Bayesian Cluster Tool

Generated by Doxygen 1.9.1

Mon May 15 2023 11:47:03

Chapter 1

Todo List

Member Data::CalculateLocalizationScore (const std::vector< Data > &aData, const double &R) const

Remind myself how this works and what the difference is with above

Member Data::PreprocessLocalizationScores (std::vector< Data > &aData)

Remind myself how this works and what the difference is with below

Class PyIterator< U * >

There must be an out-of-the box way, but I can't find it

Class PyIterator< U >

There must be an out-of-the box way, but I can't find it

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

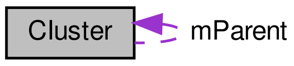
Class Documentation

4.1 Cluster Class Reference

A class representing a cluster.

`#include <Cluster.hpp>`

Collaboration diagram for Cluster:



Classes

- struct Parameter

  A struct representing the cluster parameters.

Public Member Functions

- Cluster ()

  Default constructor.
- Cluster (const Data &aData)

  Construct a cluster from a single data-point.
- Cluster & operator+= (const Cluster &aOther)

  Add another cluster to this one.
- Cluster ∗ GetParent ()

  Get a pointer to this cluster's ultimate parent.
- void UpdateLogScore ()

  Update log-probability after a scan.

Public Attributes

- std::vector< Parameter > mParams

  Get the points after clustering.
- std::size_t mClusterSize

  The number of points in the current cluster.
- std::size_t mLastClusterSize

  The number of points in the cluster on the previous scan iteration.
- PRECISION mClusterScore

  The log-probability of the current cluster.
- Cluster ∗ mParent

  A pointer to the immediate parent of the current cluster.
- std::vector< Data ∗ > mData

  List of points in the cluster after clustering.

4.1.1 Detailed Description

A class representing a cluster.

Definition at line 14 of file Cluster.hpp.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Cluster()

```
Cluster::Cluster (
             const Data & aData )
```

Construct a cluster from a single data-point.

Parameters

aData	A data-point with which to initialize the cluster

Definition at line 102 of file Cluster.cpp.

References Configuration::Instance, mParams, Data::r2, Data::s, Data::x, and Data::y.

4.1.3 Member Function Documentation

4.1.3.1 GetParent()

```
Cluster * Cluster::GetParent ( )
```

Get a pointer to this cluster's ultimate parent.

Returns

A pointer to this cluster's ultimate parent

Definition at line 161 of file Cluster.cpp.

References GetParent(), and mParent.

Referenced by DataProxy::GetCluster(), and GetParent().

4.1.3.2 operator+=()

```
Cluster & Cluster::operator+= (
            const Cluster & aOther )
```

Add another cluster to this one.

Parameters

aOther	Another cluster of parameters to add to this one

Returns

Reference to this, for chaining calls

Definition at line 151 of file Cluster.cpp.

References mClusterSize, and mParams.

4.1.4 Member Data Documentation

4.1.4.1 mParams

```
std::vector< Parameter > Cluster::mParams
```

Get the points after clustering.

Returns

Reference to a list of points in the cluster after clustering The collection of parameters, each corresponding to a different sigma hypothesis

Definition at line 82 of file Cluster.hpp.

Referenced by Cluster(), operator+=(), UpdateLogScore(), and EventProxy::ValidateLogScore().

The documentation for this class was generated from the following files:

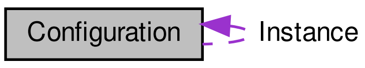
- include/BayesianClustering/Cluster.hpp
- src/BayesianClustering/Cluster.cpp

4.2 Configuration Class Reference

Class for storing the configuration parameters.

```
#include <Configuration.hpp>
```

Collaboration diagram for Configuration:



Public Member Functions

- Configuration ()

  Default constructor.
- void SetCentre (const double &aPhysicalCentreX, const double &aPhysicalCentreY)

  Setter for the centre of the scan window.
- void SetZoom (const double &aScale)

  Setter for the half-width of the scan window.
- void SetSigmaParameters (const std::size_t &aSigmacount, const double &aSigmaMin, const double &a↩ SigmaMax, const std::function< double(const double &) > &aInterpolator)

  Setter for the sigma-bins to be integrated over.
- void SetRBins (const std::size_t &aRbins, const double &aMinScanR=0.0, const double &aMaxScanR=-1)

  Setter for the R bins for the RT scan.
- void SetTBins (const std::size_t &aTbins, const double &aMinScanT=0.0, const double &aMaxScanT=-1)
- void SetPb (const double &aPB)

  Setter for the P_b parameter.
- void SetAlpha (const double &aAlpha)

  Setter for the alpha parameter.
- void SetValidate (const bool &aValidate)

Set whether to validate clusterization.
- void SetInputFile (const std::string &aFileName)

  Setter for the input file.
- void SetOutputFile (const std::string &aFileName)

  Setter for the output file.
- void FromCommandline (int argc, char $**$argv)

  Parse the parameters when passed in as commandline arguments.
- void FromVector (const std::vector$<$ std::string $>$ &aArgs)

  Parse the parameters when passed in as commandline arguments.
- const double & scale2 () const

  Getter for the scale-parameter squared.
- const std::size_t & sigmacount () const

  Getter for the sigma count.
- const double & sigmaspacing () const

  Getter for the sigma spacing.
- const std::vector$<$ double $>$ & sigmabins () const

  Getter for the values of sigma.
- const std::vector$<$ double $>$ & sigmabins2 () const

  Getter for the values of sigma squared.
- const std::vector$<$ double $>$ & probability_sigma () const

  Getter for the probabilities of a given sigma.
- const std::vector$<$ double $>$ & log_probability_sigma () const

  Getter for the log of the probabilities of a given sigma.
- const double & sigmabins (const std::size_t &i) const

  Getter for the i'th value of sigma.
- const double & sigmabins2 (const std::size_t &i) const

  Getter for the i'th value of sigma squared.
- const double & probability_sigma (const std::size_t &i) const

  Getter for the probability of the i'th value of sigma.
- const double & log_probability_sigma (const std::size_t &i) const

  Getter for the log-probability of the i'th value of sigma.
- const double & maxR () const

  Getter for the maximum value of R.
- const double & maxR2 () const

  Getter for the maximum value of R squared.
- const double & max2R () const

  Getter for the maximum value of 2R.
- const double & max2R2 () const

  Getter for the maximum value of 2R squared.
- const double & minScanR () const

  Getter for the lowest value of R to scan.
- const double & maxScanR () const

  Getter for the highest value of R to scan.
- const double & minScanT () const

  Getter for the lowest value of T to scan.
- const double & maxScanT () const

  Getter for the highest value of T to scan.
- const double & dR () const

  Getter for the spacing of value of R to scan.
- const std::size_t & Rbins () const

  Getter for the number of R values to scan.

- const double & dT () const
  Getter for the spacing of value of T to scan.
- const std::size_t & Tbins () const
  Getter for the number of T values to scan.
- const double & logPb () const
  Logarithm of the P_b parameter
- const double & logPbDagger () const
  Logarithm of the (1 - P_b) parameter
- const double & alpha () const
  Getter for the alpha parameter
- const double & logAlpha () const
  Getter for the logarithm of the alpha parameter
- const double & logGammaAlpha () const
  Getter for the logarithm of the gamma function of alpha parameter
- const bool & validate () const
  Getter for whether or not to run the validation on the clustering.
- const std::string & inputFile () const
  Getter for the input file.
- const std::string & outputFile () const
  Getter for the output file.
- const double & ClusterR () const
  Getter for the R value for a clusterization pass.
- const double & ClusterT () const
  Getter for the T value for a clusterization pass.
- double toPhysicalUnits (const double &aAlgorithmUnits) const
  Utility function to convert a normalized algorithm distance to physical distance.
- double toAlgorithmUnits (const double &aPhysicalUnits) const
  Utility function to convert physical distances to a normalized algorithm distances.
- double toPhysicalX (const double &aAlgorithmX) const
  Utility function to convert a normalized algorithm x-coordinate to a physical x-coordinate.
- double toAlgorithmX (const double &aPhysicalX) const
  Utility function to convert a physical x-coordinate to a normalized algorithm x-coordinate.
- double toPhysicalY (const double &aAlgorithmY) const
  Utility function to convert a normalized algorithm y-coordinate to a physical y-coordinate.
- double toAlgorithmY (const double &aPhysicalY) const
  Utility function to convert a physical y-coordinate to a normalized algorithm y-coordinate.
- double getCentreX () const
  Getter for the x-coordinate of the physical centre.
- double getCentreY () const
  Getter for the y-coordinate of the physical centre.
- double getZoom () const
  Getter for the scaling factor applied to the dataset.

Static Public Member Functions

- static Configuration & getInstance ()
  Getter for the singleton instance.

Static Public Attributes

- static Configuration Instance

  A single global copy of the global variables.

Private Attributes

- double mScale

  The scale parameter.
- double mScale2

  The scale parameter squared.
- double mPhysicalCentreX

  The x-coordinate of the centre of the window in physical units.
- double mPhysicalCentreY

  The y-coordinate of the centre of the window in physical units.
- std::size_t mSigmacount

  The number of sigma bins.
- double mSigmaspacing

  The spacing of sigma bins.
- std::vector$<$ double $>$ mSigmabins

  The values of sigma.
- std::vector$<$ double $>$ mSigmabins2

  The values of sigma squared.
- std::vector$<$ double $>$ mProbabilitySigma

  The probability of a given sigma.
- std::vector$<$ double $>$ mLogProbabilitySigma

  The log-probability of a gievn sigma.
- double mMaxR

  The maximum value of R.
- double mMaxR2

  The maximum value of R squared.
- double mMax2R

  The maximum value of 2R
- double mMax2R2

  The maximum value of 2R squared.
- double mMinScanR

  The lowest value of R to scan.
- double mMaxScanR

  The largest value of R to scan.
- double mMinScanT

  The lowest value of T to scan.
- double mMaxScanT

  The largest value of T to scan.
- double mDR

  The spacing of value of R to scan.
- double mDT

  The spacing of value of T to scan.
- std::size_t mRbins

  The number of R values to scan.

- std::size_t mTbins

  The number of T values to scan.
- double mAlpha

  The alpha parameter.
- double mLogAlpha

  Logarithm of the alpha parameter.
- double mLogGammaAlpha

  Logarithm of the gamma function of alpha parameter
- double mLogPb

  Logarithm of the P_b parameter
- double mLogPbDagger

  Logarithm of the(1- P_b) parameter
- bool mValidate

  Whether or not to run the validation on the clustering.
- std::string mInputFile

  The input event file.
- std::string mOutputFile

  The output file.
- double mClusterR

  The value of R for clustering.
- double mClusterT

  The value of T for clustering.

4.2.1 Detailed Description

Class for storing the configuration parameters.

Definition at line 71 of file Configuration.hpp.

4.2.2 Member Function Documentation

4.2.2.1 alpha()

```
const double& Configuration::alpha ( ) const  [inline]
```

Getter for the alpha parameter

Returns

The alpha parameter

Definition at line 224 of file Configuration.hpp.

References mAlpha.

4.2.2.2 ClusterR()

```
const double& Configuration::ClusterR ( ) const  [inline]
```

Getter for the R value for a clusterization pass.

Returns

The R value for a clusterization pass

Definition at line 247 of file Configuration.hpp.

References mClusterR.

4.2.2.3 ClusterT()

```
const double& Configuration::ClusterT ( ) const  [inline]
```

Getter for the T value for a clusterization pass.

Returns

The T value for a clusterization pass

Definition at line 250 of file Configuration.hpp.

References mClusterT.

4.2.2.4 dR()

```
const double& Configuration::dR ( ) const  [inline]
```

Getter for the spacing of value of R to scan.

Returns

The spacing of value of R to scan

Definition at line 204 of file Configuration.hpp.

References mDR.

Referenced by Data::PreprocessLocalizationScores().

4.2.2.5 dT()

```
const double& Configuration::dT ( ) const  [inline]
```

Getter for the spacing of value of T to scan.

Returns

The spacing of value of T to scan

Definition at line 210 of file Configuration.hpp.

References mDT.

Referenced by EventProxy::ScanRT().

4.2.2.6 FromCommandline()

```
void Configuration::FromCommandline (
             int argc,
             char ** argv )
```

Parse the parameters when passed in as commandline arguments.

Parameters

argc	The number of commandline arguments
argv	The commandline arguments

Definition at line 147 of file Configuration.cpp.

References FromVector().

4.2.2.7 FromVector()

```
void Configuration::FromVector (
             const std::vector< std::string > & aArgs )
```

Parse the parameters when passed in as commandline arguments.

Parameters

aArgs	The commandline arguments

Definition at line 153 of file Configuration.cpp.

References GSLInterpolator::Eval(), mClusterR, mClusterT, SetAlpha(), SetCentre(), SetInputFile(), SetOutput↩
File(), SetPb(), SetRBins(), SetSigmaParameters(), SetTBins(), SetValidate(), SetZoom(), and toAlgorithmUnits().

Referenced by ConfigFromVector(), and FromCommandline().

4.2.2.8 getCentreX()

```
double Configuration::getCentreX ( ) const  [inline]
```

Getter for the x-coordinate of the physical centre.

Returns

The x-coordinate of the physical centre

Definition at line 303 of file Configuration.hpp.

References mPhysicalCentreX.

4.2.2.9 getCentreY()

```
double Configuration::getCentreY ( ) const  [inline]
```

Getter for the y-coordinate of the physical centre.

Returns

The y-coordinate of the physical centre

Definition at line 306 of file Configuration.hpp.

References mPhysicalCentreY.

4.2.2.10 getInstance()

```
static Configuration& Configuration::getInstance ( )  [inline], [static]
```

Getter for the singleton instance.

Returns

The singleton instance

Definition at line 317 of file Configuration.hpp.

References Instance.

4.2.2.11 getZoom()

```
double Configuration::getZoom ( ) const  [inline]
```

Getter for the scaling factor applied to the dataset.

Returns

The scaling factor applied to the dataset

Definition at line 309 of file Configuration.hpp.

References mScale.

4.2.2.12 inputFile()

```
const std::string& Configuration::inputFile ( ) const  [inline]
```

Getter for the input file.

Returns

The name of the input event file

Definition at line 239 of file Configuration.hpp.

References mInputFile.

Referenced by Event::Event().

4.2.2.13 log_probability_sigma() [1/2]

```
const std::vector< double >& Configuration::log_probability_sigma ( ) const  [inline]
```

Getter for the log of the probabilities of a given sigma.

Returns

The log of the probabilities of given sigma

Definition at line 157 of file Configuration.hpp.

References mLogProbabilitySigma.

Referenced by Cluster::UpdateLogScore().

4.2.2.14 log_probability_sigma() [2/2]

```
const double& Configuration::log_probability_sigma (
             const std::size_t & i ) const  [inline]
```

Getter for the log-probability of the i'th value of sigma.

Parameters

i	The index of the value of sigma to get the log-probability for

Returns

The log-probability of sigma_i

Definition at line 174 of file Configuration.hpp.

References mLogProbabilitySigma.

4.2.2.15 logAlpha()

```
const double& Configuration::logAlpha ( ) const  [inline]
```

Getter for the logarithm of the alpha parameter

Returns

The logarithm of the alpha parameter

Definition at line 227 of file Configuration.hpp.

References mLogAlpha.

4.2.2.16 logGammaAlpha()

```
const double& Configuration::logGammaAlpha ( ) const  [inline]
```

Getter for the logarithm of the gamma function of alpha parameter

Returns

The logarithm of the gamma function of alpha parameter

Definition at line 230 of file Configuration.hpp.

References mLogGammaAlpha.

4.2.2.17 logPb()

```
const double& Configuration::logPb ( ) const  [inline]
```

Logarithm of the P_b parameter

Returns

Logarithm of the P_b parameter

Definition at line 217 of file Configuration.hpp.

References mLogPb.

Referenced by EventProxy::UpdateLogScore().

4.2.2.18 logPbDagger()

```
const double& Configuration::logPbDagger ( ) const  [inline]
```

Logarithm of the (1 - P_b) parameter

Returns

Logarithm of the (1 - P_b) parameter

Definition at line 220 of file Configuration.hpp.

References mLogPbDagger.

4.2.2.19 max2R()

```
const double& Configuration::max2R ( ) const  [inline]
```

Getter for the maximum value of 2R.

Returns

The maximum value of 2R

Definition at line 184 of file Configuration.hpp.

References mMax2R.

Referenced by Data::Preprocess().

4.2.2.20 max2R2()

```
const double& Configuration::max2R2 ( ) const  [inline]
```

Getter for the maximum value of 2R squared.

Returns

The maximum value of 2R squared

Definition at line 187 of file Configuration.hpp.

References mMax2R2.

4.2.2.21 maxR()

```
const double& Configuration::maxR ( ) const  [inline]
```

Getter for the maximum value of R.

Returns

The maximum value of R

Definition at line 178 of file Configuration.hpp.

References mMaxR.

4.2.2.22 maxR2()

```
const double& Configuration::maxR2 ( ) const  [inline]
```

Getter for the maximum value of R squared.

Returns

The maximum value of R squared

Definition at line 181 of file Configuration.hpp.

References mMaxR2.

4.2.2.23 maxScanR()

```
const double& Configuration::maxScanR ( ) const  [inline]
```

Getter for the highest value of R to scan.

Returns

The highest value of R to scan

Definition at line 194 of file Configuration.hpp.

References mMaxScanR.

4.2.2.24 maxScanT()

```
const double& Configuration::maxScanT ( ) const  [inline]
```

Getter for the highest value of T to scan.

Returns

The highest value of T to scan

Definition at line 200 of file Configuration.hpp.

References mMaxScanT.

Referenced by EventProxy::ScanRT().

4.2.2.25 minScanR()

```
const double& Configuration::minScanR ( ) const  [inline]
```

Getter for the lowest value of R to scan.

Returns

The lowest value of R to scan

Definition at line 191 of file Configuration.hpp.

References mMinScanR.

4.2.2.26 minScanT()

```
const double& Configuration::minScanT ( ) const  [inline]
```

Getter for the lowest value of T to scan.

Returns

The lowest value of T to scan

Definition at line 197 of file Configuration.hpp.

References mMinScanT.

4.2.2.27 outputFile()

```
const std::string& Configuration::outputFile ( ) const  [inline]
```

Getter for the output file.

Returns

The name of the output file

Definition at line 242 of file Configuration.hpp.

References mOutputFile.

4.2.2.28 probability_sigma() [1/2]

```
const std::vector< double >& Configuration::probability_sigma ( ) const  [inline]
```

Getter for the probabilities of a given sigma.

Returns

The probabilities of given sigma

Definition at line 154 of file Configuration.hpp.

References mProbabilitySigma.

4.2.2.29 probability_sigma() [2/2]

```
const double& Configuration::probability_sigma (
             const std::size_t & i ) const  [inline]
```

Getter for the probability of the i'th value of sigma.

Parameters

i	The index of the value of sigma to get the probability for

Returns

The probability of sigma_i

Definition at line 170 of file Configuration.hpp.

References mProbabilitySigma.

4.2.2.30 Rbins()

```
const std::size_t& Configuration::Rbins ( ) const  [inline]
```

Getter for the number of R values to scan.

Returns

The number of R values to scan

Definition at line 207 of file Configuration.hpp.

References mRbins.

Referenced by Data::PreprocessLocalizationScores(), and EventProxy::ScanRT().

4.2.2.31 scale2()

```
const double& Configuration::scale2 ( ) const  [inline]
```

Getter for the scale-parameter squared.

Returns

The scale-parameter squared

Definition at line 136 of file Configuration.hpp.

References mScale2.

4.2.2.32 SetAlpha()

```
void Configuration::SetAlpha (
             const double & aAlpha )
```

Setter for the alpha parameter.

Parameters

aAlpha	The alpha parameter

Definition at line 100 of file Configuration.cpp.

References mAlpha, mLogAlpha, and mLogGammaAlpha.

Referenced by FromVector().

4.2.2.33 SetCentre()

```
void Configuration::SetCentre (
             const double & aPhysicalCentreX,
             const double & aPhysicalCentreY )
```

Setter for the centre of the scan window.

Parameters

aPhysicalCentreX	The x-coordinate of the centre of the window in physical units (becomes 0 in algorithm units)
aPhysicalCentreY	The y-coordinate of the centre of the window in physical units (becomes 0 in algorithm units)

Definition at line 36 of file Configuration.cpp.

References mPhysicalCentreX, and mPhysicalCentreY.

Referenced by FromVector().

4.2.2.34 SetInputFile()

```
void Configuration::SetInputFile (
             const std::string & aFileName )
```

Setter for the input file.

Parameters

aFileName	The name of the file

Definition at line 116 of file Configuration.cpp.

References mInputFile.

Referenced by FromVector().

4.2.2.35 SetOutputFile()

```
void Configuration::SetOutputFile (
            const std::string & aFileName )
```

Setter for the output file.

Parameters

aFileName	The name of the file

Definition at line 123 of file Configuration.cpp.

References mOutputFile.

Referenced by FromVector().

4.2.2.36 SetPb()

```
void Configuration::SetPb (
            const double & aPB )
```

Setter for the P_b parameter.

Parameters

aPB	The P_b parameter

Definition at line 93 of file Configuration.cpp.

References mLogPb, and mLogPbDagger.

Referenced by FromVector().

4.2.2.37 SetRBins()

```
void Configuration::SetRBins (
            const std::size_t & aRbins,
            const double & aMinScanR = 0.0,
            const double & aMaxScanR = -1 )
```

Setter for the R bins for the RT scan.

Parameters

aRbins	The number of R bins to scan over
aMinScanR	The lowest value of R to scan
aMaxScanR	The largest value of R to scan

Definition at line 68 of file Configuration.cpp.

References mDR, mMax2R, mMax2R2, mMaxR, mMaxR2, mMaxScanR, mMinScanR, mRbins, toAlgorithmUnits(), and toPhysicalUnits().

Referenced by FromVector().

4.2.2.38 SetSigmaParameters()

```
void Configuration::SetSigmaParameters (
            const std::size_t & aSigmacount,
            const double & aSigmaMin,
            const double & aSigmaMax,
            const std::function< double(const double &) > & aInterpolator )
```

Setter for the sigma-bins to be integrated over.

Parameters

aSigmacount	The number of sigma bins
aSigmaMin	The lowest sigma bin
aSigmaMax	The highest sigma bin
aInterpolator	Function-object to generate the probability of any given sigma

Definition at line 50 of file Configuration.cpp.

References mLogProbabilitySigma, mProbabilitySigma, mScale, mSigmabins, mSigmabins2, mSigmacount, m↩
Sigmaspacing, and toAlgorithmUnits().

Referenced by FromVector().

4.2.2.39 SetTBins()

```
void Configuration::SetTBins (
            const std::size_t & aTbins,
            const double & aMinScanT = 0.0,
            const double & aMaxScanT = -1 )
```

Parameters

aTbins	The number of T bins to scan over
aMinScanT	The lowest value of T to scan
aMaxScanT	The largest value of T to scan

Definition at line 83 of file Configuration.cpp.

References mDT, mMaxScanT, mMinScanT, mTbins, toAlgorithmUnits(), and toPhysicalUnits().

Referenced by FromVector().

4.2.2.40 SetValidate()

```
void Configuration::SetValidate (
            const bool & aValidate )
```

Set whether to validate clusterization.

Parameters

aValidate	Whether to validate clusterization

Definition at line 108 of file Configuration.cpp.

References mValidate.

Referenced by FromVector().

4.2.2.41 SetZoom()

```
void Configuration::SetZoom (
            const double & aScale )
```

Setter for the half-width of the scan window.

Parameters

aScale	The scale of the window in physical units (becomes ±1 in algorithm units)

Definition at line 43 of file Configuration.cpp.

References mScale, and mScale2.

Referenced by FromVector().

4.2.2.42 sigmabins() [1/2]

```
const std::vector< double >& Configuration::sigmabins ( ) const  [inline]
```

Getter for the values of sigma.

Returns

The values of sigma

Definition at line 148 of file Configuration.hpp.

References mSigmabins.

4.2.2.43 sigmabins() [2/2]

```
const double& Configuration::sigmabins (
            const std::size_t & i ) const  [inline]
```

Getter for the i'th value of sigma.

Parameters

i	The index of the value of sigma to get

Returns

The value of sigma_i

Definition at line 162 of file Configuration.hpp.

References mSigmabins.

4.2.2.44 sigmabins2() [1/2]

```
const std::vector< double >& Configuration::sigmabins2 ( ) const  [inline]
```

Getter for the values of sigma squared.

Returns

The values of sigma squared

Definition at line 151 of file Configuration.hpp.

References mSigmabins2.

4.2.2.45 sigmabins2() [2/2]

```
const double& Configuration::sigmabins2 (
            const std::size_t & i ) const  [inline]
```

Getter for the i'th value of sigma squared.

Parameters

i	The index of the value of sigma squared to get

Returns

The value of sigma_i squared

Definition at line 166 of file Configuration.hpp.

References mSigmabins2.

4.2.2.46 sigmacount()

```
const std::size_t& Configuration::sigmacount ( ) const  [inline]
```

Getter for the sigma count.

Returns

The sigma count

Definition at line 140 of file Configuration.hpp.

References mSigmacount.

Referenced by Cluster::UpdateLogScore(), and EventProxy::ValidateLogScore().

4.2.2.47 sigmaspacing()

```
const double& Configuration::sigmaspacing ( ) const  [inline]
```

Getter for the sigma spacing.

Returns

The sigma spacing

Definition at line 144 of file Configuration.hpp.

References mSigmaspacing.

4.2.2.48 Tbins()

```
const std::size_t& Configuration::Tbins ( ) const  [inline]
```

Getter for the number of T values to scan.

Returns

The number of T values to scan

Definition at line 213 of file Configuration.hpp.

References mTbins.

Referenced by EventProxy::ScanRT().

4.2.2.49 toAlgorithmUnits()

```
double Configuration::toAlgorithmUnits (
             const double & aPhysicalUnits ) const  [inline]
```

Utility function to convert physical distances to a normalized algorithm distances.

Parameters

aPhysicalUnits	A physical distance

Returns

A normalized algorithm distances

Definition at line 264 of file Configuration.hpp.

References mScale.

Referenced by FromVector(), SetRBins(), SetSigmaParameters(), SetTBins(), toAlgorithmX(), and toAlgorithmY().

4.2.2.50 toAlgorithmX()

```
double Configuration::toAlgorithmX (
             const double & aPhysicalX ) const  [inline]
```

Utility function to convert a physical x-coordinate to a normalized algorithm x-coordinate.

Parameters

aPhysicalX	A physical x-coordinate

Returns

A normalized x-coordinate

Definition at line 280 of file Configuration.hpp.

References mPhysicalCentreX, and toAlgorithmUnits().

4.2.2.51 toAlgorithmY()

```
double Configuration::toAlgorithmY (
             const double & aPhysicalY ) const  [inline]
```

Utility function to convert a physical y-coordinate to a normalized algorithm y-coordinate.

Parameters

aPhysicalY	A physical y-coordinate

Returns

A normalized y-coordinate

Definition at line 296 of file Configuration.hpp.

References mPhysicalCentreY, and toAlgorithmUnits().

4.2.2.52 toPhysicalUnits()

```
double Configuration::toPhysicalUnits (
            const double & aAlgorithmUnits ) const  [inline]
```

Utility function to convert a normalized algorithm distance to physical distance.

Parameters

aAlgorithmUnits	A normalized algorithm distance

Returns

A physical distances

Definition at line 256 of file Configuration.hpp.

References mScale.

Referenced by SetRBins(), SetTBins(), toPhysicalX(), toPhysicalY(), and Event::WriteCSV().

4.2.2.53 toPhysicalX()

```
double Configuration::toPhysicalX (
            const double & aAlgorithmX ) const  [inline]
```

Utility function to convert a normalized algorithm x-coordinate to a physical x-coordinate.

Parameters

aAlgorithmX	A normalized x-coordinate

Returns

A physical x-coordinate

Definition at line 272 of file Configuration.hpp.

References mPhysicalCentreX, and toPhysicalUnits().

4.2.2.54 toPhysicalY()

```
double Configuration::toPhysicalY (
             const double & aAlgorithmY ) const  [inline]
```

Utility function to convert a normalized algorithm y-coordinate to a physical y-coordinate.

Parameters

aAlgorithmY	A normalized y-coordinate

Returns

A physical y-coordinate

Definition at line 288 of file Configuration.hpp.

References mPhysicalCentreY, and toPhysicalUnits().

Referenced by Event::WriteCSV().

4.2.2.55 validate()

```
const bool& Configuration::validate ( ) const  [inline]
```

Getter for whether or not to run the validation on the clustering.

Returns

Whether or not to run the validation on the clustering

Definition at line 234 of file Configuration.hpp.

References mValidate.

The documentation for this class was generated from the following files:

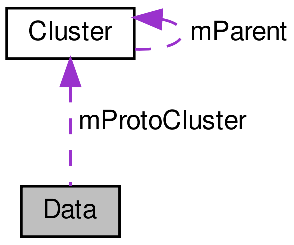
- include/BayesianClustering/Configuration.hpp
- src/BayesianClustering/Configuration.cpp
- src/BayesianClustering/Event.cpp

4.3 Data Class Reference

A class to store the raw data-points.

```
#include <Data.hpp>
```

Collaboration diagram for Data:



Public Member Functions

- Data (const PRECISION &aX, const PRECISION &aY, const PRECISION &aS)

  Constructor.
- Data (const Data &aOther)=delete

  Deleted copy constructor.
- Data & operator= (const Data &aOther)=delete

  Deleted assignment operator.
- Data (Data &&aOther)=default

  Default move constructor.
- Data & operator= (Data &&aOther)=default

  Default move-assignment constructor.
- virtual ∼Data ()

  Destructor.
- bool operator< (const Data &aOther) const

  Comparison operator for sorting data-points by distance from the origin.
- PRECISION dR2 (const Data &aOther) const

  Return the squared-distance of this data-points from another.
- PRECISION dR (const Data &aOther) const

  Return the distance of this data-points from another.
- PRECISION dPhi (const Data &aOther) const

  Return the angle between this data-points and another.
- void Preprocess (std::vector< Data > &aData, const std::size_t &aIndex)

  All the necessary pre-processing to get this data-point ready for an RT-scan.
- void PreprocessLocalizationScores (std::vector< Data > &aData)

  Calculate the localization score from the local neighbourhood.
- PRECISION CalculateLocalizationScore (const std::vector< Data > &aData, const double &R) const

  Calculate the localization score from the local neighbourhood.

Public Attributes

- PRECISION x

  The x-position of the data-point.
- PRECISION y

  The y-position of the data-point.
- PRECISION s

  The sigma of the data-point
- PRECISION r2

  The squared radial distance of the data-point.
- PRECISION r

  The radial distance of the data-point.
- PRECISION phi

  The phi-position of the data-point.
- std::vector< PRECISION > mLocalizationScores

  The locaalization scores, one per R-bin.
- std::vector< std::pair< PRECISION, std::size_t > > mNeighbours

  The list of neighbours as a pair of squared-distance and index into the list of points.
- Cluster ∗ mProtoCluster

  A cluster containing only this data-point.

4.3.1 Detailed Description

A class to store the raw data-points.

Definition at line 14 of file Data.hpp.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Data() [1/3]

```
Data::Data (
            const PRECISION & aX,
            const PRECISION & aY,
            const PRECISION & aS )
```

Constructor.

Parameters

aX	The x-position of the data-point in algorithm units
aY	The y-position of the data-point in algorithm units
aS	The sigma of the data-point in algorithm units

Definition at line 13 of file Data.cpp.

4.3.2.2 Data() [2/3]

```
Data::Data (
            const Data & aOther )  [delete]
```

Deleted copy constructor.

Parameters

aOther	Anonymous argument

4.3.2.3 Data() [3/3]

```
Data::Data (
            Data && aOther )  [default]
```

Default move constructor.

Parameters

aOther	Anonymous argument

4.3.3 Member Function Documentation

4.3.3.1 CalculateLocalizationScore()

```
PRECISION Data::CalculateLocalizationScore (
            const std::vector< Data > & aData,
            const double & R ) const
```

Calculate the localization score from the local neighbourhood.

Todo Remind myself how this works and what the difference is with above

Parameters

aData	?
R	?

Returns

The localization score

Definition at line 109 of file Data.cpp.

References mNeighbours, x, and y.

4.3.3.2 dPhi()

```
PRECISION Data::dPhi (
             const Data & aOther ) const  [inline]
```

Return the angle between this data-points and another.

Returns

The angle between this data-points and another

Parameters

aOther	A data-point to compare against

Definition at line 68 of file Data.hpp.

References phi.

4.3.3.3 dR()

```
PRECISION Data::dR (
             const Data & aOther ) const  [inline]
```

Return the distance of this data-points from another.

Returns

The distance of this data-points from another

Parameters

aOther	A data-point to compare against

Definition at line 60 of file Data.hpp.

References dR2().

4.3.3.4 dR2()

```
PRECISION Data::dR2 (
             const Data & aOther ) const  [inline]
```

Return the squared-distance of this data-points from another.

Returns

The squared-distance of this data-points from another

Parameters

aOther	A data-point to compare against

Definition at line 51 of file Data.hpp.

References x, and y.

Referenced by dR().

4.3.3.5 operator<()

```
bool Data::operator< (
             const Data & aOther ) const  [inline]
```

Comparison operator for sorting data-points by distance from the origin.

Returns

Whether this data-point is closer to the origin than another

Parameters

aOther	A data-point to compare against

Definition at line 43 of file Data.hpp.

References r.

4.3.3.6 operator=() [1/2]

```
Data& Data::operator= (
             const Data & aOther )  [delete]
```

Deleted assignment operator.

Returns

Reference to this, for chaining calls

Parameters

aOther	Anonymous argument

4.3.3.7 operator=() [2/2]

```
Data& Data::operator= (
             Data && aOther )  [default]
```

Default move-assignment constructor.

Returns

Reference to this, for chaining calls

Parameters

aOther	Anonymous argument

4.3.3.8 Preprocess()

```
void Data::Preprocess (
             std::vector< Data > & aData,
             const std::size_t & aIndex )
```

All the necessary pre-processing to get this data-point ready for an RT-scan.

Parameters

aData	The collection of data-points
aIndex	The index of the current data-point

Definition at line 27 of file Data.cpp.

References Configuration::Instance, and Configuration::max2R().

4.3.3.9 PreprocessLocalizationScores()

```
void Data::PreprocessLocalizationScores (
            std::vector< Data > & aData )
```

Calculate the localization score from the local neighbourhood.

Todo Remind myself how this works and what the difference is with below

Parameters

aData	?

Definition at line 71 of file Data.cpp.

References Configuration::dR(), Configuration::Instance, and Configuration::Rbins().

The documentation for this class was generated from the following files:

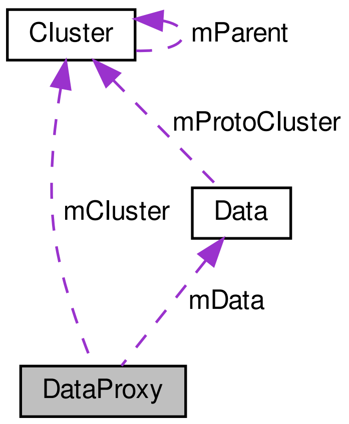
- include/BayesianClustering/Data.hpp
- src/BayesianClustering/Data.cpp

4.4 DataProxy Class Reference

A light-weight proxy for the raw data-points.

```
#include <DataProxy.hpp>
```

Collaboration diagram for DataProxy:



Public Member Functions

- DataProxy (Data &aData)

    Default constructor.
- DataProxy (const DataProxy &aOther)=delete

    Deleted copy constructor.
- DataProxy & operator= (const DataProxy &aOther)=delete

    Deleted assignment operator.
- DataProxy (DataProxy &&aOther)=default

    Default move constructor.
- DataProxy & operator= (DataProxy &&aOther)=default

    Default move-assignment constructor.
- void Clusterize (const PRECISION &a2R2, EventProxy &aEvent)

    Entry point clusterization function - a new cluster will be created.
- void Clusterize (const PRECISION &a2R2, EventProxy &aEvent, Cluster ∗aCluster)

    Recursive clusterization function.
- Cluster ∗ GetCluster ()

    Get a pointer to this data-proxy's ultimate parent cluster (or null if unclustered.

Public Attributes

- Data ∗ mData

    The data-point for which this is the proxy.
- Cluster ∗ mCluster

    This data-proxy's immediate parent cluster.
- bool mExclude

    Whether this data-point is to be included in the clusterization.

4.4.1 Detailed Description

A light-weight proxy for the raw data-points.

Definition at line 17 of file DataProxy.hpp.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 DataProxy() [1/3]

```
DataProxy::DataProxy (
            Data & aData )
```

Default constructor.

Parameters

aData	The data-point for which this is the proxy

Definition at line 10 of file DataProxy.cpp.

4.4.2.2 DataProxy() [2/3]

```
DataProxy::DataProxy (
            const DataProxy & aOther )  [delete]
```

Deleted copy constructor.

Parameters

aOther	Anonymous argument

4.4.2.3 DataProxy() [3/3]

```
DataProxy::DataProxy (
            DataProxy && aOther )  [default]
```

Default move constructor.

Parameters

aOther	Anonymous argument

4.4.3 Member Function Documentation

4.4.3.1 Clusterize() [1/2]

```
void DataProxy::Clusterize (
             const PRECISION & a2R2,
             EventProxy & aEvent )
```

Entry point clusterization function - a new cluster will be created.

Parameters

a2R2	The clusterization radius
aEvent	The event-proxy in which we are running

Definition at line 15 of file DataProxy.cpp.

References mCluster, EventProxy::mClusters, and mExclude.

Referenced by Clusterize().

4.4.3.2 Clusterize() [2/2]

```
void DataProxy::Clusterize (
             const PRECISION & a2R2,
             EventProxy & aEvent,
             Cluster * aCluster )
```

Recursive clusterization function.

Parameters

a2R2	The clusterization radius
aEvent	The event-proxy in which we are running
aCluster	The cluster we are building

Definition at line 23 of file DataProxy.cpp.

References Clusterize(), GetCluster(), EventProxy::GetData(), mCluster, Cluster::mClusterSize, mData, mExclude, Data::mNeighbours, Cluster::mParent, and Data::mProtoCluster.

4.4.3.3 GetCluster()

```
Cluster* DataProxy::GetCluster ( )  [inline]
```

Get a pointer to this data-proxy's ultimate parent cluster (or null if unclustered.

Returns

A pointer to this data-proxy's ultimate parent cluster

Definition at line 51 of file DataProxy.hpp.

References Cluster::GetParent(), and mCluster.

Referenced by Clusterize().

4.4.3.4 operator=() [1/2]

```
DataProxy& DataProxy::operator= (
            const DataProxy & aOther )  [delete]
```

Deleted assignment operator.

Returns

Reference to this, for chaining calls

Parameters

aOther	Anonymous argument

4.4.3.5 operator=() [2/2]

```
DataProxy& DataProxy::operator= (
            DataProxy && aOther )  [default]
```

Default move-assignment constructor.

Returns

Reference to this, for chaining calls

Parameters

aOther	Anonymous argument

The documentation for this class was generated from the following files:

- include/BayesianClustering/DataProxy.hpp
- src/BayesianClustering/DataProxy.cpp

4.5 Event Class Reference

A class which holds the raw event data and global parameters.

```
#include <Event.hpp>
```

Public Member Functions

- Event ()

  Default Constructor.
- Event (const Event &aOther)=delete

  Deleted copy constructor.
- Event & operator= (const Event &aOther)=delete

  Deleted assignment operator.
- Event (Event &&aOther)=default

  Default move constructor.
- Event & operator= (Event &&aOther)=default

  Default move-assignment constructor.
- void Preprocess ()

  All the necessary pre-processing to get the event ready for an RT-scan.
- void ScanRT (const std::function< void(const EventProxy &, const double &, const double &, std::pair< int, int >) > &aCallback)

  Run the scan.
- void Clusterize (const double &R, const double &T, const std::function< void(const EventProxy &) > &a↩Callback)

  Run clusterization for a specific choice of R and T.
- void LoadCSV (const std::string &aFilename)

  Load an event from given file.
- void WriteCSV (const std::string &aFilename)

  Save an event to a file.

Public Attributes

- std::vector< Data > mData

  The collection of raw data points.

4.5.1 Detailed Description

A class which holds the raw event data and global parameters.

Definition at line 16 of file Event.hpp.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 Event() [1/2]

```
Event::Event (
            const Event & aOther )  [delete]
```

Deleted copy constructor.

Parameters

aOther	Anonymous argument

4.5.2.2 Event() [2/2]

```
Event::Event (
            Event && aOther )  [default]
```

Default move constructor.

Parameters

aOther	Anonymous argument

4.5.3 Member Function Documentation

4.5.3.1 Clusterize()

```
void Event::Clusterize (
            const double & R,
            const double & T,
            const std::function< void(const EventProxy &) > & aCallback )
```

Run clusterization for a specific choice of R and T.

Parameters

R	The R parameter for clusterization
T	The T parameter for clusterization
aCallback	A callback for the clusterization results

Definition at line 49 of file Event.cpp.

References EventProxy::Clusterize(), and Preprocess().

Referenced by OneStopGetClusters().

4.5.3.2 LoadCSV()

```
void Event::LoadCSV (
             const std::string & aFilename )
```

Load an event from given file.

Parameters

aFilename	The name of the file to load

Definition at line 106 of file Event.cpp.

References mData.

Referenced by Event().

4.5.3.3 operator=() [1/2]

```
Event& Event::operator= (
             const Event & aOther )  [delete]
```

Deleted assignment operator.

Returns

Reference to this, for chaining calls

Parameters

aOther	Anonymous argument

4.5.3.4 operator=() [2/2]

```
Event& Event::operator= (
             Event && aOther )  [default]
```

Default move-assignment constructor.

Returns

Reference to this, for chaining calls

Parameters

aOther	Anonymous argument

4.5.3.5 ScanRT()

```
void Event::ScanRT (
             const std::function< void(const EventProxy &, const double &, const double &,
std::pair< int, int >) > & aCallback )
```

Run the scan.

Parameters

aCallback	A callback for each RT-scan result

Definition at line 33 of file Event.cpp.

References mData, and Preprocess().

4.5.3.6 WriteCSV()

```
void Event::WriteCSV (
             const std::string & aFilename )
```

Save an event to a file.

Parameters

aFilename	The name of the file to which to save

Definition at line 144 of file Event.cpp.

References Configuration::Instance, mData, Configuration::toPhysicalUnits(), and Configuration::toPhysicalY().

The documentation for this class was generated from the following files:

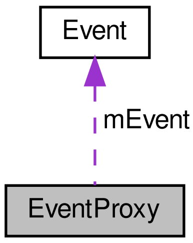
- include/BayesianClustering/Event.hpp
- src/BayesianClustering/Event.cpp

4.6 EventProxy Class Reference

A lightweight wrapper for the event to store clusters for a given scan.

```
#include <EventProxy.hpp>
```

Collaboration diagram for EventProxy:



Public Member Functions

- EventProxy (Event &aEvent)

    Default constructor.
- EventProxy (const EventProxy &aOther)=delete

    Deleted copy constructor.
- EventProxy & operator= (const EventProxy &aOther)=delete

    Deleted assignment operator.
- EventProxy (EventProxy &&aOther)=default

    Default move constructor.
- EventProxy & operator= (EventProxy &&aOther)=default

    Default move-assignment constructor.
- void CheckClusterization (const double &R, const double &T)

    Run validation tests on the clusters.
- void ScanRT (const std::function< void(const EventProxy &, const double &, const double &, std::pair< int, int >) > &aCallback, const uint8_t &aParallelization=1, const uint8_t &aOffset=0)

    Run an RT-scan.
- void Clusterize (const double &R, const double &T, const std::function< void(const EventProxy &) > &a↵ Callback)

Run clusterization for a specific choice of R and T.
- void UpdateLogScore ()

  Update log-probability after a scan.
- void ValidateLogScore ()

  Sean's validation code for testing when the running log-score fails.
- DataProxy & GetData (const std::size_t &aIndex)

  Get the proxy for the Nth neighbour of this data-point.

Public Attributes

- std::vector< DataProxy > mData

  The collection of lightweight data-point wrappers used by this event wrapper.
- std::vector< Cluster > mClusters

  The collection of clusters found by this scan.
- std::size_t mClusteredCount

  The number of clustered data-points.
- std::size_t mBackgroundCount

  The number of background data-points.
- std::size_t mClusterCount

  The number of non-Null clusters.
- double mLogP

  The log-probability density associated with the last scan.

Private Attributes

- const Event & mEvent

  The underlying event this is a proxy to.

4.6.1 Detailed Description

A lightweight wrapper for the event to store clusters for a given scan.

Definition at line 16 of file EventProxy.hpp.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 EventProxy() [1/3]

```
EventProxy::EventProxy (
            Event & aEvent )
```

Default constructor.

Parameters

aEvent	An event for which this is a lightweight proxy

Definition at line 17 of file EventProxy.cpp.

References mClusters, Event::mData, and mData.

4.6.2.2 EventProxy() [2/3]

```
EventProxy::EventProxy (
            const EventProxy & aOther )  [delete]
```

Deleted copy constructor.

Parameters

aOther	Anonymous argument

4.6.2.3 EventProxy() [3/3]

```
EventProxy::EventProxy (
            EventProxy && aOther )  [default]
```

Default move constructor.

Parameters

aOther	Anonymous argument

4.6.3 Member Function Documentation

4.6.3.1 CheckClusterization()

```
void EventProxy::CheckClusterization (
            const double & R,
            const double & T )
```

Run validation tests on the clusters.

Parameters

R	The R of the last run scan
T	The T of the last run scan

Definition at line 25 of file EventProxy.cpp.

References GetData(), mBackgroundCount, mClusterCount, mClusters, and mData.

4.6.3.2 Clusterize()

```
void EventProxy::Clusterize (
            const double & R,
            const double & T,
            const std::function< void(const EventProxy &) > & aCallback )
```

Run clusterization for a specific choice of R and T.

Parameters

R	The R parameter for clusterization
T	The T parameter for clusterization
aCallback	A callback for the clusterization results

Definition at line 135 of file EventProxy.cpp.

References mClusters, Event::mData, mData, mEvent, and UpdateLogScore().

Referenced by Event::Clusterize().

4.6.3.3 GetData()

```
DataProxy& EventProxy::GetData (
            const std::size_t & aIndex )  [inline]
```

Get the proxy for the Nth neighbour of this data-point.

Returns

A reference to the neighbour data-proxy

Parameters

aIndex	The index of the neighbour we are looking for

Definition at line 63 of file EventProxy.hpp.

References mData.

Referenced by CheckClusterization(), and DataProxy::Clusterize().

4.6.3.4 operator=() [1/2]

```
EventProxy& EventProxy::operator= (
             const EventProxy & aOther )  [delete]
```

Deleted assignment operator.

Returns

Reference to this, for chaining calls

Parameters

aOther	Anonymous argument

4.6.3.5 operator=() [2/2]

```
EventProxy& EventProxy::operator= (
             EventProxy && aOther )  [default]
```

Default move-assignment constructor.

Returns

Reference to this, for chaining calls

Parameters

aOther	Anonymous argument

4.6.3.6 ScanRT()

```
void EventProxy::ScanRT (
             const std::function< void(const EventProxy &, const double &, const double &,
std::pair< int, int >) > & aCallback,
             const uint8_t & aParallelization = 1,
             const uint8_t & aOffset = 0 )
```

Run an RT-scan.

Parameters

aCallback	A callback for each RT-scan result
aParallelization	The stride with which we will iterate across RT parameters
aOffset	The starting point for the strides as we iterate across RT parameters

Definition at line 97 of file EventProxy.cpp.

References Configuration::dT(), Configuration::Instance, Configuration::maxScanT(), Configuration::Rbins(), and Configuration::Tbins().

The documentation for this class was generated from the following files:

- include/BayesianClustering/EventProxy.hpp
- src/BayesianClustering/EventProxy.cpp

4.7 GSLInterpolator Class Reference

A utility wrapper around the GSL interpolator to give it a clean C++ interface.

```
#include <GSLInterpolator.hpp>
```

Public Member Functions

- GSLInterpolator (const gsl_interp_type ∗type, const unsigned int &ndata)

  Empty splice constructor.
- GSLInterpolator (const gsl_interp_type ∗type, const std::vector< double > &x, const std::vector< double > &y)

  Initialised splice constructor.
- virtual ∼GSLInterpolator ()

  Destructor.
- GSLInterpolator (const GSLInterpolator &aOther)=delete

  Deleted copy constructor.
- GSLInterpolator & operator= (const GSLInterpolator &aOther)=delete

  Deleted assignment operator.
- GSLInterpolator (GSLInterpolator &&aOther)=default

  Default move constructor.
- GSLInterpolator & operator= (GSLInterpolator &&aOther)=default

  Default move-assignment constructor.
- bool SetData (const std::vector< double > &x, const std::vector< double > &y)

  Set the spline data points.
- bool SetData (const unsigned int &ndata, const double ∗x, const double ∗y)

  Set the spline data points.
- double Evaluate (const std::function< int(double &) > &aFunction, const std::string &aName)

Utility function that runs the GSL function that has been wrapped in a lambda below.

- double Eval (const double &x)

  Evaluate the spline at the given x.
- double Deriv (const double &x)

  The first derivative of the spline at the given x.
- double Deriv2 (const double &x)

  The second derivative of the spline at the given x.
- double Integ (const double &a, const double &b)

  The integral over the spline between two bounds.

Private Attributes

- unsigned int nErrors

  An error counter to suppress excess messages.
- gsl_interp_accel ∗ fAccel

  Underlying GSL machinery.
- gsl_spline ∗ fSpline

  Underlying GSL machinery for the spline itself.
- const gsl_interp_type ∗ fInterpType

  Underlying GSL machinery for the interpolation type.

4.7.1 Detailed Description

A utility wrapper around the GSL interpolator to give it a clean C++ interface.

Definition at line 18 of file GSLInterpolator.hpp.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 GSLInterpolator() [1/4]

```
GSLInterpolator::GSLInterpolator (
             const gsl_interp_type ∗ type,
             const unsigned int & ndata )
```

Empty splice constructor.

Parameters

type	The spline type
ndata	The number of points that will be added to the spline

Definition at line 7 of file GSLInterpolator.cpp.

References fInterpType, and fSpline.

4.7.2.2 GSLInterpolator() `[2/4]`

```
GSLInterpolator::GSLInterpolator (
            const gsl_interp_type * type,
            const std::vector< double > & x,
            const std::vector< double > & y )
```

Initialised splice constructor.

Parameters

type	The spline type
x	The points on the x-axis
y	The points on the y-axis

Definition at line 17 of file GSLInterpolator.cpp.

References fInterpType, fSpline, and SetData().

4.7.2.3 GSLInterpolator() `[3/4]`

```
GSLInterpolator::GSLInterpolator (
            const GSLInterpolator & aOther )  [delete]
```

Deleted copy constructor.

Parameters

aOther	Anonymous argument

4.7.2.4 GSLInterpolator() `[4/4]`

```
GSLInterpolator::GSLInterpolator (
            GSLInterpolator && aOther )  [default]
```

Default move constructor.

Parameters

aOther	Anonymous argument

4.7.3 Member Function Documentation

4.7.3.1 Deriv()

```
double GSLInterpolator::Deriv (
            const double & x )  [inline]
```

The first derivative of the spline at the given x.

Parameters

x	The x-coordinate at which to evaluate the derivative

Returns

The first derivative of the spline at the given x-coordinate

Definition at line 100 of file GSLInterpolator.hpp.

References Evaluate(), fAccel, and fSpline.

4.7.3.2 Deriv2()

```
double GSLInterpolator::Deriv2 (
            const double & x )  [inline]
```

The second derivative of the spline at the given x.

Parameters

x	The x-coordinate at which to evaluate the derivative

Returns

The second derivative of the spline at the given x-coordinate

Definition at line 108 of file GSLInterpolator.hpp.

References Evaluate(), fAccel, and fSpline.

4.7.3.3 Eval()

```
double GSLInterpolator::Eval (
            const double & x )  [inline]
```

Evaluate the spline at the given x.

Parameters

x	The x-coordinate at which to evaluate the spline

Returns

The value of the spline at the given x-coordinate

Definition at line 92 of file GSLInterpolator.hpp.

References Evaluate(), fAccel, and fSpline.

Referenced by Configuration::FromVector().

4.7.3.4 Evaluate()

```
double GSLInterpolator::Evaluate (
             const std::function< int(double &) > & aFunction,
             const std::string & aName )  [inline]
```

Utility function that runs the GSL function that has been wrapped in a lambda below.

Parameters

aFunction	A lambda that will be evaluated
aName	The operation name for the debugging messages

Returns

The interpolated value

Definition at line 73 of file GSLInterpolator.hpp.

References fAccel, and nErrors.

Referenced by Deriv(), Deriv2(), Eval(), and Integ().

4.7.3.5 Integ()

```
double GSLInterpolator::Integ (
             const double & a,
             const double & b )  [inline]
```

The integral over the spline between two bounds.

Parameters

a	The lower bound of the integral
b	The upper bound of the integral

Returns

The integral over the spline between a and b

Definition at line 117 of file GSLInterpolator.hpp.

References Evaluate(), fAccel, and fSpline.

4.7.3.6 operator=() [1/2]

```
GSLInterpolator& GSLInterpolator::operator= (
            const GSLInterpolator & aOther )  [delete]
```

Deleted assignment operator.

Returns

Reference to this, for chaining calls

Parameters

aOther	Anonymous argument

4.7.3.7 operator=() [2/2]

```
GSLInterpolator& GSLInterpolator::operator= (
            GSLInterpolator && aOther )  [default]
```

Default move-assignment constructor.

Returns

Reference to this, for chaining calls

Parameters

aOther	Anonymous argument

4.7.3.8 SetData() [1/2]

```
bool GSLInterpolator::SetData (
            const std::vector< double > & x,
            const std::vector< double > & y ) [inline]
```

Set the spline data points.

Parameters

x	The x-coordinates of the datapoints
y	The y-coordinates of the datapoints

Returns

success or fail

Definition at line 56 of file GSLInterpolator.hpp.

Referenced by GSLInterpolator().

4.7.3.9 SetData() [2/2]

```
bool GSLInterpolator::SetData (
            const unsigned int & ndata,
            const double * x,
            const double * y )
```

Set the spline data points.

Parameters

ndata	The number of data points
x	Pointer to the first element of an array of x-coordinates
y	Pointer to the first element of an array of y-coordinates

Returns

success or fail

Definition at line 36 of file GSLInterpolator.cpp.

References fAccel, fInterpType, fSpline, and nErrors.

The documentation for this class was generated from the following files:

- include/Utilities/GSLInterpolator.hpp
- src/Utilities/GSLInterpolator.cpp

4.8 Cluster::Parameter Struct Reference

A struct representing the cluster parameters.

```
#include <Cluster.hpp>
```

Public Member Functions

- Parameter ()

  Default constructor.
- Parameter & operator+= (const Parameter &aOther)

  Add another set of parameters to this set.
- double log_score () const

  Convert the parameters to a log-probability.
- double alt_log_score () const

  Sean's alternative function to calculate the log-score using only the A's and B's as per the original paper for debugging.

Public Attributes

- PRECISION A

  Parameter A defined in the math.
- PRECISION Bx

  Parameter Bx defined in the math.
- PRECISION By

  Parameter By defined in the math.
- PRECISION C

  Parameter C defined in the math.
- PRECISION logF

  Parameter logF defined in the math.
- PRECISION weightedCentreX

  Parameters added by Sean for validation.
- PRECISION weightedCentreY

  Parameters added by Sean for validation.
- PRECISION S2

  Parameters added by Sean for validation.

4.8.1 Detailed Description

A struct representing the cluster parameters.

Definition at line 19 of file Cluster.hpp.

4.8.2 Member Function Documentation

4.8.2.1 alt_log_score()

```
double Cluster::Parameter::alt_log_score ( ) const
```

Sean's alternative function to calculate the log-score using only the A's and B's as per the original paper for debugging.

Returns

the log-probability of this set of cluster parameters

Definition at line 50 of file Cluster.cpp.

4.8.2.2 log_score()

```
double Cluster::Parameter::log_score ( ) const
```

Convert the parameters to a log-probability.

Returns

the log-probability of this set of cluster parameters

Definition at line 75 of file Cluster.cpp.

4.8.2.3 operator+=()

```
Cluster::Parameter & Cluster::Parameter::operator+= (
            const Parameter & aOther )
```

Add another set of parameters to this set.

Parameters

aOther	Another set of parameters to add to this set

Returns

Reference to this, for chaining calls

Definition at line 32 of file Cluster.cpp.

References A, Bx, By, C, and logF.

The documentation for this struct was generated from the following files:

- include/BayesianClustering/Cluster.hpp
- src/BayesianClustering/Cluster.cpp

4.9 ProgressBar Struct Reference

A utility progress-bar.

```
#include <ProgressBar.hpp>
```

Public Member Functions

- ProgressBar (const std::string &aLabel, const uint32_t &aMax)

  Constructor.
- virtual ∼ProgressBar ()

  Destructor.
- void operator++ ()

  Postfix increment.
- void operator++ (int aDummy)

  Prefix increment.

Public Attributes

- float mBlockSize

  The size of each increment.
- float mNextThreshold

  The next threshold at which we will write a block to stdout.
- std::size_t mCount

  The number of times we have incremented.
- std::chrono::high_resolution_clock::time_point mStart

  A timer for end-of-task stats.

4.9.1 Detailed Description

A utility progress-bar.

Definition at line 6 of file ProgressBar.hpp.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 ProgressBar()

```
ProgressBar::ProgressBar (
            const std::string & aLabel,
            const uint32_t & aMax )
```

Constructor.

Parameters

aLabel	A description of the task being timed
aMax	The number of calls equalling 100%

Definition at line 7 of file ProgressBar.cpp.

4.9.3 Member Function Documentation

4.9.3.1 operator++()

```
void ProgressBar::operator++ (
              int aDummy )
```

Prefix increment.

Parameters

aDummy	Anonymous argument

Definition at line 27 of file ProgressBar.cpp.

References operator++().

The documentation for this struct was generated from the following files:

- include/Utilities/ProgressBar.hpp
- src/Utilities/ProgressBar.cpp

4.10 ProgressBar2 Struct Reference

A utility code timer.

```
#include <ProgressBar.hpp>
```

Public Member Functions

- ProgressBar2 (const std::string &aLabel, const uint32_t &aMax)

  Constructor.
- virtual ∼ProgressBar2 ()

  Destructor.
- void operator++ ()

  Postfix increment.
- void operator++ (int aDummy)

  Prefix increment.

Public Attributes

- std::chrono::high_resolution_clock::time_point mStart

  A timer for end-of-task stats.

4.10.1 Detailed Description

A utility code timer.

Definition at line 34 of file ProgressBar.hpp.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 ProgressBar2()

```
ProgressBar2::ProgressBar2 (
            const std::string & aLabel,
            const uint32_t & aMax )
```

Constructor.

Parameters

aLabel	A description of the task being timed
aMax	The number of calls equalling 100%

Definition at line 32 of file ProgressBar.cpp.

4.10.3 Member Function Documentation

4.10.3.1 operator++()

```
void ProgressBar2::operator++ (
            int aDummy )
```

Prefix increment.

Parameters

aDummy	Anonymous argument

Definition at line 44 of file ProgressBar.cpp.

References operator++().

The documentation for this struct was generated from the following files:

- include/Utilities/ProgressBar.hpp
- src/Utilities/ProgressBar.cpp

4.11 PyIterator< U > Struct Template Reference

A python iterator over a C++ container.

Public Member Functions

- PyIterator (const std::vector< U > &aData)

  Constructor.
- const U & next ()

  Return the current value and advance the iterator.

Public Attributes

- std::vector< U >::const_iterator mIt

  The current location of the iterator.
- const std::vector< U >::const_iterator mEnd

  The end of the underlying container.

4.11.1 Detailed Description

template<typename U>
struct PyIterator< U >

A python iterator over a C++ container.

Todo There must be an out-of-the box way, but I can't find it

Template Parameters

U	The type of the data in the C++ container

Definition at line 45 of file PythonBindings.cpp.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 PyIterator()

```
template<typename U >
PyIterator< U >::PyIterator (
             const std::vector< U > & aData )  [inline]
```

Constructor.

Parameters

aData	The underlying data to be iterated over

Definition at line 54 of file PythonBindings.cpp.

4.11.3 Member Function Documentation

4.11.3.1 next()

```
template<typename U >
const U& PyIterator< U >::next ( )  [inline]
```

Return the current value and advance the iterator.

Returns

The current value

Definition at line 58 of file PythonBindings.cpp.

The documentation for this struct was generated from the following file:

- src/PythonBindings/PythonBindings.cpp

4.12 PyIterator< U * > Struct Template Reference

A partial specialization creating a python iterator over a C++ container of pointers.

Public Member Functions

- PyIterator (const std::vector< U * > &aData)

  Constructor.
- const U & next ()

  Return the current value and advance the iterator.

Public Attributes

- std::vector< U * >::const_iterator mIt

  The current location of the iterator.
- const std::vector< U * >::const_iterator mEnd

  The end of the underlying container.

4.12.1 Detailed Description

template<typename U>
struct PyIterator< U * >

A partial specialization creating a python iterator over a C++ container of pointers.

Todo There must be an out-of-the box way, but I can't find it

Template Parameters

U	The type of the pointers in the C++ container

Definition at line 74 of file PythonBindings.cpp.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 PyIterator()

```
template<typename U >
PyIterator< U * >::PyIterator (
            const std::vector< U * > & aData )  [inline]
```

Constructor.

Parameters

aData	The underlying data to be iterated over

Definition at line 83 of file PythonBindings.cpp.

4.12.3 Member Function Documentation

4.12.3.1 next()

```
template<typename U >
const U& PyIterator< U * >::next ( )  [inline]
```

Return the current value and advance the iterator.

Returns

The current value

Definition at line 87 of file PythonBindings.cpp.

References PyIterator< U >::mEnd, and PyIterator< U >::mIt.

The documentation for this struct was generated from the following file:

- src/PythonBindings/PythonBindings.cpp

Chapter 5

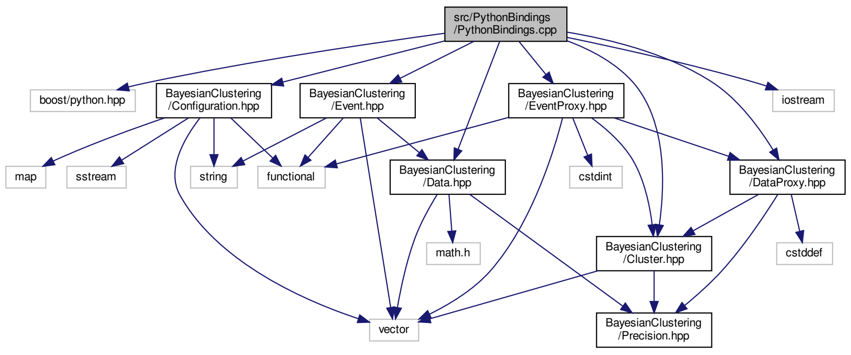
File Documentation

5.1 src/PythonBindings/PythonBindings.cpp File Reference

Self-contained sourcefile for producing python-bindings.

```
#include <boost/python.hpp>
#include "BayesianClustering/Configuration.hpp"
#include "BayesianClustering/Event.hpp"
#include "BayesianClustering/EventProxy.hpp"
#include "BayesianClustering/Cluster.hpp"
#include "BayesianClustering/Data.hpp"
#include "BayesianClustering/DataProxy.hpp"
#include <iostream>
```

Include dependency graph for PythonBindings.cpp:



Classes

- struct PyIterator< U >

  A python iterator over a C++ container.
- struct PyIterator< U ∗ >

  A partial specialization creating a python iterator over a C++ container of pointers.

Functions

- template< typename T >
  std::vector< T > py_list_to_std_vector (const boost::python::object &aIterable)

  Utility function to convert a python list to STL vector.
- template< class T >
  boost::python::list std_vector_to_py_list (const std::vector< T > &aVector)

  Utility function to convert an STL vector to python list.
- void ConfigFromVector (const boost::python::object &aList)

  Set the Bayesian-clustering configuration from a vector of arguments.
- void OneStopGetClusters (const boost::python::object &aCallback)

  Run a 1-pass clustering for a specified R & T and pass the results to a callback function.
- PyIterator< Data ∗ > Cluster_GetIterator (const Cluster &aCluster)

  Utility function to get a python iterator over all the data points in a clusters.
- std::size_t Cluster_GetSize (Cluster &aCluster)

  Utility function to get the number of data points in a clusters.
- PyIterator< Data > Event_GetIterator (const Event &aEvent)

  Utility function to get a python iterator over all the data points in an event.
- std::size_t Event_GetSize (Event &aEvent)

  Utility function to get the number of data points in an event.
- BOOST_PYTHON_MODULE (BayesianClustering)

  Boost Python Wrapper providing bindings for our C++ functions.

5.1.1 Detailed Description

Self-contained sourcefile for producing python-bindings.

5.1.2 Function Documentation

5.1.2.1 Cluster_GetIterator()

```
PyIterator<Data*> Cluster_GetIterator (
             const Cluster & aCluster )
```

Utility function to get a python iterator over all the data points in a clusters.

Parameters

aCluster	The cluster over which we are iterating

Returns

An iterator object pointing to a member of the cluster

Definition at line 138 of file PythonBindings.cpp.

References Cluster::mData.

Referenced by BOOST_PYTHON_MODULE().

5.1.2.2 Cluster_GetSize()

```
std::size_t Cluster_GetSize (
            Cluster & aCluster )
```

Utility function to get the number of data points in a clusters.

Parameters

aCluster	The cluster we are inspecting

Returns

The number of data points in the clusters

Definition at line 143 of file PythonBindings.cpp.

References Cluster::mData.

Referenced by BOOST_PYTHON_MODULE().

5.1.2.3 ConfigFromVector()

```
void ConfigFromVector (
            const boost::python::object & aList )
```

Set the Bayesian-clustering configuration from a vector of arguments.

Parameters

aList	A list of strings to parse as config arguments

Definition at line 102 of file PythonBindings.cpp.

References Configuration::FromVector(), and Configuration::Instance.

Referenced by BOOST_PYTHON_MODULE().

5.1.2.4 Event_GetIterator()

```
PyIterator<Data> Event_GetIterator (
             const Event & aEvent )
```

Utility function to get a python iterator over all the data points in an event.

Parameters

aEvent	The event over which we are iterating

Returns

An iterator object pointing to a member of the event

Definition at line 151 of file PythonBindings.cpp.

References Event::mData.

Referenced by BOOST_PYTHON_MODULE().

5.1.2.5 Event_GetSize()

```
std::size_t Event_GetSize (
             Event & aEvent )
```

Utility function to get the number of data points in an event.

Parameters

aEvent	The event we are inspecting

Returns

The number of data points in the event

Definition at line 156 of file PythonBindings.cpp.

References Event::mData.

Referenced by BOOST_PYTHON_MODULE().

5.1.2.6 OneStopGetClusters()

```
void OneStopGetClusters (
             const boost::python::object & aCallback )
```

Run a 1-pass clustering for a specified R & T and pass the results to a callback function.

Parameters

aCallback	A callback to which results are passed

Definition at line 109 of file PythonBindings.cpp.

References Event::Clusterize(), Configuration::Instance, EventProxy::mClusters, and EventProxy::mData.

Referenced by BOOST_PYTHON_MODULE().

5.1.2.7 py_list_to_std_vector()

```
template<typename T >
std::vector< T > py_list_to_std_vector (
            const boost::python::object & aIterable )  [inline]
```

Utility function to convert a python list to STL vector.

Template Parameters

T	The object type in the STL container

Parameters

aIterable	The python list to convert

Returns

The python data in an STL vector

Definition at line 23 of file PythonBindings.cpp.

5.1.2.8 std_vector_to_py_list()

```
template<class T >
boost::python::list std_vector_to_py_list (
            const std::vector< T > & aVector )  [inline]
```

Utility function to convert an STL vector to python list.

Template Parameters

T	The object type in the STL container

Parameters

aVector	The STL vector to convert

Returns

The STL data in a python list

Definition at line 33 of file PythonBindings.cpp.

Index