` (class in `peakingduck.core`), 10
`show()` (`peakingduck.plotting.PlotAdapter` method), 13
`signal` (in module `peakingduck.core`), 12
`SimplePeakFinder` (class in `peakingduck.core`), 11
`SimpleProcessManager` (class in `peaking-`
`duck.core`), 11
`slice()` (`peakingduck.core.IntegerData` method), 7

`slice()` (*peakingduck.core.NumericalData* method), 9
`snip()` (*peakingduck.core.NumericalData* method), 9
`SpectrumChannelBased` (class in *peakingduck.core*), 11
`SpectrumEnergyBased` (class in *peakingduck.core*), 11
`sqrt()` (*peakingduck.core.NumericalData* method), 10
`square()` (*peakingduck.core.NumericalData* method), 10
`stddev()` (*peakingduck.core.NumericalData* method), 10
`sum()` (*peakingduck.core.IntegerData* method), 7
`sum()` (*peakingduck.core.NumericalData* method), 10

T

`to_list()` (*peakingduck.core.IntegerData* method), 7
`to_list()` (*peakingduck.core.NumericalData* method), 10

V

`value` (*peakingduck.core.PeakInfo* attribute), 10
`value()` (*peakingduck.core.PeakInfo* property), 10

W

`WeightedMovingAverageNative` (class in *peakingduck.core*), 11
`WeightedMovingAverageSmoother` (class in *peakingduck.core*), 11
`window()` (in module *peakingduck.core*), 12
`WindowPeakFinder` (class in *peakingduck.core*), 12

X

`X()` (*peakingduck.core.Histogram* property), 6
`X()` (*peakingduck.core.HistogramChannelBased* property), 6

Y

`Y()` (*peakingduck.core.Histogram* property), 6
`Y()` (*peakingduck.core.HistogramChannelBased* property), 6