Monte Carlo Risk Engine

Generated by Doxygen 1.9.1

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 DjiaPortfolioSpec Struct Reference

Configuration parameters for automatic DJIA portfolio construction.

```
#include <djia_builder.hpp>
```

Public Attributes

- std::string snapshot_date
- double portfolio_value = 10'000'000
- std::string history_dir
- std::string output_file

3.1.1 Detailed Description

Configuration parameters for automatic DJIA portfolio construction.

This struct specifies:

- snapshot_date: date on which the portfolio should be generated
- portfolio_value: total notional value to allocate across Dow Jones stocks
- history_dir: directory containing Yahoo-format historical CSV files
- output_file: resulting CSV file where generated portfolio will be stored

3.1.2 Member Data Documentation

3.1.2.1 history_dir

```
std::string DjiaPortfolioSpec::history_dir
```

3.1.2.2 output_file

```
std::string DjiaPortfolioSpec::output_file
```

3.1.2.3 portfolio_value

```
double DjiaPortfolioSpec::portfolio_value = 10'000'000
```

3.1.2.4 snapshot_date

```
std::string DjiaPortfolioSpec::snapshot_date
```

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

- include/djia_builder.hpp

3.2 Instrument Struct Reference

Represents a single portfolio position.

```
#include <portfolio.hpp>
```

Public Attributes

- InstrumentType type
- std::string ticker
- int quantity
- double strike = 0.0

  option strike price
- double maturity = 0.0

  option maturity in years
- std::string option_type

3.2.1 Detailed Description

Represents a single portfolio position.

STOCK fields:

- ticker
- quantity

OPTION fields:

- strike
- maturity (not used yet in pricing)
- option_type: "CALL" or "PUT"

3.2.2 Member Data Documentation

3.2.2.1 maturity

```
double Instrument::maturity = 0.0
```

option maturity in years

3.2.2.2 option_type

```
std::string Instrument::option_type
```

3.2.2.3 quantity

```
int Instrument::quantity
```

3.2.2.4 strike

```
double Instrument::strike = 0.0
```

option strike price

3.2.2.5 ticker

```
std::string Instrument::ticker
```

3.2.2.6 type

```
InstrumentType Instrument::type
```

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

- include/portfolio.hpp

3.3 MarketDataHistory Class Reference

Container for historical daily price data for multiple tickers.

```
#include <market_data_history.hpp>
```

Public Member Functions

- void load_directory (const std::string &path)

  Loads all ∗.csv files from a directory.
- double get_price (const std::string &ticker, const std::string &date) const

  Returns the close price for a given ticker and date.
- std::vector< std::string > get_future_dates (const std::string &snapshot_date, int horizon_days) const

  Returns a list of the next horizon_days trading dates after snapshot_date.
- std::vector< std::string > get_past_dates (const std::string &snapshot_date, int lookback_days) const

  Returns a list of lookback_days dates preceding snapshot_date.
- std::unordered_map< std::string, std::vector< std::string > > get_all_dates () const

  Returns all available trading dates for every ticker.

Public Attributes

- std::unordered_map< std::string, std::map< std::string, double > > prices

3.3.1 Detailed Description

Container for historical daily price data for multiple tickers.

Stores: prices[ticker][date] = close price

Provides helper functions for:

- loading CSV files from a directory,
- accessing individual prices,
- retrieving past/future date ranges,
- extracting available dates for each ticker.

3.3.2 Member Function Documentation

3.3.2.1 get_all_dates()

```
std::unordered_map< std::string, std::vector< std::string > > MarketDataHistory::get_all_↩
dates ( ) const
```

Returns all available trading dates for every ticker.

Returns

Map ticker → array of available dates.

3.3.2.2 get_future_dates()

```
std::vector< std::string > MarketDataHistory::get_future_dates (
             const std::string & snapshot_date,
             int horizon_days ) const
```

Returns a list of the next horizon_days trading dates after snapshot_date.

Used to compute realized (historical) VaR/ES.

3.3.2.3 get_past_dates()

```
std::vector< std::string > MarketDataHistory::get_past_dates (
             const std::string & snapshot_date,
             int lookback_days ) const
```

Returns a list of lookback_days dates preceding snapshot_date.

Used to estimate drift, volatility, and correlations.

3.3.2.4 get_price()

```
double MarketDataHistory::get_price (
             const std::string & ticker,
             const std::string & date ) const
```

Returns the close price for a given ticker and date.

Parameters

ticker	Ticker symbol.
date	Date in YYYY-MM-DD format.

Exceptions

std::runtime_error	if ticker or date is missing.

3.3.2.5 load_directory()

```
void MarketDataHistory::load_directory (
            const std::string & path )
```

Loads all ∗.csv files from a directory.

CSV format (Yahoo-style): Date,Open,High,Low,Close,Adj Close,Volume

Ticker name is taken from filename (e.g., AAPL.csv → AAPL).

Parameters

path	Directory containing historical CSV files.

3.3.3 Member Data Documentation

3.3.3.1 prices

```
std::unordered_map<std::string, std::map<std::string, double> > MarketDataHistory::prices
```

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

- include/market_data_history.hpp
- src/market_data_history.cpp

3.4 MarketSnapshot Struct Reference

Represents a calibrated market state for Monte Carlo simulation.

```
#include <market_snapshot.hpp>
```

Public Attributes

- std::vector< std::string > tickers
- std::unordered_map< std::string, double > spot
- std::vector< double > mu
- std::vector< double > sigma
- std::vector< std::vector< double > > corr

3.4.1 Detailed Description

Represents a calibrated market state for Monte Carlo simulation.

Contains:

- list of required tickers,
- spot prices on the snapshot date,
- daily log-return mean vector (mu),
- daily volatilities (sigma),
- correlation matrix (corr).

This structure is generated from historical market data and used as input to Monte Carlo risk calculations.

3.4.2 Member Data Documentation

3.4.2.1 corr

```
std::vector<std::vector<double> > MarketSnapshot::corr
```

3.4.2.2 mu

```
std::vector<double> MarketSnapshot::mu
```

3.4.2.3 sigma

```
std::vector<double> MarketSnapshot::sigma
```

3.4.2.4 spot

```
std::unordered_map<std::string, double> MarketSnapshot::spot
```

3.4.2.5 tickers

```
std::vector<std::string> MarketSnapshot::tickers
```

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

- include/market_snapshot.hpp

3.5 MonteCarloEngine Class Reference

Monte Carlo simulation engine for multi-asset portfolios.

```
#include <monte_carlo.hpp>
```

Public Member Functions

- MonteCarloEngine (const MarketSnapshot &snap, const Portfolio &portfolio, int horizon_days)

  Constructs the Monte Carlo engine.
- void compute (int scenarios, double confidence, double &var_out, double &es_out)

  Runs Monte Carlo simulation and computes VaR and ES.

3.5.1 Detailed Description

Monte Carlo simulation engine for multi-asset portfolios.

The engine:

- uses GBM (geometric Brownian motion) price dynamics,
- generates correlated shocks using Cholesky decomposition,
- simulates value changes over a given horizon,
- computes portfolio-level P&L,
- estimates Value-at-Risk (VaR) and Expected Shortfall (ES).

Supports stocks and European-style options.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 MonteCarloEngine()

```
MonteCarloEngine::MonteCarloEngine (
            const MarketSnapshot & snap,
            const Portfolio & portfolio,
            int horizon_days )
```

Constructs the Monte Carlo engine.

Parameters

snap	Market snapshot containing drift, vol, correlation.
portfolio	Portfolio of instruments (stock / option).
horizon_days	Horizon for VaR simulation in trading days.

The constructor automatically computes the Cholesky matrix from the correlation matrix.

3.5.3 Member Function Documentation

3.5.3.1 compute()

```
void MonteCarloEngine::compute (
             int scenarios,
             double confidence,
             double & var_out,
             double & es_out )
```

Runs Monte Carlo simulation and computes VaR and ES.

Parameters

scenarios	Number of Monte Carlo paths to generate.
confidence	Confidence level (e.g., 0.95).
var_out	Output absolute VaR value.
es_out	Output absolute ES value.

Resulting VaR/ES are positive numbers representing losses.

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

- include/monte_carlo.hpp
- src/monte_carlo.cpp

3.6 Portfolio Class Reference

Portfolio consisting of stocks and European-style options.

```
#include <portfolio.hpp>
```

Public Member Functions

- void load (const std::string &file)

  Loads portfolio positions from a CSV file.

Public Attributes

- std::vector< Instrument > instruments

3.6.1 Detailed Description

Portfolio consisting of stocks and European-style options.

Provides loading from CSV format: type,ticker,quantity,strike,maturity,option_type

Unknown instrument types trigger exceptions.

3.6.2 Member Function Documentation

3.6.2.1 load()

```
void Portfolio::load (
            const std::string & file )
```

Loads portfolio positions from a CSV file.

Parameters

file	Path to portfolio CSV.

Exceptions

std::runtime_error	if: • file cannot be opened, • row format is invalid, • instrument type is unknown.

3.6.3 Member Data Documentation

3.6.3.1 instruments

```
std::vector<Instrument> Portfolio::instruments
```

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

- include/portfolio.hpp
- src/portfolio.cpp

3.7 RealizedRisk Struct Reference

Container holding historical VaR and ES values.

```
#include <realized_risk.hpp>
```

Public Attributes

- double historical_var
- double historical_es

3.7.1 Detailed Description

Container holding historical VaR and ES values.

Values are expressed as positive losses (e.g. VaR = 0.03 = 3% loss).

3.7.2 Member Data Documentation

3.7.2.1 historical_es

```
double RealizedRisk::historical_es
```

3.7.2.2 historical_var

```
double RealizedRisk::historical_var
```

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

- include/realized_risk.hpp

Chapter 4

File Documentation

4.1 include/cholesky.hpp File Reference

```
#include <vector>
```

Functions

- std::vector< std::vector< double > > cholesky (const std::vector< std::vector< double >> &A)

  Computes the Cholesky decomposition of a symmetric positive-definite matrix.

4.1.1 Function Documentation

4.1.1.1 cholesky()

```
std::vector<std::vector<double> > cholesky (
            const std::vector< std::vector< double >> & A )
```

Computes the Cholesky decomposition of a symmetric positive-definite matrix.

This function constructs the lower-triangular matrix L such that: A = L $*$ L

It is used in the Monte-Carlo engine to introduce correlations between asset returns by transforming independent normal shocks into correlated ones.

Parameters

A	Input square matrix (NxN), must be symmetric and positive-definite.

Returns

Lower-triangular matrix L of the same size, where L ∗ L = A.

Exceptions

std::runtime_error	If the matrix is not positive-definite (i.e., diagonal element becomes 0 during factorization).

4.2 include/djia_builder.hpp File Reference

```
#include <string>
```

Classes

- struct DjiaPortfolioSpec

  Configuration parameters for automatic DJIA portfolio construction.

Functions

- bool build_djia_portfolio (const DjiaPortfolioSpec &spec)

  Builds a synthetic Dow Jones (DJIA) stock-only portfolio.

4.2.1 Function Documentation

4.2.1.1 build_djia_portfolio()

```
bool build_djia_portfolio (
            const DjiaPortfolioSpec & spec )
```

Builds a synthetic Dow Jones (DJIA) stock-only portfolio.

The portfolio is constructed using price-weighted DJIA methodology:

- loads historical data for all 30 DJIA tickers,
- extracts closing prices on the given snapshot date,
- assigns weights proportional to stock prices,
- allocates total portfolio_value according to weights,
- writes result into a CSV file.

Parameters

spec	Structure describing portfolio generation parameters.

Returns

true on success, false if any ticker is missing or historical data is unavailable.

Exceptions

No	exceptions are thrown directly; errors are printed to stderr.

4.3 include/market_data_history.hpp File Reference

```
#include <string>
#include <unordered_map>
#include <map>
#include <vector>
```

Classes

- class MarketDataHistory

  Container for historical daily price data for multiple tickers.

4.4 include/market_snapshot.hpp File Reference

```
#include <vector>
#include <string>
#include <unordered_map>
```

Classes

- struct MarketSnapshot

  Represents a calibrated market state for Monte Carlo simulation.

Functions

- MarketSnapshot build_snapshot (const MarketDataHistory &hist, const std::vector< std::string > &tickers, const std::string &snapshot_date, int lookback_days)

  Builds a market snapshot calibrated from historical prices.

4.4.1 Function Documentation

4.4.1.1 build_snapshot()

```
MarketSnapshot build_snapshot (
            const MarketDataHistory & hist,
            const std::vector< std::string > & tickers,
            const std::string & snapshot_date,
            int lookback_days )
```

Builds a market snapshot calibrated from historical prices.

Steps:

1. Reads spot prices for all tickers on snapshot_date.
2. Extracts lookback_days of historical returns.
3. Computes per-asset drift (mu) and volatility (sigma).
4. Computes full correlation matrix.

Parameters

hist	MarketDataHistory object containing all historical prices.
tickers	List of portfolio tickers.
snapshot_date	Date on which risk should be evaluated.
lookback_days	Number of past trading days for estimation.

Exceptions

std::runtime_error	if insufficient data is available.

4.5 include/monte_carlo.hpp File Reference

```
#include "market_snapshot.hpp"
#include "portfolio.hpp"
#include "cholesky.hpp"
#include <vector>
#include <string>
#include <random>
```

Classes

- class MonteCarloEngine

  Monte Carlo simulation engine for multi-asset portfolios.

4.6 include/portfolio.hpp File Reference

```
#include <vector>
#include <string>
```

Classes

- struct Instrument

  Represents a single portfolio position.
- class Portfolio

  Portfolio consisting of stocks and European-style options.

Enumerations

- enum class InstrumentType { STOCK , OPTION }

  Type of financial instrument supported by the system.

4.6.1 Enumeration Type Documentation

4.6.1.1 InstrumentType

```
enum InstrumentType [strong]
```

Type of financial instrument supported by the system.

Enumerator

STOCK	
OPTION	

4.7 include/realized_risk.hpp File Reference

```
#include <string>
```

Classes

- struct RealizedRisk

  Container holding historical VaR and ES values.

Functions

- RealizedRisk compute_realized_risk (const Portfolio &portfolio, const MarketDataHistory &history, const std::string &snapshot_date, int horizon_days)

  Computes realized (historical) VaR and ES.

4.7.1 Function Documentation

4.7.1.1 compute_realized_risk()

```
RealizedRisk compute_realized_risk (
             const Portfolio & portfolio,
             const MarketDataHistory & history,
             const std::string & snapshot_date,
             int horizon_days )
```

Computes realized (historical) VaR and ES.

Method:

1. Computes portfolio value on snapshot_date.
2. Collects actual market prices for the next horizon_days.
3. Computes percentage returns.
4. Converts them to losses.
5. Computes: VaR = 95th percentile of losses, ES = mean of worst 5% losses.

VaR/ES are capped at zero (cannot be negative).

Parameters

portfolio	The portfolio whose risk is measured.
history	Historical price database.
snapshot_date	Starting date of VaR horizon.
horizon_days	Number of future trading days to examine.

Returns

RealizedRisk struct with VaR and ES.

Exceptions

std::runtime_error	if insufficient data is available.

4.8 include/trading_day_utils.hpp File Reference

```
#include <string>
#include <vector>
#include <unordered_map>
```

Functions

- std::string find_common_previous_date (const std::unordered_map< std::string, std::vector< std::string >> &ticker_dates, const std::string &target)

  Finds the nearest trading date common to all tickers that does not exceed the given target date.

4.8.1 Function Documentation

4.8.1.1 find_common_previous_date()

```
std::string find_common_previous_date (
            const std::unordered_map< std::string, std::vector< std::string >> & ticker_↩
dates,
            const std::string & target )
```

Finds the nearest trading date common to all tickers that does not exceed the given target date.

Used for:

- aligning inconsistent historical datasets,
- ensuring snapshot_date exists for all tickers,
- DJIA portfolio generation when a day is missing.

Parameters

ticker_dates	Map: ticker → list of sorted trading dates.
target	Target date (YYYY-MM-DD).

Returns

Latest common date target.

Exceptions

std::runtime_error	if: • no tickers provided, • there is no common trading day, • target date precedes all common dates.

4.9 src/cholesky.cpp File Reference

```
#include "cholesky.hpp"
#include <cmath>
#include <stdexcept>
```

Functions

- std::vector< std::vector< double > > cholesky (const std::vector< std::vector< double >> &A)

  Computes the Cholesky decomposition of a symmetric positive-definite matrix.

4.9.1 Function Documentation

4.9.1.1 cholesky()

```
std::vector<std::vector<double> > cholesky (
            const std::vector< std::vector< double >> & A )
```

Computes the Cholesky decomposition of a symmetric positive-definite matrix.

This function constructs the lower-triangular matrix L such that: A = L ∗ L

It is used in the Monte-Carlo engine to introduce correlations between asset returns by transforming independent normal shocks into correlated ones.

Parameters

A	Input square matrix (NxN), must be symmetric and positive-definite.

Returns

Lower-triangular matrix L of the same size, where L ∗ L = A.

Exceptions

std::runtime_error	If the matrix is not positive-definite (i.e., diagonal element becomes 0 during factorization).

4.10 src/djia_builder.cpp File Reference

```
#include "djia_builder.hpp"
#include "market_data_history.hpp"
#include <fstream>
#include <iostream>
#include <unordered_map>
#include <filesystem>
```

Functions

- bool build_djia_portfolio (const DjiaPortfolioSpec &spec)

  Builds a synthetic Dow Jones (DJIA) stock-only portfolio.

4.10.1 Function Documentation

4.10.1.1 build_djia_portfolio()

```
bool build_djia_portfolio (
            const DjiaPortfolioSpec & spec )
```

Builds a synthetic Dow Jones (DJIA) stock-only portfolio.

The portfolio is constructed using price-weighted DJIA methodology:

- loads historical data for all 30 DJIA tickers,
- extracts closing prices on the given snapshot date,
- assigns weights proportional to stock prices,
- allocates total portfolio_value according to weights,
- writes result into a CSV file.

Parameters

spec	Structure describing portfolio generation parameters.

Returns

true on success, false if any ticker is missing or historical data is unavailable.

Exceptions

No	exceptions are thrown directly; errors are printed to stderr.

4.11 src/main.cpp File Reference

```
#include <iostream>
#include <unordered_set>
#include "market_data_history.hpp"
#include "market_snapshot.hpp"
#include "portfolio.hpp"
#include "monte_carlo.hpp"
#include "realized_risk.hpp"
#include "djia_builder.hpp"
#include "trading_day_utils.hpp"
```

Functions

- std::vector< std::string > get_unique_tickers (const Portfolio &p)
- void remove_missing_tickers (Portfolio &p, const MarketDataHistory &h)
- int main (int argc, char **argv)

4.11.1 Function Documentation

4.11.1.1 get_unique_tickers()

```
std::vector<std::string> get_unique_tickers (
            const Portfolio & p )
```

4.11.1.2 main()

```
int main (
            int argc,
            char ** argv )
```

4.11.1.3 remove_missing_tickers()

```
void remove_missing_tickers (
            Portfolio & p,
            const MarketDataHistory & h )
```

4.12 src/market_data_history.cpp File Reference

```
#include "market_data_history.hpp"
#include <filesystem>
#include <fstream>
#include <sstream>
#include <stdexcept>
```

4.13 src/market_snapshot.cpp File Reference

```
#include "market_snapshot.hpp"
#include "market_data_history.hpp"
#include <cmath>
#include <stdexcept>
```

Functions

- MarketSnapshot build_snapshot (const MarketDataHistory &hist, const std::vector< std::string > &tickers, const std::string &snapshot_date, int lookback_days)

  Builds a market snapshot calibrated from historical prices.

4.13.1 Function Documentation

4.13.1.1 build_snapshot()

```
MarketSnapshot build_snapshot (
            const MarketDataHistory & hist,
            const std::vector< std::string > & tickers,
            const std::string & snapshot_date,
            int lookback_days )
```

Builds a market snapshot calibrated from historical prices.

Steps:

1. Reads spot prices for all tickers on snapshot_date.
2. Extracts lookback_days of historical returns.
3. Computes per-asset drift (mu) and volatility (sigma).
4. Computes full correlation matrix.

Parameters

hist	MarketDataHistory object containing all historical prices.
tickers	List of portfolio tickers.
snapshot_date	Date on which risk should be evaluated.
lookback_days	Number of past trading days for estimation.

Exceptions

std::runtime_error	if insufficient data is available.

4.14 src/monte_carlo.cpp File Reference

```
#include "monte_carlo.hpp"
#include "cholesky.hpp"
#include <random>
#include <cmath>
#include <algorithm>
#include <stdexcept>
```

4.15 src/portfolio.cpp File Reference

```
#include "portfolio.hpp"
#include <fstream>
#include <sstream>
#include <stdexcept>
```

4.16 src/realized_risk.cpp File Reference

```
#include "realized_risk.hpp"
#include "market_data_history.hpp"
#include "portfolio.hpp"
#include <algorithm>
#include <stdexcept>
```

Functions

- RealizedRisk compute_realized_risk (const Portfolio &portfolio, const MarketDataHistory &history, const std::string &snapshot_date, int horizon_days)

  Computes realized (historical) VaR and ES.

4.16.1 Function Documentation

4.16.1.1 compute_realized_risk()

```
RealizedRisk compute_realized_risk (
              const Portfolio & portfolio,
              const MarketDataHistory & history,
              const std::string & snapshot_date,
              int horizon_days )
```

Computes realized (historical) VaR and ES.

Method:

1. Computes portfolio value on snapshot_date.
2. Collects actual market prices for the next horizon_days.
3. Computes percentage returns.
4. Converts them to losses.
5. Computes: VaR = 95th percentile of losses, ES = mean of worst 5% losses.

VaR/ES are capped at zero (cannot be negative).

Parameters

portfolio	The portfolio whose risk is measured.
history	Historical price database.
snapshot_date	Starting date of VaR horizon.
horizon_days	Number of future trading days to examine.

Returns

RealizedRisk struct with VaR and ES.

Exceptions

std::runtime_error	if insufficient data is available.

4.17 src/trading_day_utils.cpp File Reference

```
#include "trading_day_utils.hpp"
#include <algorithm>
#include <stdexcept>
```

Functions

- std::string find_common_previous_date (const std::unordered_map< std::string, std::vector< std::string >> &ticker_dates, const std::string &target)

  Finds the nearest trading date common to all tickers that does not exceed the given target date.

4.17.1 Function Documentation

4.17.1.1 find_common_previous_date()

```
std::string find_common_previous_date (
                const std::unordered_map< std::string, std::vector< std::string >> & ticker_↩
dates,
                const std::string & target )
```

Finds the nearest trading date common to all tickers that does not exceed the given target date.

Used for:

- aligning inconsistent historical datasets,
- ensuring snapshot_date exists for all tickers,
- DJIA portfolio generation when a day is missing.

Parameters

ticker_dates	Map: ticker → list of sorted trading dates.
target	Target date (YYYY-MM-DD).

Returns

Latest common date target.

Exceptions

std::runtime_error	if: • no tickers provided, • there is no common trading day, • target date precedes all common dates.