

C++ Financial Library

Generated by Doxygen 1.8.1.2

Sat Aug 24 2013 18:35:01

Contents

1	Namespace Index	1
1.1	Namespace List	1
2	Class Index	3
2.1	Class List	3
3	File Index	5
3.1	File List	5
4	Namespace Documentation	7
4.1	financial Namespace Reference	7
4.1.1	Detailed Description	8
4.1.2	Function Documentation	8
4.1.2.1	compound_factor	8
4.1.2.2	discount_factor	8
4.1.2.3	fv	9
4.1.2.4	loan_repayment	9
4.1.2.5	pv	10
4.1.2.6	pv_annuity	10
4.1.2.7	pv_perpetuity	10
4.1.2.8	pv_stream	11
4.1.2.9	sinking_fund_payment	11
5	Class Documentation	13
5.1	financial::TimedCashFlow Struct Reference	13
5.1.1	Detailed Description	13
5.1.2	Constructor & Destructor Documentation	13
5.1.2.1	TimedCashFlow	13
5.1.3	Member Data Documentation	13
5.1.3.1	amount	13
5.1.3.2	time_period	14
6	File Documentation	15

6.1	common_financial_types.h File Reference	15
6.1.1	Detailed Description	16

Chapter 1

Namespace Index

1.1 Namespace List

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

financial	Basic financial functions	7
---------------------------	-------------------------------------	---

Chapter 2

Class Index

2.1 Class List

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

financial::TimedCashFlow	
Timed cash flow structure	13

Chapter 3

File Index

3.1 File List

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

basic_dcf.h	??
common_financial_types.h	
Common financial types and constants	15
financial.h	??

Chapter 4

Namespace Documentation

4.1 financial Namespace Reference

Basic financial functions.

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](#) dt=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 Detailed Description

Basic financial functions, including:

- present and future value calculations; and
- perpetuity and annuity valuations.

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 .

Parameters

<i>interest_rate</i>	the periodic interest rate.
<i>num_periods</i>	the number of periods over which to compound.
<i>dt</i>	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 .

Parameters

<i>interest_rate</i>	the periodic interest rate.
<i>num_periods</i>	the number of periods over which to discount.
<i>dt</i>	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.

Parameters

<i>cashflow</i>	the nominal amount of the invested cash flow
<i>interest_rate</i>	the periodic interest rate
<i>num_periods</i>	the number of periods until maturity
<i>dt</i>	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.

Parameters

<i>loan_amount</i>	the amount of the loan
<i>interest_rate</i>	the periodic interest rate
<i>num_periods</i>	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.

Parameters

<i>cashflow</i>	the nominal amount of the single cash flow
<i>interest_rate</i>	the periodic interest rate
<i>num_periods</i>	the number of periods until the cash flow
<i>dt</i>	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

<i>cashflow</i>	the nominal amount of the periodic cash flow
<i>interest_rate</i>	the periodic interest rate
<i>num_periods</i>	the number of periodic cash flows
<i>at</i>	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

<i>cashflow</i>	the nominal amount of the periodic cash flow
<i>interest_rate</i>	the periodic interest rate
<i>at</i>	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

<i>cashflows</i>	a std::vector of TimedCashFlow structs representing the stream of cash flows.
<i>interest_rate</i>	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 = sinking_fund_payment(10000, 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 per year." << std::endl;
```

Parameters

<i>fund_value</i>	the terminal value of the sinking fund
<i>interest_rate</i>	the periodic interest rate
<i>num_periods</i>	the number of periodic payments

Returns

the nominal amount of the required periodic payment

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 (const double amount, const double time_period) [inline], [explicit]`

Parameters

<i>amount</i>	the amount of the cash flow
<i>time_period</i>	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](#)

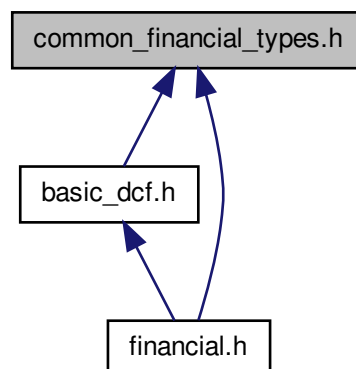
Chapter 6

File Documentation

6.1 common_financial_types.h File Reference

Common financial types and constants.

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



Classes

- struct `financial::TimedCashFlow`
Timed cash flow structure.

Namespaces

- namespace `financial`
Basic financial functions.

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.

6.1.1 Detailed Description

Index

- amount
 - financial::TimedCashFlow, [13](#)
- common_financial_types.h, [15](#)
- compound_factor
 - financial, [8](#)
- discount_factor
 - financial, [8](#)
- financial, [7](#)
 - compound_factor, [8](#)
 - discount_factor, [8](#)
 - fv, [9](#)
 - loan_repayment, [9](#)
 - pv, [10](#)
 - pv_annuity, [10](#)
 - pv_perpetuity, [10](#)
 - pv_stream, [11](#)
 - sinking_fund_payment, [11](#)
- financial::TimedCashFlow, [13](#)
 - amount, [13](#)
 - time_period, [13](#)
 - TimedCashFlow, [13](#)
- fv
 - financial, [9](#)
- loan_repayment
 - financial, [9](#)
- pv
 - financial, [10](#)
- pv_annuity
 - financial, [10](#)
- pv_perpetuity
 - financial, [10](#)
- pv_stream
 - financial, [11](#)
- sinking_fund_payment
 - financial, [11](#)
- time_period
 - financial::TimedCashFlow, [13](#)
- TimedCashFlow
 - financial::TimedCashFlow, [13](#)