NAME

RandomLM< derivedRandomLM, copulaPolicy, USNG >

SYNOPSIS

`#include <ql/experimental/credit/randomdefaultlatentmodel.hpp>`

Inherits **LazyObject**, and **DefaultLossModel**.

Protected Member Functions

RandomLM (**Size** numFactors, **Size** numLMVars, const copulaPolicy &copula, **Size** nSims, **BigNatural** seed)
void **update** ()
void **performCalculations** () const
void **performSimulations** () const
const std::vector< **simEvent**< derivedRandomLM< copulaPolicy, USNG > > > & **getSim** (const **Size** iSim) const
Real getEventRecovery (const **simEvent**< derivedRandomLM< copulaPolicy, USNG > > &evt) const

Statistics, DefaultLossModel interface.

virtual **Probability probAtLeastNEvents** (**Size** n, const **Date** &d) const
virtual **Disposable**< std::vector< **Probability** > > **probsBeingNthEvent** (**Size** n, const **Date** &d) const
virtual **Real defaultCorrelation** (const **Date** &d, **Size** iName, **Size** jName) const
Pearsons' default probability correlation.
virtual **Real expectedTrancheLoss** (const **Date** &d) const
virtual std::pair< **Real**, **Real** > **expectedTrancheLossInterval** (const **Date** &d, **Probability** confidencePerc) const
virtual **Disposable**< std::map< **Real**, **Probability** > > **lossDistribution** (const **Date** &d) const
Full loss distribution.
virtual **Histogram computeHistogram** (const **Date** &d) const
virtual **Real expectedShortfall** (const **Date** &d, **Real** percent) const
Expected shortfall given a default loss percentile.
virtual **Real percentile** (const **Date** &d, **Real** percentile) const
Value at Risk given a default loss percentile.
virtual boost::tuples::tuple< **Real**, **Real**, **Real** > **percentileAndInterval** (const **Date** &d, **Real percentile**) const
virtual **Disposable**< std::vector< **Real** > > **splitVaRLevel** (const **Date** &date, **Real** loss) const
virtual **Disposable**< std::vector< std::vector< **Real** > > > **splitVaRAndError** (const **Date** &date, **Real** loss, **Probability** confInterval) const

Protected Attributes

const **Size numFactors_**
const **Size numLMVars_**
const **Size nSims_**
std::vector< std::vector< **simEvent**< derivedRandomLM< copulaPolicy, USNG > > > > **simsBuffer_**
copulaPolicy **copula_**
boost::shared_ptr< copulaRNG_type > **copulasRng_**

Static Protected Attributes

static const **Size maxHorizon_** = 4050

Additional Inherited Members

Detailed Description

template<template< class, class > class derivedRandomLM, class copulaPolicy, class USNG = SobolRsg>

class QuantLib::RandomLM< derivedRandomLM, copulaPolicy, USNG >" Base class for latent model monte carlo simulation. Independent of the copula type and the generator. Generates the factors and variable samples and determines event threshold but it is not responsible for actual event specification; thats the derived classes responsibility according to what they model. Derived classes need mainly to implement nextSample (Worker::nextSample in the multithreaded version) to compute the simulation event generated, if any, from the latent variables sample. They also have the accompanying event trait to specify.

Member Function Documentation

void update () `[protected], [virtual]`

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer**.

void performCalculations () const `[protected], [virtual]`

This method must implement any calculations which must be (re)done in order to calculate the desired results.

Implements **LazyObject**.

Probability probAtLeastNEvents (Size n, const Date & d) const `[protected], [virtual]`

Returns the probaility of having a given or larger number of defaults in the basket portfolio at a given time.

Reimplemented from **DefaultLossModel**.

Disposable< std::vector< Probability > > probsBeingNthEvent (Size n, const Date & d) const `[protected], [virtual]`

Order of results refers to the simulated (super)pool not the basket's pool. Notice that this statistic suffers from heavy dispersion. To see techniques to improve it (not implemented here) see: Joshi, M., D. Kainth. 2004. Rapid and accurate development of prices and **Greeks** for nth to default credit swaps in the Li model. Quantitative Finance, Vol. 4. Institute of Physics Publishing, London, UK, 266-275 and: Chen, Z., Glasserman, P. 'Fast pricing of basket default swaps' in Operations Research Vol. 56, No. 2, March/April 2008, pp. 286-303

Reimplemented from **DefaultLossModel**.

boost::tuples::tuple< Real, Real, Real > percentileAndInterval (const Date & d, Real percentile) const `[protected], [virtual]`

Returns the VaR value for a given percentile and the 95 confidence interval of that value.

Disposable< std::vector< Real > > splitVaRLevel (const Date & date, Real loss) const `[protected], [virtual]`

Distributes the total VaR amount along the portfolio counterparties. The passed loss amount is in loss units.

Reimplemented from **DefaultLossModel**.

Disposable< std::vector< std::vector< Real > > > splitVaRAndError (const Date & date, Real loss, Probability confInterval) const `[protected], [virtual]`

Distributes the total VaR amount along the portfolio counterparties.

Provides confidence interval for split so that portfolio optimization can be performed outside those limits.

The passed loss amount is in loss units.

Author

Generated automatically by Doxygen for QuantLib from the source code.