HowMuchData

0.8.0

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 progress, as there are some items that I can't closely replicate. However, none of the differences are large enough to call into question Taleb's results.

What's included

I've experimented with three algorithms implemented in C++, one of which relies on monte carlo simulations, a second, which relies on the use of the discrete fourier transform of the characteristic function of the convolution of the distribution functions, and finally one that takes advantage of an integral representation of the MAD given the derivative of characteristic function of the underlying distribution.

The package includes several files developed for this project:

- convolution_test.cpp. The main program to nun the convolution test, which will be retired in the next iteration. The pinelis_taleb integral is vastly superior in terms of speed and accuracy.

- monte_carlo_test.cpp. The main program to run the monte carlo test. It's bullet-proof but so slow that it's hard to obtain the required accuracy in a reasonable time frame even using multi-threading.
- pinelis_taleb_test.cpp. The main program implementing the integral representation of MAD. This is my current method of choice.
- 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 asymptotic series for small angular frequency and one based on a approximation using Lambert W functions.

In addition, i've included the apparatus I developed for adaptive integration, which seems to work much better that the Boost version that was originally used. The code is spread through the following files:

- adaptive_integration.h. The file that defines the interface to the adaptive integration routines.
- adaptive_integration_impl.h. Most of the actual implementation.
- gauss_kronrad.h. Interface to the functions used to calculate the nodes and weights for the integration.
- gauss_kronrod_impl.h. The implementing code for gauss_kronrod.
- myfloat.h. Some basic definitions allowing various types of multi-precision numbers. Multiprecision is not used in the current implementation of how_much_data.

Finally included is a test program.

- lognormal_test.cpp. A program to test the various versions of the calculation of the characteristic function. .

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.

Update Feb. 15, 2019

I noticed that Taleb covered the same topic in N. N. Taleb,`Statistical Consequences of Fat Tails`. The procedure outlined in Section 6.5 of that monograph using results of Pinelis is of general applicability and is incorporated into pinelis_taleb_test.

Further Documentation

The code has been documented using the doxygen system. The result can be accessed in the doc subdirectory of the output directory. The html version is accessed through html/index.html

Observations

So far my results are close to Taleb's. 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. The pinelis_taleb integral bypasses most of these problems, but it requires a very accurate calculation of the characteristic function and its derivative and some ingenuity in choosing the right contour for integration.

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.

Relationship with the Stable Distribution

Almost all of the distributions modeled have an associated stable distribution that is asymptotically equivalent to the modeled distribution. The kappa for stable distributions is always 2 - alpha. However, in most cases the MAD for the associated stable distribution is significantly different from the MAD for the modelled distribution and the divergence of kappa from 2 - alpha is associated with the rate of convergence of the MAD to the MAD of the associated stable distribution.

Lognormal Distribution

As Taleb mentions in his paper, the lognormal distribution is a well behaved thin tail distriubtion for small sigma, but becomes a fat-tailed distribution for sigmas much in excess of one. The weakest part of the calculation here is associated with the large sigma runs of the for the convolution test of the lognormal distribution. These have three widely divergent scales associated with them. For instance, when mu=0 and sigma=5, the mode is about 1.4e-11, the mean and MAD are about 2.7e+5 and the stardard deviation is about 7.2e+10. The MAD for the normal distribution is sqrt(2/pi) times it's standard deviation and therefore the the MAD normallized by n^.5 must go from 2.7e+5 to 5.7e+10 to reach it's asymptotic state.

To Do

- Implement the normal with switching variance distribution. This is the one that Taleb uses demonstrate the possibility of negative kappa.

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 holding the large arrays that are used by the fft. These are available at Eigen.
3. As distributed the Eigen header wraps the fftw3 available at fftw. 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_test.cpp, which will revert to the kissfft. Warning: kissfft uses much more memory and causes a severe slow down on my machine for the larger arrays because of swapping activiity.
4. One of the calculations 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.

5. The routines in the adaptive_integration routine started out life as machine C++ translations of Fortran routines in QUADPACK, which is part of SLATEC and therefore in the public domain (http://en.wikipedia.org/wiki/QUADPACK). The routines were then heavily modified to take advantage of the C++ language.

6. One of the modifications made to QUADPACK is the addition of the ability to calculate the nodes and weights for the Gauss Kronrod integration on the fly. For this purpose Dirk Laurie's method is used kronrod.ps. The routines used here are C++ translations of Dirk Laurie's MATLAB code, which is included in Walter Gautschi's OPQ suite OPQ.

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

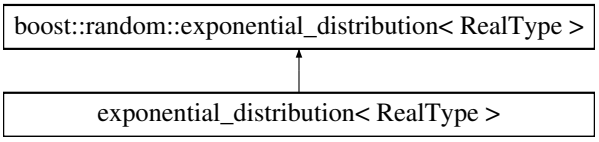
Class Documentation

5.1 exponential_distribution< RealType > Struct Template Reference

instances of struct exponential_distribution generate random variates for exponential distribution

`#include <exponential_distribution.h>`

Inheritance diagram for 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 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 > characteristic_function (RealType omega) const

  return the characteristic function given real omega
- complex< RealType > characteristic_function (complex< RealType > omega) const

  return the characteristic function given complex omega
- complex< RealType > characteristic_function_prime (RealType omega) const

  return the derivative of characteristic function given real omega
- complex< RealType > characteristic_function_prime (complex< RealType > omega) const

  return the derivative of characteristic function given complex 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 >

instances of struct exponential_distribution generate random variates for exponential distribution

Author

Created by Joseph Dunn on 1/3/19.

Copyright

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

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

5.2 KappaResult Struct Reference

the struct holding the resutls of a single alpha

Public Member Functions

- 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
- KappaResult (vector< int > ns)

  constructor
- void calc_kappa ()

  the kappa_mad by duration

Public Attributes

- double param

  the parameter for the run
- double mad_rel_err

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

  the durations calculated
- 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
- double ci_rel_err

  the relative error of the ci vs theory
- vector< double > conf_int

  the calculated ci by duration
- cpu_times elapsed_time

  the elapsed time for the run
- vector< double > mad

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

5.2.1 Detailed Description

the struct holding the resutls of a single alpha

structure holding the results of a run for a single parameter value

5.2.2 Constructor & Destructor Documentation

5.2.2.1 KappaResult()

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

constructor

Parameters

ns	the durations of the output

5.2.3 Member Function Documentation

5.2.3.1 calc_kappa()

```
void KappaResult::calc_kappa ( )  [inline]
```

the kappa_mad by duration

calculate the kappa from the mad and ci variables

5.2.3.2 initialize()

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

initialize the structure

Parameters

param_in	the alpha of the run
ns_in	the durations of the run

5.2.3.3 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

5.2.3.4 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

5.2.4 Member Data Documentation

5.2.4.1 mad_rel_err

```
double KappaResult::mad_rel_err
```

the relative error of the mad vs theory

the relative error of mad vs theory

5.2.4.2 ns

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

the durations calculated

the durations saved

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

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

5.3 KappaResults Struct Reference

sturcture holding the results from all runs

Public Member Functions

- KappaResults (const vector< int > &ns, const vector< double > &params, const string param_label, size_t taleb_offset)
  constructor
- 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
- void **dump_results** (ostream &os, string dist_name)

Public Attributes

- vector< int > ns
  the durations calculated
- string param_label
  a vector holding the results of each run the label for the parameter
- vector< KappaResult > kr
  the results of the runs by duration
- size_t taleb_offset
  the offset into Taleb's table of results. =0 for none available.
- mutex kr_mutex
  a mutex for writing results

5.3.1 Detailed Description

sturcture holding the results from all runs

structure holding the results of all parameter values

5.3.2 Constructor & Destructor Documentation

5.3.2.1 KappaResults() [1/2]

```
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

5.3.2.2 KappaResults() [2/2]

```
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

5.3.3 Member Data Documentation

5.3.3.1 ns

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

the durations calculated

the durations saved

5.3.3.2 param_label

```
string KappaResults::param_label
```

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

the name of the parameter

5.3.3.3 taleb_offset

```
size_t KappaResults::taleb_offset
```

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

the column offset into Taleb's table

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

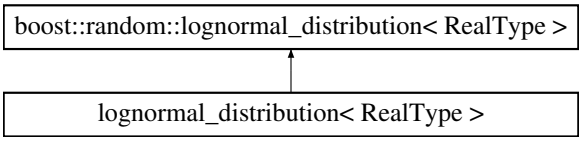
- /Users/jdunn/Documents/XCode/how_much_data/monte_carlo_test/monte_carlo_test.cpp
- /Users/jdunn/Documents/XCode/how_much_data/pinelis_taleb_test/pinelis_taleb_test.cpp

5.4 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 >:



Public Member Functions

- lognormal_distribution (RealType mu, RealType sigma, IntegrationController< RealType > &cf_ctl, 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_series (complex< RealType > omega, RealType *_error=nullptr, RealType *_l1_norm=nullptr, int *_neval=nullptr)

  cf via asymptotic series for small omega

- complex< RealType > cfprime_series (complex< RealType > omega, RealType *_error=nullptr, RealType *_l1_norm=nullptr, int *_neval=nullptr)

  derivative of cf via asymptotic series for small omega

- complex< RealType > cf_lambert_w (complex< RealType > omega, RealType *_error=nullptr, RealType *_l1_norm=nullptr, int *_neval=nullptr)

  Much faster than the fourier integrals but has problems when sigma > 1.

- complex< RealType > cfprime_lambert_w (complex< RealType > omega, RealType *_error=nullptr, RealType *_l1_norm=nullptr, int *_neval=nullptr)

  return the approximate derivative of char.

- complex< RealType > cf_fourier (complex< RealType > omega, RealType *_error=nullptr, RealType *_l1_norm=nullptr, int *_neval=nullptr)

  Return the characteristic function along a designer countour.

- complex< RealType > cfprime_fourier (complex< RealType > omega, RealType *_error=nullptr, RealType *_l1_norm=nullptr, int *_neval=nullptr)

  Return the derivative of the char. fun. wrt omega using designer contour.

- complex< RealType > cf_fourier_lnx (complex< RealType > omega, RealType *_error=nullptr, RealType *_l1_norm=nullptr, int *_neval=nullptr)

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

- complex< RealType > cfprime_fourier_lnx (complex< RealType > omega, RealType *_error=nullptr, RealType *_l1_norm=nullptr, int *_neval=nullptr)

  Return the derivative of char. function via Fourier integral in ln(x) domain.

- complex< RealType > cf_fourier_mixed (complex< RealType > omega, RealType *_error=nullptr, RealType *_l1_norm=nullptr, int *_neval=nullptr)

  Return the char. fun. using mixed Lambert and shifted contour.

- complex< RealType > cfprime_fourier_mixed (complex< RealType > omega, RealType *_error=nullptr, RealType *_l1_norm=nullptr, int *_neval=nullptr)

  The derivative of the char. fun.

- complex< RealType > characteristic_function (complex< RealType > omega, RealType *_error=nullptr, RealType *_l1_norm=nullptr, int *_neval=nullptr)

  return the approximate characteristic function

- complex< RealType > characteristic_function_prime (complex< RealType > omega, RealType *_error=nullptr, RealType *_l1_norm=nullptr, int *_neval=nullptr)

  return the derivative of the characteristic function

Static Public Member Functions

- static void print_fourier_integrand (ostream &os, int n, complex< RealType > omega, RealType sigma)

  print integrand of CFIntegrand

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.

5.4.1 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) adjusted to zero mean where X is normally distributed.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 lognormal_distribution()

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

the constructor for the distribution

Parameters

in	*mu*	the mu parameter
in	*sigma*	the sigma parameter
	cf_ctl	[in} reference controller
in	*type*	type of cf calculaiton 1 ln(x), 2 bespoke, 3 w, 4 mixed

5.4.3 Member Function Documentation

5.4.3.1 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

5.4.3.2 cf_fourier()

```
template<class RealType = double>
complex<RealType> lognormal_distribution< RealType >::cf_fourier (
            complex< RealType > omega,
            RealType * _error = nullptr,
            RealType * _l1_norm = nullptr,
            int * _neval = nullptr )  [inline]
```

Return the characteristic function along a designer countour.

Parameters

in	*omega*	the angular frequency
out	*_error*	the integration error
out	*_l1_norm*	the l1 norm
out	*_neval*	the # of evaluations

5.4.3.3 cf_fourier_lnx()

```
template<class RealType = double>
complex<RealType> lognormal_distribution< RealType >::cf_fourier_lnx (
            complex< RealType > omega,
            RealType * _error = nullptr,
            RealType * _l1_norm = nullptr,
            int * _neval = nullptr )  [inline]
```

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

Parameters

	omega	the angular frequency
out	*_error*	the integration error
out	*_l1_norm*	the
out	*_neval*	the # of evaluations

5.4.3.4 cf_fourier_mixed()

```
template<class RealType = double>
complex<RealType> lognormal_distribution< RealType >::cf_fourier_mixed (
            complex< RealType > omega,
            RealType * _error = nullptr,
            RealType * _l1_norm = nullptr,
            int * _neval = nullptr )  [inline]
```

Return the char. fun. using mixed Lambert and shifted contour.

Parameters

in	*omega*	the angular frequency
out	*_error*	the integration error
out	*_l1_norm*	the l1 norm
out	*_neval*	the # of evaluations

5.4.3.5 cf_lambert_w()

```
template<class RealType = double>
complex<RealType> lognormal_distribution< RealType >::cf_lambert_w (
            complex< RealType > omega,
            RealType * _error = nullptr,
            RealType * _l1_norm = nullptr,
            int * _neval = nullptr )  [inline]
```

Much faster than the fourier integrals but has problems when sigma > 1.

Parameters

in	*omega*	the angular frequency
out	*_error*	the integration error
out	*_l1_norm*	the l1 norm
out	*_neval*	the # of evaluations

5.4.3.6 cf_series()

```
template<class RealType = double>
complex<RealType> lognormal_distribution< RealType >::cf_series (
            complex< RealType > omega,
            RealType * _error = nullptr,
```

```
            RealType * _l1_norm = nullptr,
            int * _neval = nullptr )  [inline]
```

cf via asymptotic series for small omega

Parameters

in	*omega*	the angular frequency
out	*_error*	the integration error
out	*_l1_norm*	the l1 norm
out	*_neval*	the # of evaluations

5.4.3.7 cfprime_fourier()

```
template<class RealType = double>
complex<RealType> lognormal_distribution< RealType >::cfprime_fourier (
            complex< RealType > omega,
            RealType * _error = nullptr,
            RealType * _l1_norm = nullptr,
            int * _neval = nullptr )  [inline]
```

Return the derivative of the char. fun. wrt omega using designer contour.

Parameters

	omega	the angular frequency
out	*_error*	the integration error
out	*_l1_norm*	the
	_neval	the number of evaluations

5.4.3.8 cfprime_fourier_lnx()

```
template<class RealType = double>
complex<RealType> lognormal_distribution< RealType >::cfprime_fourier_lnx (
            complex< RealType > omega,
            RealType * _error = nullptr,
            RealType * _l1_norm = nullptr,
            int * _neval = nullptr )  [inline]
```

Return the derivative of char. function via Fourier integral in ln(x) domain.

Parameters

in	*omega*	the angular frequency
out	*_error*	the integration error
out	*_l1_norm*	the l1 norm
out	*_neval*	the # of evaluations

5.4.3.9 cfprime_fourier_mixed()

```
template<class RealType = double>
complex<RealType> lognormal_distribution< RealType >::cfprime_fourier_mixed (
            complex< RealType > omega,
            RealType * _error = nullptr,
            RealType * _l1_norm = nullptr,
            int * _neval = nullptr )  [inline]
```

The derivative of the char. fun.

Parameters

in	*omega*	the angular frequency
out	*_error*	the integration error
out	*_l1_norm*	the l1 norm
out	*_neval*	the # of evaluations

5.4.3.10 cfprime_lambert_w()

```
template<class RealType = double>
complex<RealType> lognormal_distribution< RealType >::cfprime_lambert_w (
            complex< RealType > omega,
            RealType * _error = nullptr,
            RealType * _l1_norm = nullptr,
            int * _neval = nullptr )  [inline]
```

return the approximate derivative of char.

function using Lambert W funciton Much faster than the fourier integrals but has problems when sigma $>$ 1

Parameters

in	*omega*	the angular frequency
out	*_error*	the integration error
out	*_l1_norm*	the l1 norm
out	*_neval*	the # of evaluations

5.4.3.11 cfprime_series()

```
template<class RealType = double>
complex<RealType> lognormal_distribution< RealType >::cfprime_series (
            complex< RealType > omega,
```

```
            RealType * _error = nullptr,
            RealType * _l1_norm = nullptr,
            int * _neval = nullptr )  [inline]
```

derivative of cf via asymptotic series for small omega

Parameters

in	*omega*	the angular frequency
out	*_error*	the integration error
out	*_l1_norm*	the l1 norm
out	*_neval*	the # of evaluations

5.4.3.12 characteristic_function()

```
template<class RealType = double>
complex<RealType> lognormal_distribution< RealType >::characteristic_function (
            complex< RealType > omega,
            RealType * _error = nullptr,
            RealType * _l1_norm = nullptr,
            int * _neval = nullptr )  [inline]
```

return the approximate characteristic function

Parameters

in	*omega*	the angular frequency
out	*_error*	the integration error
out	*_l1_norm*	the l1 norm
	_neval	{out} the # of evaluations

5.4.3.13 characteristic_function_prime()

```
template<class RealType = double>
complex<RealType> lognormal_distribution< RealType >::characteristic_function_prime (
            complex< RealType > omega,
            RealType * _error = nullptr,
            RealType * _l1_norm = nullptr,
            int * _neval = nullptr )  [inline]
```

return the derivative of the characteristic function

Parameters

in	*omega*	the angular frequency
out	*_error*	the integration error
out	*_l1_norm*	the
out	*_neval*	the # of evaluations

5.4.3.14 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

5.4.3.15 max()

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

Returns the largest value that the distribution can produce.

5.4.3.16 min()

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

Returns the smallest value that the distribution can produce.

5.4.3.17 pdf()

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

return the probability density function

Parameters

x	the quantile variable

5.4.3.18 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/include/lognormal_distribution.h

5.5 normal_switch_mean< RealType > Class Template Reference

class whose instances represent a 50/50 mixture of normal distriubtions with sigma = 1 and means +d and -d

`#include <normal_switch_mean.h>`

Public Member Functions

- normal_switch_mean (RealType d)
- template<typename Engine >
  RealType operator() (Engine &eng)
  return a random number from the normalized distribution
- RealType **min** () const
- RealType **max** () const
- RealType d () const
  return the half difference betwee the means
- RealType cdf (RealType x, bool lower_tail=true) const
  return the cdf
- RealType pdf (RealType x) const
  return the pdf
- RealType quantile (RealType p, bool lower_tail=true) const
  return the quantile given the probability p
- RealType alpha_stable () const
  return the alpha parameter of the asymptotically equivalent stable distribution
- complex< RealType > characteristic_function (RealType omega) const
  return the char. fun. for real arguments
- complex< RealType > characteristic_function_prime (RealType omega) const
  return the derivative of the char. fun. for real arguments
- RealType **mean** () const
- RealType **mad** () const
- RealType **mad2** () const
- RealType ci (RealType level=RealType(.05)) const
  Return the confidence interval of the distribution.

Friends

- ostream & operator<< (ostream &os, normal_switch_mean &dist)
  Write distribution to std::ostream.

5.5.1 Detailed Description

template<typename RealType = double>
class normal_switch_mean< RealType >

class whose instances represent a 50/50 mixture of normal distriubtions with sigma = 1 and means +d and -d

5.5.2 Constructor & Destructor Documentation

5.5.2.1 normal_switch_mean()

```
template<typename RealType = double>
normal_switch_mean< RealType >::normal_switch_mean (
            RealType d )  [inline]
```

Parameters

in	*d*	the means are +- d from 0

5.5.3 Member Function Documentation

5.5.3.1 cdf()

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

return the cdf

Parameters

in	*x*	the desired quantile
in	*lower_tail*	flag for tail

5.5.3.2 pdf()

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

return the pdf

Parameters

in	*x*	the desired quantile

5.5.3.3 quantile()

```
template<typename RealType = double>
RealType normal_switch_mean< RealType >::quantile (
            RealType p,
            bool lower_tail = true ) const  [inline]
```

return the quantile given the probability p

Parameters

in	*p*	the disired probability
in	*lower_tail*	flag for tail

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

- /Users/jdunn/Documents/XCode/how_much_data/include/normal_switch_mean.h

5.6 normal_switch_stddev< RealType > Class Template Reference

class whose instances represent a mixture of normal distriubtions with mu=0 and stddev of 1 w prob.

`#include <normal_switch_stddev.h>`

Public Member Functions

- normal_switch_stddev (RealType a, RealType p)
- template<typename Engine >
  RealType operator() (Engine &eng)
  return a random number from the normalized distribution
- RealType **min** () const
- RealType **max** () const
- RealType a () const
  return the second stddev
- RealType p () const
  return the prob. of a switch to a
- RealType cdf (RealType x, bool lower_tail=true) const
  return the cdf
- RealType pdf (RealType x) const

return the pdf

- RealType quantile (RealType pp, bool lower_tail=true) const

  return the quantile given the probability p

- RealType alpha_stable () const

  return the alpha parameter of the asymptotically equivalent stable distribution

- complex< RealType > characteristic_function (RealType omega) const

  return the char. fun. for real arguments

- complex< RealType > characteristic_function_prime (RealType omega) const

  return the derivative of the char. fun. for real arguments

- RealType **mean** () const
- RealType **mad** () const
- RealType **mad2** () const
- RealType ci (RealType level=RealType(.05)) const

  Return the confidence interval of the distribution.

Friends

- ostream & operator<< (ostream &os, normal_switch_stddev &dist)

  Write distribution to std::ostream.

5.6.1 Detailed Description

template<typename RealType = double>
class normal_switch_stddev< RealType >

class whose instances represent a mixture of normal distriubtions with mu=0 and stddev of 1 w prob.

(1-p) and a with prob. p

5.6.2 Constructor & Destructor Documentation

5.6.2.1 normal_switch_stddev()

```
template<typename RealType = double>
normal_switch_stddev< RealType >::normal_switch_stddev (
            RealType a,
            RealType p )  [inline]
```

Parameters

in	*a*	the means are +- d from 0
in	*p*	the prob. of sigma=a

5.6.3 Member Function Documentation

5.6.3.1 cdf()

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

return the cdf

Parameters

in	*x*	the desired quantile
in	*lower_tail*	flag for tail

5.6.3.2 pdf()

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

return the pdf

Parameters

in	*x*	the desired quantile

5.6.3.3 quantile()

```
template<typename RealType = double>
RealType normal_switch_stddev< RealType >::quantile (
            RealType pp,
            bool lower_tail = true ) const  [inline]
```

return the quantile given the probability p

Parameters

in	*pp*	the disired probability
in	*lower_tail*	flag for tail

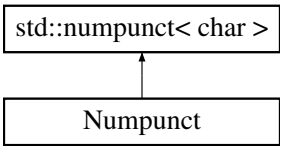
The documentation for this class was generated from the following file:

- /Users/jdunn/Documents/XCode/how_much_data/include/normal_switch_stddev.h

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/pinelis_taleb_test/pinelis_taleb_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/include/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

- complex< RealType > characteristic_function (complex< RealType > omega) const

  return the characteristic function of the distribution given complex argument

- complex< RealType > characteristic_function_prime (RealType omega) const

  return the derivative of th characteristic function of the distribution

- complex< RealType > characteristic_function_prime (complex< RealType > omega) const

  return the derivative of th char. fnct. of the dist. given complex arg.

- 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/include/pareto_distribution.h

5.10 PinelisTalebIntegrand< Dist > Class Template Reference

Functor used in the integratoin of the Pinelis Taleb result.

Public Member Functions

- PinelisTalebIntegrand (Dist &dist, const int n)
  construct the integrand
- double operator() (double x)
  return the value of the Pinelis Taleb integrand
- void **operator()** (std::vector< double > &xs)

5.10.1 Detailed Description

template<typename Dist>
class PinelisTalebIntegrand< Dist >

Functor used in the integratoin of the Pinelis Taleb result.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 PinelisTalebIntegrand()

```
template<typename Dist >
PinelisTalebIntegrand< Dist >::PinelisTalebIntegrand (
             Dist & dist,
             const int n )  [inline]
```

construct the integrand

Parameters

in	*dist*	the distribution
in	*n*	the duration

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

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

5.11 PinelisTalebIntegrand< exponential_distribution<> > Class Template Reference

Functor used in the integration of the Pinelis Taleb result for exp. dist.

Public Member Functions

- PinelisTalebIntegrand (const exponential_distribution<> &dist, const int n)
  construct the integrand
- double operator() (double x) const
  return the value of the Pinelis Taleb integrand for exp distribution Uses an adjusted contour of integration
- void **operator()** (std::vector< double > &xs) const

5.11.1 Detailed Description

template<>
class PinelisTalebIntegrand< exponential_distribution<> >

Functor used in the integration of the Pinelis Taleb result for exp. dist.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 PinelisTalebIntegrand()

```
PinelisTalebIntegrand< exponential_distribution<> >::PinelisTalebIntegrand (
            const exponential_distribution<> & dist,
            const int n )  [inline]
```

construct the integrand

Parameters

in	*dist*	the distribution
in	*n*	the duration

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

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

5.12 PinelisTalebIntegrand< pareto_distribution<> > Class Template Reference

Functor used in the integration of the Pinelis Taleb result for pareto dist.

Public Member Functions

- PinelisTalebIntegrand (pareto_distribution<> &dist, const int n)
  construct the integrand
- double operator() (double x) const
  return the value of the Pinelis Taleb integrand for pareto distribution Uses an adjusted contour of integration
- void **operator()** (std::vector< double > &xs) const

5.12.1 Detailed Description

template<>
class PinelisTalebIntegrand< pareto_distribution<> >

Functor used in the integration of the Pinelis Taleb result for pareto dist.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 PinelisTalebIntegrand()

```
PinelisTalebIntegrand< pareto_distribution<> >::PinelisTalebIntegrand (
            pareto_distribution<> & dist,
            const int n )  [inline]
```

construct the integrand

Parameters

in	*dist*	the distribution
in	*n*	the duration

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

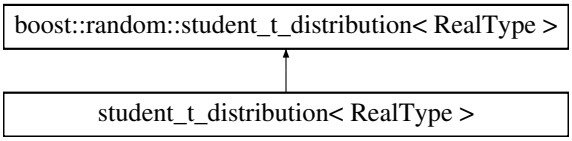
- /Users/jdunn/Documents/XCode/how_much_data/pinelis_taleb_test/pinelis_taleb_test.cpp

5.13 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 >:



Public Member Functions

- student_t_distribution (RealType alpha)
  construct an instance 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 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.
- RealType characteristic_function (RealType omega) const
  return the characteristic function of the distribution at omega
- RealType characteristic_function_prime (RealType omega) const
  return the derivative of the characteristic function 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.

5.13.1 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/include/student_t_distribution.h

Chapter 6

File Documentation

6.1 /Users/jdunn/Documents/XCode/how_much_data/include/exponential_distribution.h File Reference

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

Classes

- struct exponential_distribution< RealType >

  instances of struct exponential_distribution generate random variates for exponential distribution

6.2 /Users/jdunn/Documents/XCode/how_much_data/include/lognormal_distribution.h File Reference

```
#include <iostream>
#include <iomanip>
#include <complex>
#include <string>
#include <memory>
#include <utility>
#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/tools/roots.hpp>
#include "adaptive_integration.h"
#include "lambert_w.h"
```

Classes

- struct lognormal_distribution< RealType >

  a class with functions related to the lognormal distribution.

Functions

- template<typename RealType >
  complex< RealType > **expm1** (complex< RealType > z)
- template<typename RealType >
  RealType **sign** (RealType x)

6.3 /Users/jdunn/Documents/XCode/how_much_data/include/normal_switch_mean.h File Reference

```
#include <iostream>
#include <iomanip>
#include <complex>
#include <string>
#include <utility>
#include <boost/random.hpp>
#include <boost/random/bernoulli_distribution.hpp>
#include <boost/random/normal_distribution.hpp>
#include <boost/math/distributions/normal.hpp>
#include <boost/math/constants/constants.hpp>
#include <boost/math/tools/roots.hpp>
```

Classes

- class normal_switch_mean< RealType >

  class whose instances represent a 50/50 mixture of normal distriubtions with sigma = 1 and means +d and -d

6.4 /Users/jdunn/Documents/XCode/how_much_data/include/normal_switch_stddev.h File Reference

```
#include <iostream>
#include <iomanip>
#include <complex>
#include <string>
#include <utility>
#include <boost/random.hpp>
#include <boost/random/bernoulli_distribution.hpp>
#include <boost/random/normal_distribution.hpp>
#include <boost/math/distributions/normal.hpp>
#include <boost/math/constants/constants.hpp>
#include <boost/math/tools/roots.hpp>
```

Classes

- class normal_switch_stddev< RealType >

  class whose instances represent a mixture of normal distriubtions with mu=0 and stddev of 1 w prob.

6.5 /Users/jdunn/Documents/XCode/how_much_data/include/pareto_distribution.h File Reference

```
#include <iostream>
#include <sstream>
#include <complex>
#include <random>
#include "adaptive_integration.h"
```

Classes

- class pareto_distribution< RealType >

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

6.6 /Users/jdunn/Documents/XCode/how_much_data/include/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.

Functions

- template<typename T >
  int sign (T val)

  returns sign of its argument

6.7 /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 "adaptive_integration.h"
#include "pareto_distribution.h"
#include "student_t_distribution.h"
#include "exponential_distribution.h"
#include "lognormal_distribution.h"
#include "normal_switch_mean.h"
#include "normal_switch_stddev.h"
#include "taleb_results.h"
```

Classes

- struct KappaResult

  the struct holding the resutls of a single alpha
- struct KappaResults

  sturcture holding the results from all runs

Functions

- Kronrod< double > **k_big** (10)
- 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

- int **noext** = 0
- double **epsabs_double** = 0
- double **epsrel_double** = 64 * std::numeric_limits<double>::epsilon()
- int **limit** = 2000
- int **verbose_integration** = 0
- mutex kr_mutex

  a mutex for writing to KappaResults
- mutex cout_mutex

  a mutex for writing to cout

6.7.1 Function Documentation

6.7.1.1 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

6.7.1.2 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

6.7.1.3 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

6.7.1.4 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

6.7.1.5 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

6.8 /Users/jdunn/Documents/XCode/how_much_data/pinelis_taleb_test/pinelis_taleb_test.cpp File Reference

```
#include <iostream>
#include <iomanip>
#include <string>
#include <sstream>
#include <fstream>
#include <vector>
#include <array>
#include <algorithm>
#include <mutex>
#include <thread>
#include <complex>
#include <boost/timer/timer.hpp>
#include <boost/filesystem.hpp>
#include <boost/math/constants/constants.hpp>
#include "adaptive_integration.h"
#include "pareto_distribution.h"
#include "student_t_distribution.h"
#include "exponential_distribution.h"
#include "lognormal_distribution.h"
#include "normal_switch_mean.h"
#include "normal_switch_stddev.h"
#include "taleb_results.h"
```

Classes

- struct Numpunct

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

  the struct holding the resutls of a single alpha
- struct KappaResults

  sturcture holding the results from all runs
- class PinelisTalebIntegrand< Dist >

  Functor used in the integratoin of the Pinelis Taleb result.

- class PinelisTalebIntegrand< pareto_distribution<> >
  Functor used in the integration of the Pinelis Taleb result for pareto dist.
- class PinelisTalebIntegrand< exponential_distribution<> >
  Functor used in the integration of the Pinelis Taleb result for exp. dist.

Typedefs

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

Functions

- Kronrod< double > **k_big** (10)
- 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 result for one parameter value to an ostream
- ostream & operator<< (ostream &os, KappaResults &ks)
  output the results for all parameter values to an ostream
- template<typename Dist >
  void calculate_kappa (vector< int > ns, Dist dist, KappaResult &k, int verbose=0)
  set up integrand and calculate the results for one parameter value
- void show_usage (path p)
  show proper usage after improper command line argument
- int main (int argc, const char *argv[ ])
  main program for pinelis-taleb run

Variables

- int **noext** = 0
- double **epsabs_double** = 0
- double **epsrel_double** = 64 * std::numeric_limits<double>::epsilon()
- int **limit** = 2000
- int **verbose_integration** = 0

6.8.1 Function Documentation

6.8.1.1 calculate_kappa()

```
template<typename Dist >
void calculate_kappa (
            vector< int > ns,
            Dist dist,
            KappaResult & k,
            int verbose = 0 )
```

set up integrand and calculate the results for one parameter value

Parameters

in	*ns*	the durations to calculate
in	*dist*	the distribution
out	*k*	the results
in	*verbose*	flag for trace

6.8.1.2 show_usage()

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

show proper usage after improper command line argument

Parameters

in	*p*	the path of the executable

Index