# HowMuchData

0.7.0

Generated by Doxygen 1.8.14

# Contents

# Chapter 1

# Observations on Taleb's "How much data do you need"

### Introduction

Nassim Nicholas Taleb has recently released a paper entitled *How much data do you need? An operational metric for fat-tailedness.* to appear in the International Journal of Forecasting. The paper focuses on the preasymptotic behavior of the sum of independent identically distributed random variables that are governed by various fat-tailed distributions. He introduces a metric labeled "kappa" that's related to the growth rate of the mean absolute deviation (MAD) as the number of summands increases. Kappa is defined by the following formula.

kappa(n0,n) = 2- (log(n)-log(n0))/(log(MAD(n) - MAD(n0))

Normally distributed variables have a kappa of zero. Thus the extent to which kappa is in excess of zero measures the "fat-tailedness".

### Purpose

Taleb analyzes a number of distributions to estimate their kappa. He gives explicit formulae for kappa(1,2), which he was able to derive analytically, but he also includes tables of kappa(1,30) and kappa(1,100) for two distributions, namely the Pareto distribution and Student's t distribution. This package is my effort to replicate those tables and perhaps add others for the other distributions. It's a work in process, as there are some items that I can't closely replicate.

### What's included

I've experimented with two algorithms both implemented in C++, one of which relies on monte carlo simulations and the other of which relies on the use of the discrete fourier transform of the characteristic function of the convolution of the distribution functions. The package includes four files with code.

- convolution_test.cpp. The main program to nun the convolution test.
- monte_carlo_test.cpp. The main program to run the monte carlo test.
- pareto_distribution.h. Contains a class for the pareto distribution, modeled on the classes in boost::random and including items normally computed in boost::math::statistical_distributions

- student_t_distribution.h. Similar to pareto but starts as a derived class from boost::random::student_t_distribution
- exponential_distribution.h. A derived class from boost::random::exponential_distribution
- lognormal_distribution.h. A derived class from boost::random::lognormal_distribution. The class implementing the distribution includes several versions of the calculation of the characteristic function, some based on numerical integration from the definition, one based on a p-spline approximation to the more accurate but much slower integrals, and one based on a approximation using Lambert W functions.
- lognormal_test.cpp. A program to test the various version of the calculation of the characteristic function. So far the Lambert W version seems to be the best compromise between speed and accuracy, but it's not without problems.

In order to improve portability, a meson.build file is included, which allows an easy port to other systems once the needed packages are installed.

## Observations

So far my results are close to Taleb's except for the cases where alpha is close to one. As Taleb mentions in his paper such distributions require huge amounts of data to produce reasonable estimates of the MAD and this fact is mirrored in the number of monte carlo runs or in the size the arrays used in the fast fourier transform. I'm pushing the limit of my computer's capability for the cases where alpha = 1.25 or alpha = 1.5. The convolution is limited by the size of available memory and the monte carlo approach is limited by the amount of time and the number of processors available.

I've experimented with other measures of scale, such as the 95% confidence interval spread, for which the amount of computation needed is much more modest, but these results may not be relevant if MAD is the measure which best characterizes the uncertainty.

## To Do

- Switch to the parallel version of the fftw.

## Acknowledgements

1. The package makes heavy use of the Boost C++ headers available at `boost`).
2. The package uses the Eigen headers for the purpose of wrapping the fast fourier transform code, and also uses unsupported Eigen headers for the calculation of splines. These are available at `Eigen`.
3. As distributed the Eigen header wraps the fftw3 available at [fftw]{`http://fftw.org`}. The fftw uses a GPL, so if you want a less restrictive license you should delete the line #define EIGEN_FFTW_DEFAULT at the beginning of convolution, which well revert to the kissfft.
4. One of the calculatations of the characteristic function for the lognormal distribution uses Lambert W functions. I've used the C++ code for the complex Lambert W function from `Istvan Mezo's web page`.

## License

The code included here is covered the the MIT license.

# Chapter 2

# Hierarchical Index

## 2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

# Chapter 3

# Class Index

## 3.1 Class List

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

# Chapter 4

# File Index

## 4.1 File List

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

# Chapter 5

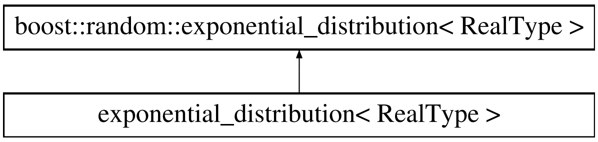
# Class Documentation

## 5.1 exponential_distribution< RealType > Struct Template Reference

instnaces of sturct exponential_distribution generate random variates for exponential distribution

```
#include <exponential_distribution.h>
```

Inheritance diagram for exponential_distribution< RealType >:


boost::random::exponential_distribution< RealType >
exponential_distribution< RealType >

### Public Member Functions

- exponential_distribution (RealType lambda)

  *constructor give lambda*
- RealType cdf (RealType x, bool lower_tail=true) const

  *return the cdf or the complement of the cdf*
- RealType pdf (RealType x) const

  *return the pdf given x*
- RealType quantile (RealType p) const

  *return the quantile give the probability p*
- RealType lambda () const

  *return the lambda paramter of the distribution*
- RealType alpha_stable () const

  *return the alpha of the asymptotic stable distribution */*
- RealType mean () const

  *Return mean of distribution.*
- RealType mad () const

  *Return mean average deviation of the distribution.*
- RealType mad2 () const

  *Return the mad of the square of the distribution.*
- RealType ci (RealType level=RealType(.05)) const

  *Return the confidence interval of the distribution.*
- complex< RealType > characteristic_function (RealType omega) const

  *return the characteristic function of the distribution given omega*

### Friends

- template<class charT , class traits >
  std::basic_ostream< charT, traits > & operator<< (std::basic_ostream< charT, traits > &os, const exponential_distribution &dist)

  *Write distribution to std::ostream.*

### 5.1.1 Detailed Description

**template<class RealType = double>**
**struct exponential_distribution< RealType >**

instnaces of sturct exponential_distribution generate random variates for exponential distribution

**Author**

Created by Joseph Dunn on 1/3/19.

**Copyright**

© 2019 Joseph Dunn. All rights reserved.

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

- /Users/jdunn/Documents/XCode/how_much_data/convolution_test/exponential_distribution.h

## 5.2 Job Struct Reference

instances] of Job used to parcel characteristic funciion calculation to threads

### Public Member Functions

- Job (int nmin, int nmax, int nchunk)

  *consturctor*
- bool get_next (int &nstart, int &nend)

  *get a range and return true if okay*

### Public Attributes

- int nmin

  *the overall minimum index*
- int nmax

  *the overall maximum index*
- int ncurrent

  *the next index to parcel out*
- int nchunk

  *the size of the range to parcel out*
- mutex job_mutex

  *to prevent multiple accesses*

### 5.2.1 Detailed Description

instances] of Job used to parcel characteristic funciion calculation to threads

### 5.2.2 Constructor & Destructor Documentation

#### 5.2.2.1 Job()

```
Job::Job (
             int nmin,
             int nmax,
             int nchunk )  [inline]
```

consturctor

**Parameters**

| | | |
|---|---|---|
| in | *nmin* | the overall minimum index |
| in | *nmax* | the overall maximum index |
| in | *nchunk* | the size of the range to parcel out |

### 5.2.3 Member Function Documentation

#### 5.2.3.1 get_next()

```
bool Job::get_next (
             int & nstart,
             int & nend )  [inline]
```

get a range and return true if okay

**Parameters**

| | |
|---|---|
| *nstart* | start of the assigned range |
| *nend* | next index after assigned range |

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

- /Users/jdunn/Documents/XCode/how_much_data/convolution_test/convolution_test.cpp

## 5.3 KappaResult Struct Reference

structure holding the results of a run for a single alpha

### Public Member Functions

- KappaResult (vector< int > ns)

  *constructor*
- void calc_kappa ()

  *calculate the kappa from the made variable*
- void initialize (double param_in, const vector< int > &ns_in)

  *initialize the structure*
- void update_dev (size_t m, size_t m_ci, const vector< double > &dev, const vector< double > &abs_dev, const vector< vector< double > > &x_in)

  *update the deviations w result from one thread*
- void update_conf_int (double ci_level)

  *calculate the confidence interval for all trials*

### Public Attributes

- double param

  *the parameter for the run*
- int nsize

  *the size of the vector pased to fft*
- double mad_rel_err

  *the relative error of mad vs theory*
- vector< int > ns

  *the durations saved*
- vector< double > mad

  *the mean absolute deviation by duration*
- vector< double > **kappa_mad**
- double ci_rel_err

  *the kappa_mad by duration*
- vector< double > ci

  *the conficence interval by duration*
- vector< double > kappa_ci

  *the kappa ci by duration*
- size_t m = 0

  *the number of trials run for mad*
- size_t m_ci = 0

  *the number of trials for the conf. interval*
- vector< double > sum_dev

  *sum or raw deviation by duration*
- vector< double > sum_abs_dev

  *sum of abs deviations by duration*
- vector< vector< double > > x_ci

  *the results by trial and duration*
- vector< double > conf_int

  *the calculated ci by duration*
- cpu_times elapsed_time

  *the elapsed time for the run*

### 5.3.1 Detailed Description

structure holding the results of a run for a single alpha

the struct holding the resutls of a single alpha

### 5.3.2 Constructor & Destructor Documentation

#### 5.3.2.1 KappaResult()

```
KappaResult::KappaResult (
            vector< int > ns )  [inline]
```

constructor

**Parameters**

| | |
|---|---|
| *ns* | the durations of the output |

### 5.3.3 Member Function Documentation

#### 5.3.3.1 initialize()

```
void KappaResult::initialize (
            double param_in,
            const vector< int > & ns_in )  [inline]
```

initialize the structure

**Parameters**

| | |
|---|---|
| *param↩<br>_in* | the alpha of the run |
| *ns_in* | the durations of the run |

#### 5.3.3.2 update_conf_int()

```
void KappaResult::update_conf_int (
            double ci_level )  [inline]
```

calculate the confidence interval for all trials

**Parameters**

| | |
|---|---|
| *ci_level* | the confidence level to use |

#### 5.3.3.3 update_dev()

```
void KappaResult::update_dev (
             size_t m,
             size_t m_ci,
             const vector< double > & dev,
             const vector< double > & abs_dev,
             const vector< vector< double > > & x_in )  [inline]
```

update the deviations w result from one thread

**Parameters**

| | |
|---|---|
| *m* | the number of trials for mad |
| *m_ci* | the number of trials for ci |
| *dev* | sum of deviation by duration |
| *abs_dev* | sum of abs dev by duration |
| *x_in* | the variate by trial |

### 5.3.4 Member Data Documentation

#### 5.3.4.1 ci_rel_err

```
double KappaResult::ci_rel_err
```

the kappa_mad by duration

the relative error of the ci vs theory

the relative error of conf. int. vs theory

#### 5.3.4.2 mad_rel_err

```
double KappaResult::mad_rel_err
```

the relative error of mad vs theory

the relative error of the mad vs theory

#### 5.3.4.3 ns

```
vector< int > KappaResult::ns
```

the durations saved

the durations calculated

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

- /Users/jdunn/Documents/XCode/how_much_data/convolution_test/convolution_test.cpp
- /Users/jdunn/Documents/XCode/how_much_data/monte_carlo_test/monte_carlo_test.cpp

## 5.4 KappaResults Struct Reference

structure holding the results of all runs

### Public Member Functions

- KappaResults (const vector< int > &ns, size_t n_params, string param_label, size_t taleb_offset)

  *constructor*
- KappaResult & at (size_t i)

  *return reference to a particular result*
- KappaResults (const vector< int > &ns, const vector< double > &params, const string param_label, size_t taleb_offset)

  *constructor*

### Public Attributes

- vector< int > ns

  *the durations saved*
- string param_label

  *the name of the parameter*
- vector< KappaResult > kr

  *the results of the runs by duration*
- size_t taleb_offset

  *the column offset into Taleb's table*
- mutex kr_mutex

  *a mutex for writing results*

### 5.4.1 Detailed Description

structure holding the results of all runs

sturcture holding the results from all runs

## 5.4.2 Constructor & Destructor Documentation

### 5.4.2.1 KappaResults() [1/2]

```
KappaResults::KappaResults (
             const vector< int > & ns,
             size_t n_params,
             string param_label,
             size_t taleb_offset )  [inline]
```

constructor

**Parameters**

| | |
|---|---|
| *ns* | the durations |
| *n_params* | the # of params |
| *param_label* | the param_label |
| *taleb_offset* | the column offset into Talebs table |

### 5.4.2.2 KappaResults() [2/2]

```
KappaResults::KappaResults (
             const vector< int > & ns,
             const vector< double > & params,
             const string param_label,
             size_t taleb_offset )  [inline]
```

constructor

**Parameters**

| | |
|---|---|
| *ns* | the durations calculated |
| *params* | the params for each run |
| *param_label* | the label for the param |
| *taleb_offset* | the column offset into the table of taleg's results. 0 for none |

## 5.4.3 Member Data Documentation

### 5.4.3.1 ns

```
vector< int > KappaResults::ns
```

the durations saved

the durations calculated

#### 5.4.3.2 param_label

```
string KappaResults::param_label
```

the name of the parameter

a vector holding the results of each run the label for the parameter

#### 5.4.3.3 taleb_offset

```
size_t KappaResults::taleb_offset
```

the column offset into Taleb's table

the offset into Taleb's table of results. =0 for none available.

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

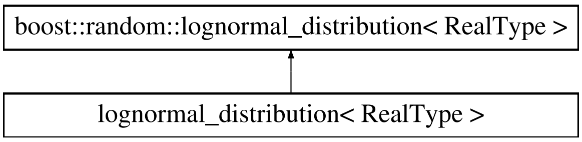
- /Users/jdunn/Documents/XCode/how_much_data/convolution_test/convolution_test.cpp
- /Users/jdunn/Documents/XCode/how_much_data/monte_carlo_test/monte_carlo_test.cpp

## 5.5 lognormal_distribution< RealType > Struct Template Reference

a class with functions related to the lognormal distribution.

```
#include <lognormal_distribution.h>
```

Inheritance diagram for lognormal_distribution< RealType >:


boost::random::lognormal_distribution< RealType >
lognormal_distribution< RealType >

## Public Member Functions

- lognormal_distribution (RealType mu, RealType sigma, int type=1)

  *the constructor for the distribution*
- template<typename Engine >
  RealType operator() (Engine &eng)

  *return a random number from the normalized distribution*
- RealType cdf (RealType x, bool lower_tail=true) const

  *return the cumulative distribution function*
- RealType pdf (RealType x) const

  *return the probability density function*
- RealType quantile (RealType p) const

  *return the quantile corresponding to a given propability*
- RealType mu () const

  *return the mu parameter of the distribution*
- RealType sigma () const

  *return sigma parameter of the distribution*
- RealType alpha_stable () const

  *return the alpha parameter of the asymptotically equivalent stable distribution*
- RealType min () const

  *Returns the smallest value that the distribution can produce.*
- RealType max () const

  *Returns the largest value that the distribution can produce.*
- RealType mean () const

  *Return mean of distribution.*
- RealType mad () const

  *Return mean absolute deviation of the distribution.*
- RealType mad2 () const

  *Return the mad of the square of the distribution.*
- RealType ci (RealType level=RealType(.05)) const

  *Return the confidence interval of the distribution.*
- complex< RealType > cf_fourier_x (RealType omega) const

  *return the characteristic function via Fourier integral in x domain*
- complex< RealType > cf_lambert_w (RealType omega) const

  *return the approximate characteristic function using Lambert W funciton Much faster than either integral but has problems when sigma > 1*
- complex< RealType > cf_fourier_lnx (RealType omega) const

  *Return the characteristic function via Fourier integral in ln(x) domain.*
- complex< RealType > characteristic_function (RealType omega) const

  *return the approximate characteristic function using precompluted cubic splines*

## Friends

- template<class charT , class traits >
  std::basic_ostream< charT, traits > & operator<< (std::basic_ostream< charT, traits > &os, const lognormal_distribution &dist)

  *Write distribution to std::ostream.*

### 5.5.1 Detailed Description

**template<class RealType = double>**
**struct lognormal_distribution< RealType >**

a class with functions related to the lognormal distribution.

Describes a random variable distributed as exp(sigma X + mu) where X is normally distributed

### 5.5.2 Constructor & Destructor Documentation

#### 5.5.2.1 lognormal_distribution()

```
template<class RealType = double>
lognormal_distribution< RealType >::lognormal_distribution (
             RealType mu,
             RealType sigma,
             int type = 1 )  [inline]
```

the constructor for the distribution

**Parameters**

| | | |
|---|---|---|
| in | *mu* | the mu parameter |
| in | *sigma* | the sigma parameter |
| in | *type* | type of cf calculaiton 1 ln(x), 2 x, 3 w, 4 spline |

### 5.5.3 Member Function Documentation

#### 5.5.3.1 cdf()

```
template<class RealType = double>
RealType lognormal_distribution< RealType >::cdf (
             RealType x,
             bool lower_tail = true ) const  [inline]
```

return the cumulative distribution function

**Parameters**

| | | |
|---|---|---|
| in | *x* | the quantile variable |
| in | *lower_tail* | flag indicating which tail to use |

### 5.5.3.2 cf_fourier_lnx()

```
template<class RealType = double>
complex<RealType> lognormal_distribution< RealType >::cf_fourier_lnx (
            RealType omega ) const  [inline]
```

Return the characteristic function via Fourier integral in ln(x) domain.

**Parameters**

| | |
|---|---|
| *omega* | the angular frequency |

### 5.5.3.3 cf_fourier_x()

```
template<class RealType = double>
complex<RealType> lognormal_distribution< RealType >::cf_fourier_x (
            RealType omega ) const  [inline]
```

return the characteristic function via Fourier integral in x domain

**Parameters**

| | |
|---|---|
| *omega* | the angular frequency |

### 5.5.3.4 cf_lambert_w()

```
template<class RealType = double>
complex<RealType> lognormal_distribution< RealType >::cf_lambert_w (
            RealType omega ) const  [inline]
```

return the approximate characteristic function using Lambert W funciton Much faster than either integral but has problems when sigma > 1

**Parameters**

| | |
|---|---|
| *omega* | the angular frequency |

### 5.5.3.5 characteristic_function()

```
template<class RealType = double>
```

```
complex<RealType> lognormal_distribution< RealType >::characteristic_function (
             RealType omega ) const  [inline]
```

return the approximate characteristic function using precomplulted cubic splines

**Parameters**

| | | |
|---|---|---|
| in | *omega* | the angular frequency |

#### 5.5.3.6 mad2()

```
template<class RealType = double>
RealType lognormal_distribution< RealType >::mad2 ( ) const  [inline]
```

Return the mad of the square of the distribution.

this is the approximation used by Taleb

#### 5.5.3.7 max()

```
template<class RealType = double>
RealType lognormal_distribution< RealType >::max ( ) const  [inline]
```

Returns the largest value that the distribution can produce.

#### 5.5.3.8 min()

```
template<class RealType = double>
RealType lognormal_distribution< RealType >::min ( ) const  [inline]
```

Returns the smallest value that the distribution can produce.

#### 5.5.3.9 pdf()

```
template<class RealType = double>
RealType lognormal_distribution< RealType >::pdf (
             RealType x ) const  [inline]
```

return the probability density function

**Parameters**

| | |
|---|---|
| *x* | the quantile variable |

#### 5.5.3.10 quantile()

```
template<class RealType = double>
RealType lognormal_distribution< RealType >::quantile (
             RealType p ) const  [inline]
```

return the quantile corresponding to a given propability

**Parameters**

| | |
|---|---|
| *p* | the target probability |

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

- /Users/jdunn/Documents/XCode/how_much_data/convolution_test/lognormal_distribution.h

## 5.6 Lower< Dist > Class Template Reference

functor for determining lower limit for target mad

### Public Member Functions

- Lower (double delta, Dist &dist)

  *constructor*
- double **operator()** (double x)

### 5.6.1 Detailed Description

**template<typename Dist>**
**class Lower< Dist >**

functor for determining lower limit for target mad

### 5.6.2 Constructor & Destructor Documentation

#### 5.6.2.1 Lower()

```
template<typename Dist >
Lower< Dist >::Lower (
             double delta,
             Dist & dist )  [inline]
```

constructor

**Parameters**

| | |
|---|---|
| *delta* | the target mad arising from lower tail |
| *dist* | the distribution |

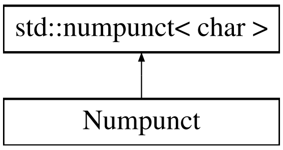
The documentation for this class was generated from the following file:

- /Users/jdunn/Documents/XCode/how_much_data/convolution_test/convolution_test.cpp

## 5.7 Numpunct Struct Reference

sturcture passed by imbue to ostreams to use commas in numbers

Inheritance diagram for Numpunct:



### Protected Member Functions

- virtual char **do_thousands_sep** () const
- virtual std::string **do_grouping** () const

### 5.7.1 Detailed Description

sturcture passed by imbue to ostreams to use commas in numbers

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

- /Users/jdunn/Documents/XCode/how_much_data/convolution_test/convolution_test.cpp

## 5.8 pareto_distribution< RealType >::param_type Class Reference

### Public Types

- typedef pareto_distribution **distribution_type**

## Public Member Functions

- param_type (RealType alpha_arg, RealType mu_arg=RealType(0.0), RealType sigma_arg=RealType(1.0))

  *Constructs the parameters of a pareto_distribution.*
- RealType alpha () const

  *Returns the "alpha" parameter of the distribution.*
- RealType mu () const

  *Returns the "mu" parameter of the distribution.*
- RealType sigma () const

  *Returns the "sigma" parameter of the distribution.*

## Friends

- template<class charT , class traits >
  std::basic_ostream< charT, traits > & operator<< (std::basic_ostream< charT, traits > &os, const param_type &parm)

  *Writes the parameters to a std::ostream.*
- template<class charT , class traits >
  std::basic_istream< charT, traits > & operator>> (std::basic_istream< charT, traits > &is, param_type &parm)

  *Reads the parameters from a std::istream.*
- bool operator== (const param_type &lhs, const param_type &rhs)

  *Returns true if the two sets of parameters are equal.*
- bool operator!= (const param_type &lhs, const param_type &rhs)

  *Returns true if the two sets of parameters are different.*

## 5.8.1 Constructor & Destructor Documentation

### 5.8.1.1 param_type()

```
template<class RealType = double>
pareto_distribution< RealType >::param_type::param_type (
            RealType alpha_arg,
            RealType mu_arg = RealType(0.0),
            RealType sigma_arg = RealType(1.0) )  [inline], [explicit]
```

Constructs the parameters of a pareto_distribution.

## 5.8.2 Member Function Documentation

#### 5.8.2.1 alpha()

```
template<class RealType = double>
RealType pareto_distribution< RealType >::param_type::alpha ( ) const  [inline]
```

Returns the "alpha" parameter of the distribution.

#### 5.8.2.2 mu()

```
template<class RealType = double>
RealType pareto_distribution< RealType >::param_type::mu ( ) const  [inline]
```

Returns the "mu" parameter of the distribution.

#### 5.8.2.3 sigma()

```
template<class RealType = double>
RealType pareto_distribution< RealType >::param_type::sigma ( ) const  [inline]
```

Returns the "sigma" parameter of the distribution.

### 5.8.3 Friends And Related Function Documentation

#### 5.8.3.1 operator"!=

```
template<class RealType = double>
bool operator!= (
             const param_type & lhs,
             const param_type & rhs )  [friend]
```

Returns true if the two sets of parameters are different.

#### 5.8.3.2 operator<<

```
template<class RealType = double>
template<class charT , class traits >
std::basic_ostream<charT,traits>& operator<< (
             std::basic_ostream< charT, traits > & os,
             const param_type & parm )  [friend]
```

Writes the parameters to a std::ostream.

#### 5.8.3.3 operator==

```
template<class RealType = double>
bool operator== (
             const param_type & lhs,
             const param_type & rhs )  [friend]
```

Returns true if the two sets of parameters are equal.

#### 5.8.3.4 operator>>

```
template<class RealType = double>
template<class charT , class traits >
std::basic_istream<charT,traits>& operator>> (
             std::basic_istream< charT, traits > & is,
             param_type & parm )  [friend]
```

Reads the parameters from a std::istream.

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

- /Users/jdunn/Documents/XCode/how_much_data/convolution_test/pareto_distribution.h

## 5.9 pareto_distribution< RealType > Class Template Reference

Instantiations of class template pareto_distribution model a Pareto Type 2 distribution.

```
#include <pareto_distribution.h>
```

### Classes

- class param_type

### Public Types

- typedef RealType **result_type**

### Public Member Functions

- pareto_distribution (RealType alpha_arg, RealType mu_arg=RealType(0.0), RealType sigma_arg=RealType(1.0))

  *Constructs a pareto_distribution.*
- pareto_distribution (const param_type &parm)

  *Constructs a pareto_distribution from its parameters.*
- RealType alpha () const

  *Returns the alpha parameter of the distribution.*
- RealType mu () const

  *Returns the mu parameter of the distribution.*
- RealType sigma () const

  *Returns the sigma parameter of the distribution.*
- RealType alpha_stable () const

  *Return the alpha of the asymptotic stable distribution.*
- RealType min () const

  *Returns the smallest value that the distribution can produce.*
- RealType max () const

  *Returns the largest value that the distribution can produce.*
- param_type param () const

  *Returns the parameters of the distribution.*
- void param (const param_type &parm)

  *Sets the parameters of the distribution.*
- RealType cdf (RealType x, bool lower_tail=true) const

  *the cdf of the distribution*
- RealType pdf (RealType x) const

  *return the pdf of the distribution*
- RealType quantile (RealType p) const

  *the quantile for a given probability*
- RealType mean () const

  *Return the mean of the distribution.*
- RealType mad () const

  *Return the MAD of the distribution.*
- RealType mad2 () const

  *Return the MAD of the square of the distribution.*
- complex< RealType > characteristic_function (RealType omega) const

  *return the characteristic function of the distribution*
- RealType ci (RealType level=RealType(.05)) const

  *Return the 95% confidence interval.*
- void reset ()

  *Effects: Subsequent uses of the distribution do not depend on values produced by any engine prior to invoking reset.*
- template<class Engine >
  result_type operator() (Engine &eng) const

  *Returns a random variate distributed according to the Pareto distribution.*
- template<class Engine >
  result_type operator() (Engine &eng, const param_type &parm)

  *Returns a random variate distributed according to the Pareto distribution with parameters specified by param.*

## Friends

- template<class charT , class traits >
  std::basic_ostream< charT, traits > & operator<< (std::basic_ostream< charT, traits > &os, const pareto_distribution &dist)

  *Write distribution to std::ostream.*
- template<class charT , class traits >
  std::basic_istream< charT, traits > & operator>> (std::basic_istream< charT, traits > &is, pareto_distribution &dist)

  *Reads the parameters from a std::istream.*
- bool operator== (const pareto_distribution &lhs, const pareto_distribution &rhs)

  *Returns true if the two distributions will produce identical sequences of values given equal generators.*
- bool operator!= (const pareto_distribution &lhs, const pareto_distribution &rhs)

  *Returns true if the two distributions may produce different sequences of values given equal generators.*

## 5.9.1 Detailed Description

**template<class RealType = double>
class pareto_distribution< RealType >**

Instantiations of class template pareto_distribution model a Pareto Type 2 distribution.

Such a distribution produces random numbers with $1 - F(x) = (1 + \frac{x-\mu}{\sigma})^{-\alpha}$ for $x > \mu$.

## 5.9.2 Constructor & Destructor Documentation

### 5.9.2.1 pareto_distribution()

```
template<class RealType = double>
pareto_distribution< RealType >::pareto_distribution (
            RealType alpha_arg,
            RealType mu_arg = RealType(0.0),
            RealType sigma_arg = RealType(1.0) )  [inline], [explicit]
```

Constructs a pareto_distribution.

`alpha` `mu` and `sigma` are the parameters of the distribution.

## 5.9.3 Member Function Documentation

#### 5.9.3.1 alpha()

```
template<class RealType = double>
RealType pareto_distribution< RealType >::alpha ( ) const  [inline]
```

Returns the alpha parameter of the distribution.

#### 5.9.3.2 max()

```
template<class RealType = double>
RealType pareto_distribution< RealType >::max ( ) const  [inline]
```

Returns the largest value that the distribution can produce.

#### 5.9.3.3 min()

```
template<class RealType = double>
RealType pareto_distribution< RealType >::min ( ) const  [inline]
```

Returns the smallest value that the distribution can produce.

#### 5.9.3.4 mu()

```
template<class RealType = double>
RealType pareto_distribution< RealType >::mu ( ) const  [inline]
```

Returns the mu parameter of the distribution.

#### 5.9.3.5 param() [1/2]

```
template<class RealType = double>
param_type pareto_distribution< RealType >::param ( ) const  [inline]
```

Returns the parameters of the distribution.

#### 5.9.3.6 param() [2/2]

```
template<class RealType = double>
void pareto_distribution< RealType >::param (
             const param_type & parm )  [inline]
```

Sets the parameters of the distribution.

#### 5.9.3.7 sigma()

```
template<class RealType = double>
RealType pareto_distribution< RealType >::sigma ( ) const  [inline]
```

Returns the sigma parameter of the distribution.

### 5.9.4 Friends And Related Function Documentation

#### 5.9.4.1 operator>>

```
template<class RealType = double>
template<class charT , class traits >
std::basic_istream<charT,traits>& operator>> (
             std::basic_istream< charT, traits > & is,
             pareto_distribution< RealType > & dist )  [friend]
```

Reads the parameters from a std::istream.

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

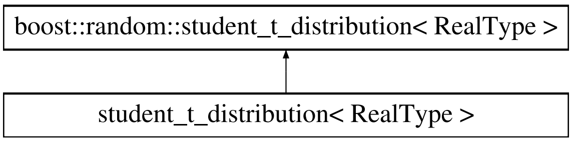
- /Users/jdunn/Documents/XCode/how_much_data/convolution_test/pareto_distribution.h

## 5.10 student_t_distribution< RealType > Struct Template Reference

Instances of class student_t_distribution give random variates for a student t distribution with parameter n = alpha.

```
#include <student_t_distribution.h>
```

Inheritance diagram for student_t_distribution< RealType >:


boost::random::student_t_distribution< RealType >
student_t_distribution< RealType >

### Public Member Functions

- student_t_distribution (RealType alpha)

  *construct an instnace give alpha*
- RealType cdf (RealType x, bool lower_tail=true) const

  *return the cdf or the complement of the cdf*
- RealType pdf (RealType x) const

  *return the probability density function at x*
- RealType quantile (RealType p) const

  *return the quantile for probability p*
- RealType alpha () const

  *return the alpha = n of the distribution*
- RealType alpha_stable () const

  *return the alpha of the asymptotic stable distribution*
- RealType mean () const

  *Return mean of distribution.*
- RealType mad () const

  *Return mean average deviation of the distribution.*
- RealType mad2 () const

  *Return the mad of the square of the distribution.*
- RealType ci (RealType level=RealType(.05)) const

  *Return the confidence interval of the distribution.*
- RealType characteristic_function (RealType omega) const

  *return the characteristic function of the distribution at omega*

### Friends

- template<class charT , class traits >
  std::basic_ostream< charT, traits > & operator<< (std::basic_ostream< charT, traits > &os, const student_t_distribution &dist)

  *Write distribution to std::ostream.*

### 5.10.1 Detailed Description

**template<class RealType = double>**
**struct student_t_distribution< RealType >**

Instances of class student_t_distribution give random variates for a student t distribution with parameter n = alpha.

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

- /Users/jdunn/Documents/XCode/how_much_data/convolution_test/student_t_distribution.h

## 5.11 Upper< Dist > Class Template Reference

functor for determining upper limit for target mad

## Public Member Functions

- Upper (double delta, Dist &dist)

  *constructor*
- double operator() (double x)

  *return excess of estimated mad in tail over target*

### 5.11.1 Detailed Description

**template<typename Dist>**
**class Upper< Dist >**

functor for determining upper limit for target mad

### 5.11.2 Constructor & Destructor Documentation

#### 5.11.2.1 Upper()

```
template<typename Dist >
Upper< Dist >::Upper (
             double delta,
             Dist & dist )   [inline]
```

constructor

**Parameters**

| | |
|---|---|
| *delta* | target mad arising from upper tail |
| *dist* | reference to distribution |

### 5.11.3 Member Function Documentation

#### 5.11.3.1 operator()()

```
template<typename Dist >
double Upper< Dist >::operator() (
             double x )   [inline]
```

return excess of estimated mad in tail over target

**Parameters**

| | |
|---|---|
| *x* | the quantile |

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

- /Users/jdunn/Documents/XCode/how_much_data/convolution_test/convolution_test.cpp

# Chapter 6

# File Documentation

## 6.1 /Users/jdunn/Documents/XCode/how_much_data/convolution_test/convolution_test.cpp File Reference

```
#include <iostream>
#include <iomanip>
#include <string>
#include <sstream>
#include <fstream>
#include <random>
#include <vector>
#include <array>
#include <algorithm>
#include <numeric>
#include <utility>
#include <mutex>
#include <thread>
#include <complex>
#include <unsupported/Eigen/FFT>
#include <boost/timer/timer.hpp>
#include <boost/filesystem.hpp>
#include <boost/math/tools/roots.hpp>
#include <boost/math/constants/constants.hpp>
#include "pareto_distribution.h"
#include "student_t_distribution.h"
#include "exponential_distribution.h"
#include "lognormal_distribution.h"
#include "taleb_results.h"
```

### Classes

- struct Numpunct

  *sturcture passed by imbue to ostreams to use commas in numbers*
- struct KappaResult

  *structure holding the results of a run for a single alpha*
- struct KappaResults

  *structure holding the results of all runs*

- class Upper< Dist >

  *functor for determining upper limit for target mad*
- class Lower< Dist >

  *functor for determining lower limit for target mad*
- struct Job

  *instances] of Job used to parcel characteristic funciion calculation to threads*

## Typedefs

- using **dcomplex** = std::complex< double >

## Functions

- template<typename RealType >
  RealType rel_err (RealType a, RealType b)

  *return the relative differnece between two numbers*
- ostream & operator<< (ostream &os, const KappaResult &k)

  *output the results to an ostream*
- ostream & operator<< (ostream &os, KappaResults &ks)

  *output the results of all runs to an ostream*
- int mod (int a, int b)

  *modulo calculation return nonnegative numer less than the modulus*
- double confidence_interval (int nmin, int nmax, double delta, const vector< double > &pdf, const vector< double > &x, double ci_level)

  *calcuate confidence interval given pdf at points in range*
- bool factor_check (int n, vector< int > primes)

  *Check whether the factorization of n contains only listed primes.*
- template<typename Dist >
  void calc_characteristic_function (const Dist &dist, const double mean, const int n, const double delta_omega, Job ∗job, vector< dcomplex > ∗adj_cf)

  *calculate the characteristic function for a range assigned to trhead*
- template<typename Dist >
  void calculate_kappa (double delta, double delta2, int m, vector< int > ns, Dist dist, double ci_level, KappaResult &k, bool verbose=false)

  *set up ranges and step sizes for one alpham pass calculation of cf to threads and use fft to estimate distribution*
- void show_usage (path p)

  *show proper usage after improper command line argument*
- int main (int argc, const char ∗argv[ ])

  *main program for convolution_testl*

## Variables

- Eigen::FFT< double > fft_eng

  *cause Eigen/FFT to use fftw. delete to avoid GPL*

### 6.1.1 Function Documentation

#### 6.1.1.1 calc_characteristic_function()

```
template<typename Dist >
void calc_characteristic_function (
             const Dist & dist,
             const double mean,
             const int n,
             const double delta_omega,
             Job * job,
             vector< dcomplex > * adj_cf )
```

calculate the characteristic function for a range assigned to trhead

**Parameters**

| | | |
|---|---|---|
| in | *dist* | the distribution |
| in | *mean* | the mean to remove |
| in | *n* | the duration |
| in | *delta_omega* | step for omega |
| in,out | *job* | ptr to job assigner |
| out | *adj_cf* | the characeristic function of the normalized distrbibution |

#### 6.1.1.2 calculate_kappa()

```
template<typename Dist >
void calculate_kappa (
             double delta,
             double delta2,
             int m,
             vector< int > ns,
             Dist dist,
             double ci_level,
             KappaResult & k,
             bool verbose = false )
```

set up ranges and step sizes for one alpham pass calculation of cf to threads and use fft to estimate distribution

**Parameters**

| | | |
|---|---|---|
| in | *delta* | the step size in x / dist.mad |
| in | *delta2* | cap on % mad from the tail |
| in | *m* | cap on maximum index |
| in | *ns* | the durations to calculate |
| in | *dist* | the distribution |
| in | *ci_level* | the confidence level for kappa_ci |
| out | *k* | the results |
| in | *verbose* | flag for trace |

#### 6.1.1.3 confidence_interval()

```
double confidence_interval (
            int nmin,
            int nmax,
            double delta,
            const vector< double > & pdf,
            const vector< double > & x,
            double ci_level )
```

calcuate confidence interval given pdf at points in range

**Parameters**

| | | |
|---|---|---|
| in | *nmin* | the minimum index |
| in | *nmax* | the maximum index |
| in | *delta* | the spacing of the x's |
| in | *pdf* | the calculated pdf's |
| in | *x* | the x for the pdf |
| in | *ci_level* | the confidence level |

#### 6.1.1.4 factor_check()

```
bool factor_check (
            int n,
            vector< int > primes )
```

Check whether the factorization of n contains only listed primes.

**Parameters**

| | |
|---|---|
| *n* | the number to check |
| *primes* | the array of candidate primes |

#### 6.1.1.5 show_usage()

```
void show_usage (
            path p )
```

show proper usage after improper command line argument

**Parameters**

| | | |
|---|---|---|
| in | *p* | the path of the executable |

## 6.2 /Users/jdunn/Documents/XCode/how_much_data/convolution_test/exponential_↩ distribution.h File Reference

```
#include <boost/random.hpp>
#include <boost/math/distributions/exponential.hpp>
```

### Classes

- struct exponential_distribution< RealType >

    *instnaces of sturct exponential_distribution generate random variates for exponential distribution*

## 6.3 /Users/jdunn/Documents/XCode/how_much_data/convolution_test/lognormal_↩ distribution.h File Reference

```
#include <iostream>
#include <iomanip>
#include <complex>
#include <string>
#include <memory>
#include <boost/random.hpp>
#include <boost/math/constants/constants.hpp>
#include <boost/math/distributions/lognormal.hpp>
#include <boost/math/special_functions/erf.hpp>
#include <boost/math/quadrature/gauss_kronrod.hpp>
#include <boost/math/quadrature/exp_sinh.hpp>
#include "p_spline.h"
```

### Classes

- struct lognormal_distribution< RealType >

    *a class with functions related to the lognormal distribution.*

## 6.4 /Users/jdunn/Documents/XCode/how_much_data/convolution_test/pareto_distribution.h File Reference

```
#include <iostream>
#include <sstream>
#include <complex>
#include <random>
#include <boost/math/quadrature/gauss_kronrod.hpp>
#include <boost/math/constants/constants.hpp>
#include <string>
```

### Classes

- class pareto_distribution< RealType >

  *Instantiations of class template pareto_distribution model a Pareto Type 2 distribution.*
- class pareto_distribution< RealType >::param_type

## 6.5 /Users/jdunn/Documents/XCode/how_much_data/convolution_test/student_t_↩ distribution.h File Reference

```
#include <boost/random.hpp>
#include <boost/math/distributions/students_t.hpp>
#include <boost/math/special_functions/beta.hpp>
#include <boost/math/special_functions/bessel.hpp>
```

### Classes

- struct student_t_distribution< RealType >

  *Instances of class student_t_distribution give random variates for a student t distribution with parameter n = alpha.*

## 6.6 /Users/jdunn/Documents/XCode/how_much_data/monte_carlo_test/monte_carlo_↩ test.cpp File Reference

```
#include <iostream>
#include <iomanip>
#include <string>
#include <sstream>
#include <fstream>
#include <random>
#include <vector>
#include <array>
#include <algorithm>
#include <numeric>
#include <mutex>
#include <thread>
#include <boost/timer/timer.hpp>
#include <boost/filesystem.hpp>
#include <boost/math/special_functions/beta.hpp>
#include <boost/math/distributions/students_t.hpp>
#include "pareto_distribution.h"
#include "student_t_distribution.h"
#include "exponential_distribution.h"
#include "lognormal_distribution.h"
#include "taleb_results.h"
```

### Classes

- struct KappaResult

  *structure holding the results of a run for a single alpha*
- struct KappaResults

  *structure holding the results of all runs*

## Functions

- template<typename RealType >
  RealType rel_err (RealType a, RealType b)

  *return the relative error between two numbers*
- template<typename RealType >
  vector< RealType > quantile (const vector< RealType > &x, const vector< RealType > &probs)

  *return quantiles of the ensemble of trials*
- ostream & operator<< (ostream &os, const KappaResult &k)

  *output the results from a single run*
- ostream & operator<< (ostream &os, KappaResults &ks)

  *output the results for all runs*
- template<typename Dist >
  void calc_kappa (unsigned int thread_id, size_t m, size_t m_ci_limit, vector< int > ns, Dist dist, double ci_↩ level, KappaResult ∗kp, bool verbose=false)

  *the per thread cacluaiton engine*
- template<typename Dist >
  void calculate_kappa (size_t m, vector< int > ns, Dist dist, double ci_level, KappaResult ∗kp, bool verbose=false)

  *calculate kappa for sums of iid variables at specified durations*
- void show_usage (path p)

  *show the usage. called when the wrong # of arguments is used*
- int main (int argc, const char ∗argv[ ])

  *main program for convolution test with one input parameter the # of trials*

## Variables

- mutex kr_mutex

  *a mutex for writing to KappaResults*
- mutex cout_mutex

  *a mutex for writing to cout*

## 6.6.1 Function Documentation

### 6.6.1.1 calc_kappa()

```
template<typename Dist >
void calc_kappa (
            unsigned int thread_id,
            size_t m,
            size_t m_ci_limit,
            vector< int > ns,
            Dist dist,
            double ci_level,
            KappaResult * kp,
            bool verbose = false )
```

the per thread cacluaiton engine

**Parameters**

| | | |
|---|---|---|
| in | *thread_id* | the number of the thread used as seed for urng |
| in | *m* | the maximum # of trials for mad |
| in | *m_ci_limit* | the maximum # of trials for ci |
| in | *ns* | the durations to save |
| in | *dist* | the distribution |
| in | *ci_level* | the confidence level for kappa_ci |
| out | *kp* | the results |
| in | *verbose* | flag for trace infomation |

#### 6.6.1.2 calculate_kappa()

```
template<typename Dist >
void calculate_kappa (
             size_t m,
             vector< int > ns,
             Dist dist,
             double ci_level,
             KappaResult * kp,
             bool verbose = false )
```

calculate kappa for sums of iid variables at specified durations

**Parameters**

| | | |
|---|---|---|
| in | *m* | the number of scenarios |
| in | *ns* | the durations to save |
| in | *dist* | the distribution |
| in | *ci_level* | the confidence level to use |
| out | *kp* | a ptr to the results |
| in | *verbose* | a flag to generate trace |

#### 6.6.1.3 operator<<()

```
ostream& operator<< (
             ostream & os,
             const KappaResult & k )
```

output the results from a single run

**Parameters**

| | | |
|---|---|---|
| in,out | *os* | the output stream |
| in | *k* | the sturct with results |

#### 6.6.1.4 quantile()

```
template<typename RealType >
vector<RealType> quantile (
             const vector< RealType > & x,
             const vector< RealType > & probs )
```

return quantiles of the ensemble of trials

**Parameters**

| | | |
|---|---|---|
| in | *x* | the result by trial |
| in | *probs* | the desired probabilities |

#### 6.6.1.5 rel_err()

```
template<typename RealType >
RealType rel_err (
             RealType a,
             RealType b )
```

return the relative error between two numbers

**Parameters**

| | | |
|---|---|---|
| in | *a* | the first number |
| in | *b* | the second number |

# Index