C++ Financial Library

Generated by Doxygen 1.8.1.2

Sat Aug 24 2013 19:25:45

Contents

Chapter 1

Namespace Index

1.1 Namespace List

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Namespace Documentation

4.1 financial Namespace Reference

User library namespace.

Classes

- struct TimedCashFlow

  Timed cash flow structure.

Enumerations

- enum disc_type

  Enumeration class for discounting types.
- enum annuity_type

  Enumeration class for annuity types.

Functions

- double compound_factor (const double interest_rate, const double num_periods=1, const enum disc_type dt=disc_type::discrete)

  Calculates a compounding factor.
- double discount_factor (const double interest_rate, const double num_periods=1, const enum disc_type dt=disc_type::discrete)

  Calculates a discount factor.
- double pv (const double cashflow, const double interest_rate, const double num_periods=1, const enum disc-_type dt=disc_type::discrete)

  Calculates the present value of a single cash flow.
- double fv (const double cashflow, const double interest_rate, const double num_periods=1, const enum disc-_type=disc_type::discrete)

  Calculates the future value of a single invested cash flow.
- double pv_perpetuity (const double cashflow, const double interest_rate, const enum annuity_type at=annuity_type::immediate)

  Calculates the present value of a perpetuity.
- double pv_annuity (const double cashflow, const double interest_rate, const int num_periods, const enum annuity_type at=annuity_type::immediate)

  Calculates the present value of an annuity.

- double pv_stream (const std::vector< TimedCashFlow > &cashflows, const double interest_rate)

  Calculates the present value of a stream of cash flows.
- double sinking_fund_payment (const double fund_value, const double interest_rate, const double num_-periods)

  Calculates the periodic payment to a sinking fund.
- double loan_repayment (const double loan_amount, const double interest_rate, const double num_periods)

  Calculates the periodic repayment of a loan.

Variables

- const double e = 2.71828182845904523536

  Euler's number.

4.1.1 Enumeration Type Documentation

4.1.1.1 enum **financial::annuity_type**

Annuities (and perpetuities) can be *immediate*, where each payment comes at the end of each period, or *due*, where each payment comes at the beginning of each period.

4.1.1.2 enum **financial::disc_type**

Discounting can be done *discretely*, where interest is calculated once every period, or *continuously*, where interest is calculated on a theoretically continual basis. Discrete compounding is far more common in practice.

4.1.2 Function Documentation

4.1.2.1 double financial::compound_factor (const double *interest_rate,* const double *num_periods* = `1`, const enum **disc_type** *dt* = `disc_type::discrete`)

A compound factor is a number which, when multiplied by an initial investment, will yield the value of that investment after a specified number of periods at a specific interest rate.

For instance, \$100 invested for two years at an interest rate of 5% per year will be worth \$100 $*$ 1.05 = \$105 after the first year, and \$105 $*$ 1.05 = \$110.25 at the end of the second year. The compound factor for two periods at 5% per period is therefore 110.25 / 100.0 = 1.1025, since \$100 $*$ 1.1025 = \$110.25.

Sample usage:

```
double cf = financial::compound_factor(0.1, 7);
std::cout << "The value of $100 invested today after 7 "
          << "years at an annual interest rate of 10% wil be $"
          << 100 * cf << std::endl;
```

Parameters

interest_rate	the periodic interest rate.
num_periods	the number of periods over which to compound.
dt	the type of compounding to use.

Returns

the calculated compounding factor.

4.1.2.2 double financial::discount_factor (const double *interest_rate*, const double *num_periods* = `1`, const enum **disc_type** *dt* = `disc_type::discrete`)

A discount factor is a number which, when multiplied by a future amount, will yield the value of the investment which, if invested today for a specified number of periods at a specific interest rate, will be worth that future amount.

For instance, \$100 invested for two years at an interest rate of 5% per year will be worth \$100 $*$ 1.05 = \$105 after the first year, and \$105 $*$ 1.05 = \$110.25 at the end of the second year. The discount factor for two periods at 5% per period is therefore 100.0 / 110.25 = 0.90703, since \$110.25 $*$ 0.90703 = \$100.

Sample usage:

```
double df = financial::discount_factor(0.12, 4);
std::cout << "The value of $1000 received in 4 years time "
          << " is " << 1000 * df << " today, assuming an "
          << "annual interest rate of 12%." << std::endl;
```

Parameters

interest_rate	the periodic interest rate.
num_periods	the number of periods over which to discount.
dt	the type of compounding to use.

Returns

the calculated discount factor.

4.1.2.3 double financial::fv (const double *cashflow*, const double *interest_rate*, const double *num_periods* = `1`, const enum **disc_type** *dt* = `disc_type::discrete`)

Future value is a concept related to the time value of money, which states that an amount of money today is worth more than the same amount of money in the future, as a result of investment opportunities available which themselves arise from the fact that, all else being equal, consumption now is preferred to consumption at a future date.

For instance, if money can be invested at an interest rate of 5% per year, then \$100 invested today will be worth \$100 $*$ 1.05 = \$105 in one year's time. If given a choice between receiving \$99 today or \$105 in one year's time, therefore, a rational person would choose to receive \$105 in one year's time, since foregoing the opportunity to receive \$99 in return for a future payment is equivalent to investing \$99 today, which would yield only \$99 $*$ 1.05% = \$103.95, less than \$105. In other words, under these conditions, \$105 in one year's time is worth more than \$99 today, and the future value of \$100 today is \$105 in one year's time.

Sample usage:

```
double fval = financial::fv(100, 0.1, 7);
std::cout << "The value of $100 invested today after 7 "
          << "years at an annual interest rate of 10% wil be $"
          << fval << std::endl;
```

Parameters

cashflow	the nominal amount of the invested cash flow
interest_rate	the periodic interest rate
num_periods	the number of periods until maturity
dt	the type of compounding to use

Returns

the future value of the cash flow

4.1.2.4 double financial::loan_repayment (const double *loan_amount*, const double *interest_rate*, const double *num_periods*)

A loan continues to accrue interest on the outstanding principal even while periodic repayments are being made. The principal problem is calculating, given the time to repayment and an interest rate assumption, the periodic payment that must be made in order to pay off the entire principal and any interest accrued over the life of the loan by the payoff date.

```
double month_rate = std::pow(1.0425, 1/12) - 1;
double lp = financial::loan_repayment(250000,
      month_rate, 360);
std::cout << "To pay off a $250,000 mortgage after 30 "
          << "years at a 4.25% annual interest rate, you "
          << "will need to make 360 monthly payments of $"
          << lp << " per month." << std::endl;
```

Parameters

loan_amount	the amount of the loan
interest_rate	the periodic interest rate
num_periods	the number of periodic repayments

Returns

the nominal amount of the periodic loan repayment

4.1.2.5 double financial::pv (const double *cashflow*, const double *interest_rate*, const double *num_periods* = `1`, const enum disc_type *dt* = `disc_type::discrete`)

Present value is a concept related to the time value of money, which states that an amount of money today is worth more than the same amount of money in the future, as a result of investment opportunities available which themselves arise from the fact that, all else being equal, consumption now is preferred to consumption at a future date.

For instance, if money can be invested at an interest rate of 5% per year, then $100 invested today will be worth $100 ∗ 1.05 = $105 in one year's time. If given a choice between receiving $100 today or $104 in one year's time, therefore, a rational person would choose to receive $100 today, since by investing it she can receive $105 in one year's time rather than $104 in one year's time. In other words, under these conditions, $100 today is worth more than $105 in one year's time, and the present value of $105 received in one year's time is $100.

Sample usage:

```
double pval = financial::pv(1000, 0.12, 4);
std::cout << "The value of $1000 received in 4 years time "
          << " is " << pval << " today, assuming an annual "
          << "interest rate of 12%." << std::endl;
```

Parameters

cashflow	the nominal amount of the single cash flow
interest_rate	the periodic interest rate
num_periods	the number of periods until the cash flow
dt	the type of discounting to use

Returns

the present value of the cash flow

4.1.2.6 double financial::pv_annuity (const double *cashflow,* const double *interest_rate,* const int *num_periods,* const enum **annuity_type** *at* = `annuity_type::immediate`)

An annuity is a periodic cash flow received for a specified period of time (in reality, many annuities are received in the form of life annuities, where payments continue until the death of the holder, and the period is therefore not fully specified, causing the issuing financial institution to bear some mortality risk). The present value of an annuity is equal to the present value of a perpetuity under the same terms, less the present value of an perpetuity starting at the end of the annuity's payout period.

Parameters

cashflow	the nominal amount of the periodic cash flow
interest_rate	the periodic interest rate
num_periods	the number of periodic cash flows
at	the type of annuity

Returns

the present value of the annuity

4.1.2.7 double financial::pv_perpetuity (const double *cashflow,* const double *interest_rate,* const enum **annuity_type** *at* = `annuity_type::immediate`)

A perpetuity is a periodic cash flow received from now until the end of time. Although at first glance this may seem to have infinite value, the time value of money causes the present value of each cash flow to approach zero as the time period increases, so a perpetuity does have a finite value, equal - for an immediate perpetuity, where the first periodic payment is received at the end of the current period, rather than at the beginning - to the periodic cash flow divided by the interest rate.

Parameters

cashflow	the nominal amount of the periodic cash flow
interest_rate	the periodic interest rate
at	the type of perpetuity

Returns

the present value of the perpetuity

4.1.2.8 double financial::pv_stream (const std::vector< **TimedCashFlow** > & *cashflows,* const double *interest_rate*)

Just as the present value of a single payment can be calculated, so can the present value of a stream of timed payments. This technique is often used for valuing financial instruments with regular payouts, and for valuing investment opportunities more generally.

Parameters

cashflows	a std::vector of TimedCashFlow structs representing the stream of cash flows.
interest_rate	the periodic interest rate

Returns

the present value of the stream of cash flows.

4.1.2.9 double financial::sinking_fund_payment (const double *fund_value,* const double *interest_rate,* const double *num_periods*)

A sinking fund is an investment which is designed to equal a specific future value at a specified point in time. It is often used as a fund to pay off a known future liability such as a corporate bond maturity, and a retirement fund aiming to provide a particular amount for retirement is analagous. The principal problem with sinking funds is calculating, given the amount of time to termination and an assumption of interest rates, the amount of the periodic payment which must be made to cause the terminal value of the fund to equal the desired future amount.

Example of usage:

```
double sfp = financial::sinking_fund_payment(100
      00, 0.05, 5);
std::cout << "To achieve a fund value of $10,000 in 5 years at "
             "5% per year, you must invest " << sfp
             " dollars at the end of each year." << std::endl;
```

Parameters

fund_value	the terminal value of the sinking fund
interest_rate	the periodic interest rate
num_periods	the number of periodic payments

Returns

the nominal amount of the required periodic payment

4.1.3 Variable Documentation

4.1.3.1 const double financial::e = 2.71828182845904523536

Euler's number is the limit of `(1 + 1/n) ** n` as `n` approaches infinity.

The exponention *function*, `e ** x`, is the only nontrivial function which is its own derivative, and its own antiderivative too.

Chapter 5

Class Documentation

5.1 financial::TimedCashFlow Struct Reference

Timed cash flow structure.

`#include <common_financial_types.h>`

Public Member Functions

- TimedCashFlow ()

  Default constructor.
- TimedCashFlow (const double amount, const double time_period)

  Constructor.

Public Attributes

- double amount
- double time_period

5.1.1 Detailed Description

The `TimedCashFlow` struct describes both the amount and the timing of a future cash flow.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 financial::TimedCashFlow::TimedCashFlow ( ) `[inline], [explicit]`

Initializes members to zero.

5.1.2.2 financial::TimedCashFlow::TimedCashFlow (const double *amount*, const double *time_period*) `[inline], [explicit]`

Initializes members to provided values.

Parameters

amount	the amount of the cash flow
time_period	the period at which the cash flow will occur

5.1.3 Member Data Documentation

5.1.3.1 double financial::TimedCashFlow::amount

the amount of the cash flow

5.1.3.2 double financial::TimedCashFlow::time_period

the period at which the cash flow will occur

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

- common_financial_types.h

Chapter 6

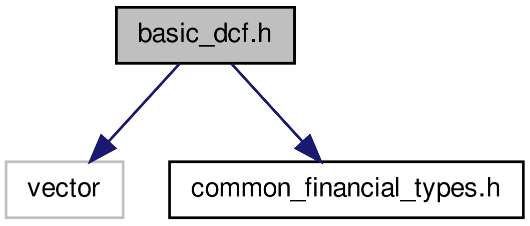
File Documentation

6.1 basic_dcf.h File Reference

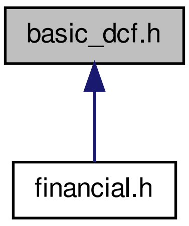
Basic discounted cash flow functions.

```
#include <vector>
#include "common_financial_types.h"
```

Include dependency graph for basic_dcf.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace financial

  User library namespace.

Functions

- double financial::compound_factor (const double interest_rate, const double num_periods=1, const enum disc_type dt=disc_type::discrete)

  Calculates a compounding factor.
- double financial::discount_factor (const double interest_rate, const double num_periods=1, const enum disc_type dt=disc_type::discrete)

  Calculates a discount factor.
- double financial::pv (const double cashflow, const double interest_rate, const double num_periods=1, const enum disc_type dt=disc_type::discrete)

  Calculates the present value of a single cash flow.
- double financial::fv (const double cashflow, const double interest_rate, const double num_periods=1, const enum disc_type=disc_type::discrete)

  Calculates the future value of a single invested cash flow.
- double financial::pv_perpetuity (const double cashflow, const double interest_rate, const enum annuity_type at=annuity_type::immediate)

  Calculates the present value of a perpetuity.
- double financial::pv_annuity (const double cashflow, const double interest_rate, const int num_periods, const enum annuity_type at=annuity_type::immediate)

  Calculates the present value of an annuity.
- double financial::pv_stream (const std::vector< TimedCashFlow > &cashflows, const double interest_rate)

  Calculates the present value of a stream of cash flows.
- double financial::sinking_fund_payment (const double fund_value, const double interest_rate, const double num_periods)

  Calculates the periodic payment to a sinking fund.
- double financial::loan_repayment (const double loan_amount, const double interest_rate, const double num_periods)

  Calculates the periodic repayment of a loan.

6.1.1 Detailed Description

Basic discounted cash flow functions, including for calculating present and future values, values of perpetuities and annuities, loan repayments and sinking fund payments.

Author

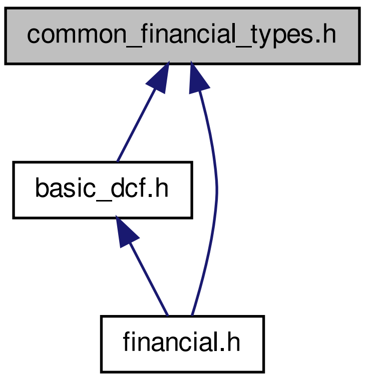
Paul Griffiths

Copyright

6.2 common_financial_types.h File Reference

Common types, enums and constants.

This graph shows which files directly or indirectly include this file:



Classes

- struct financial::TimedCashFlow

  Timed cash flow structure.

Namespaces

- namespace financial

  User library namespace.

Enumerations

- enum financial::disc_type

  Enumeration class for discounting types.
- enum financial::annuity_type

  Enumeration class for annuity types.

Variables

- const double financial::e = 2.71828182845904523536

  Euler's number.

6.2.1 Detailed Description

Common types, enums and constants.

Author

Paul Griffiths

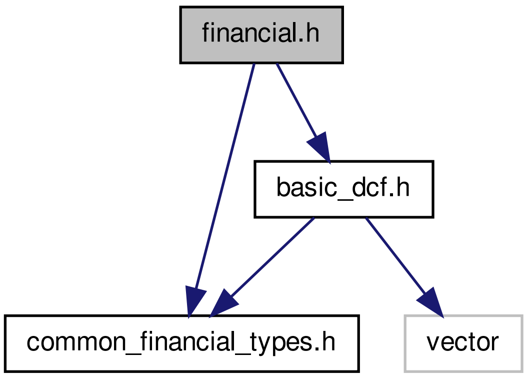
Copyright

6.3 financial.h File Reference

Financial library user header.

```
#include "common_financial_types.h"
#include "basic_dcf.h"
```

Include dependency graph for financial.h:



6.3.1 Detailed Description

User header file for financial library.

Author

Paul Griffiths

Copyright

Index