Package 'ragtop'

July 8, 2026

Type Package

Title Pricing Equity Derivatives with Extensions of Black-Scholes

Version 2.0.0

Date 2026-07-08

Description Algorithms to price American and European equity options, convertible bonds and a variety of other financial derivatives. It uses an extension of the usual Black-Scholes model in which jump to default may occur at a probability specified by a power-law link between stock price and hazard rate as found in the paper by Takahashi, Kobayashi, and Nakagawa (2001) <doi:10.3905/jfi.2001.319302>. We use ideas and techniques from Andersen and Buffum (2002) <doi:10.2139/ssrn.355308> and Linetsky (2006) <doi:10.1111/j.1467-9965.2006.00271.x>.

License GPL (>= 2)

Depends R (>= 3.5)

Imports futile.logger (>= 1.4.1), methods (>= 3.2.2), stats, utils

Suggests BondValuation, ggplot2, knitr, limSolve (>= 2.0.1), lubridate, MASS, Matrix, R.cache, RColorBrewer, reshape2, rmarkdown, roxygen2, stringr, testthat (>= 3.0.0), treasury

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

Config/testthat/edition 3

Encoding UTF-8

LazyData true

RoxygenNote 8.0.0

NeedsCompilation no

Author Brian K. Boonstra [aut, cre]

Maintainer Brian K. Boonstra <ragtop@boonstra.org>

Repository CRAN

Date/Publication 2026-07-08 21:40:02 UTC

Contents

accelerated_coupon_value . . . 3
adjust_for_dividends . . . 4
american . . . 5
AmericanOption-class . . . 7
american_implied_volatility . . . 7
blackscholes . . . 9
black_scholes_on_term_structures . . . 10
CALL . . . 12
CallableBond-class . . . 13
check_discount_factor_fcn . . . 13
check_survival_probability_fcn . . . 14
check_variance_cumulation_fcn . . . 15
construct_implicit_grid_structure . . . 15
construct_tridiagonals . . . 17
control_variate_pairs . . . 18
ConvertibleBond-class . . . 18
CouponBond-class . . . 19
coupon_value_at_exercise . . . 19
detail_from_AnnivDates . . . 20
EquityOption-class . . . 21
equivalent_bs_vola_to_jump . . . 22
equivalent_jump_vola_to_bs . . . 23
EuropeanOption-class . . . 25
find_greeks . . . 25
find_present_value . . . 28
fit_to_option_market . . . 30
fit_to_option_market_df . . . 32
fit_variance_cumulation . . . 33
form_present_value_grid . . . 35
GREEK_NAMES . . . 37
GridPricedInstrument-class . . . 37
implied_jump_process_volatility . . . 38
implied_volatilities . . . 39
implied_volatilities_with_rates_struct . . . 41
implied_volatility . . . 42
implied_volatility_with_term_struct . . . 44
infer_conforming_time_grid . . . 46
integrate_pde . . . 47
is.blank . . . 48
iterate_grid_from_timestep . . . 48
pde_matrix_solve . . . 50
penalty_with_intensity_link . . . 51
price_with_intensity_link . . . 53
PUT . . . 54
shift_for_dividends . . . 54
spot_to_df_fcn . . . 55

take_implicit_timestep 55
timestep_instruments 57
time_adj_dividends 58
TIME_RESOLUTION_FACTOR 59
TIME_RESOLUTION_SIGNIF_DIGITS 59
treasury_df 59
treasury_df_raw 60
TSLAMarket 60
value_from_prior_coupons 61
variance_cumulation_from_vols 61
warn_once_negative_default_intensity 62
ZeroCouponBond-class 63

Index **64**

`accelerated_coupon_value` *Present value of coupons according to an acceleration schedule*

Description

Compute "present" value as of time t for coupons that would otherwise have been paid up to time acceleration_t, in the case of accelerated coupon provisions for forced conversions (or sometimes even unforced ones).

Usage

```
accelerated_coupon_value(
  t,
  coupons_df,
  discount_factor_fcn,
  acceleration_t = Inf
)
```

Arguments

`t`	The time toward which all coupons should be present valued
`coupons_df`	A data.frame of details for each coupon. It should have the columns `payment_time` and `payment_size`.
`discount_factor_fcn`	A function specifying how the contract says future coupons should be discounted for this instrument in case the acceleration clause is triggered
`acceleration_t`	Time limit, up to which coupons will be accelerated

See Also

Other Bond Coupons: `coupon_value_at_exercise()`, `value_from_prior_coupons()`

Other Bond Coupon Acceleration: `coupon_value_at_exercise()`

`adjust_for_dividends` *Find the sum of time-adjusted dividend values and adjust grid prices according to their size in the given interval*

Description

Analyze `dividends` to find ones paid in the interval `(t,t+dt]`. Form present value as of time `t` for them, and then use spline interpolation to adjust instrument values accordingly.

Usage

```
adjust_for_dividends(grid_values, t, dt, r, h, S, S0, dividends)
```

Arguments

`grid_values`	A `matrix` with one row for each level of `S` and one column per set of `S`-associated instrument values
`t`	Time after this timestep has been taken
`dt`	Interval to end of timestep
`r`	risk-free interest rate
`h`	Default intensities
`S`	Underlying equity values for the grid
`S0`	Time zero price of the base equity
`dividends`	A `data.frame` with columns `time`, `fixed`, and `proportional`. Dividend size at the given `time` is then expected to be equal to `fixed + proportional * S / S0`

Value

An object like `grid_values` with entries modified according to the dividends

See Also

Other Dividends: `shift_for_dividends()`, `time_adj_dividends()`

`american` *Price one or more american-exercise options*

Description

Use a control-variate scheme to simultaneously estimate the present values of a collection of one or more American-exercise options under a default model with survival probabilities not linked to equity prices.

Usage

```
american(
  callput,
  S0,
  K,
  time,
  const_short_rate = 0,
  const_default_intensity = 0,
  discount_factor_fcn = function(T, t, ...) {
     exp(-const_short_rate * (T - t))
 },
  survival_probability_fcn = function(T, t, ...) {
     exp(-const_default_intensity * (T
    - t))
 },
  default_intensity_fcn = function(t, S, ...) {
     const_default_intensity + 0 * S
 },
  ...,
  num_time_steps = 100,
  structure_constant = 2,
  std_devs_width = 5
)
```

Arguments

`callput`	1 for calls, -1 for puts (may be a vector of the same)
`S0`	initial underlying price
`K`	strike (may be a vector)
`time`	Time from 0 until expiration (may be a vector)
`const_short_rate`	A constant to use for the instantaneous interest rate in case `discount_factor_fcn` is not given
`const_default_intensity`	A constant to use for the instantaneous default intensity in case `default_intensity_fcn` is not given

`discount_factor_fcn`
: A function for computing present values to time `t` of various cashflows occurring during this timestep, with arguments `T`, `t`

`survival_probability_fcn`
: (Implied argument) A function for probability of survival, with arguments `T`, `t` and `T>t`. E.g. with a constant volatility s this takes the form $(T-t)s^2$. Should be matched to `default_intensity_fcn`

`default_intensity_fcn`
: A function for computing default intensity occurring at a given time, dependent on time and stock price, with arguments `t`, `S`. Should be matched to `survival_probability_fcn`

`...`
: Further arguments passed on to `find_present_value`

`num_time_steps`
: Number of steps to use in the grid solver. Can usually be set quite low due to the control variate scheme.

`structure_constant`
: The maximum ratio between time intervals `dt` and the square of space intervals `dz^2`

`std_devs_width`
: The number of standard deviations, in `sigma * sqrt(T)` units, to incorporate into the grid

Details

The scheme uses find_present_value() to price the options and their European-exercise equivalents. It then compares the latter to black-scholes formula output and uses the results as an error correction on the prices of the American-exercise options.

Value

A vector of estimated option present values

See Also

Other Equity Independent Default Intensity: `american_implied_volatility()`, `black_scholes_on_term_structures()` `blackscholes()`, `equivalent_bs_vola_to_jump()`, `equivalent_jump_vola_to_bs()`, `implied_volatilities()`, `implied_volatilities_with_rates_struct()`, `implied_volatility()`, `implied_volatility_with_term_struct()`

Other American Exercise Equity Options: `american_implied_volatility()`, `control_variate_pairs()`

Examples

```
american(PUT, S0=100, K=110, time=0.77, const_short_rate = 0.06,
         const_volatility=0.20, num_time_steps=200)
american(callput=-1, S0=100, K=90, time=1, const_short_rate=0.025,
         variance_cumulation_fcn = function(T, t) {  # Term structure of vola
             0.45 ^ 2 * (T - t) + 0.15^2 * max(0, T-0.25)
         })
```

AmericanOption-class *A standard option contract allowing for* early *exercise at the choice of the option holder*

Description

A standard option contract allowing for *early* exercise at the choice of the option holder

Methods

`optionality_fcn(v, ...)` Return a version of `v` at time `t` corrected for any optionality conditions.

`recovery_fcn(v, S, t, ...)` Return recovery value, given non-default values `v` at time `t`. Subclasses may be more elaborate, this method simply returns 0.0.

american_implied_volatility *Implied volatility of an american option with equity-independent term structures*

Description

Use the grid solver to generate american option values under a default model with survival probabilities not linked to equity prices. and run them through a bisective root search method until a constant volatility matching the provided option price has been found.

Usage

```
american_implied_volatility(
  option_price,
  callput,
  S0,
  K,
  time,
  const_default_intensity = 0,
  survival_probability_fcn = function(T, t, ...) {
     exp(-const_default_intensity * (T
    - t))
 },
  default_intensity_fcn = function(t, S, ...) {
     const_default_intensity + 0 * S
 },
  ...,
  num_time_steps = 30,
  structure_constant = 2,
```

```
  std_devs_width = 5,
  relative_tolerance = 1e-04,
  max.iter = 100,
  max_vola = 4
)
```

Arguments

option_price	Option price to match
callput	1 for calls, -1 for puts
S0	An initial stock price, for setting grid scale
K	strike
time	Time from 0 until expiration
const_default_intensity	A constant to use for the instantaneous default intensity in case default_intensity_fcn is not given
survival_probability_fcn	(Implied argument) A function for probability of survival, with arguments T, t and T>t.
default_intensity_fcn	A function for computing default intensity occurring at a given time, dependent on time and stock price, with arguments t, S. Should be matched to survival_probability_fcn
...	Additional arguments to be passed on to implied_volatility_with_term_struct and american
num_time_steps	Minimum number of time steps in the grid
structure_constant	The maximum ratio between time intervals dt and the square of space intervals dz^2
std_devs_width	The number of standard deviations, in sigma * sqrt(T) units, to incorporate into the grid
relative_tolerance	Relative tolerance in instrument price defining the root-finder halting condition
max.iter	Maximum number of root-finder iterations allowed
max_vola	Maximum volatility to try

Value

Estimated volatility

See Also

implied_volatility_with_term_struct for implied volatility of European options under the same conditions, american for the underlying pricing algorithm

Other Implied Volatilities: equivalent_bs_vola_to_jump(), equivalent_jump_vola_to_bs(), fit_variance_cumulation(), implied_jump_process_volatility(), implied_volatilities(), implied_volatilities_with_rates_struct(), implied_volatility(), implied_volatility_with_term_struct()

Other Equity Independent Default Intensity: american(), black_scholes_on_term_structures(), blackscholes(), equivalent_bs_vola_to_jump(), equivalent_jump_vola_to_bs(), implied_volatilities(), implied_volatilities_with_rates_struct(), implied_volatility(), implied_volatility_with_term_struct()

Other American Exercise Equity Options: american(), control_variate_pairs()

Examples

```
american_implied_volatility(25,CALL,S0=100,K=100,time=2.2,
  const_short_rate=0.03, num_time_steps=5)
df250 =  function(t) ( exp(-0.02*t)*exp(-0.03*max(0,t-1.0))) # Simple term structure
df25 = function(T,t){df250(T)/df250(t)} # Relative discount factors
american_implied_volatility(25,-1,100,100,2.2,
  discount_factor_fcn=df25, num_time_steps=5)
```

---

blackscholes *Vectorized Black-Scholes pricing of european-exercise options*

Description

Price options according to the famous Black-Scholes formula, with the optional addition of a jump-to-default intensity and discrete dividends.

Usage

```
blackscholes(
  callput,
  S0,
  K,
  r,
  time,
  vola,
  default_intensity = 0,
  divrate = 0,
  borrow_cost = 0,
  dividends = NULL
)
```

Arguments

callput	1 for calls, -1 for puts
S0	initial underlying price
K	strike
r	risk-free interest rate
time	Time from 0 until expiration
vola	Default-free volatility of the underlying

default_intensity
: hazard rate of underlying default

divrate
: A continuous rate for dividends and other cashflows such as foreign interest rates

borrow_cost
: A continuous rate for stock borrow costs

dividends
: A data.frame with columns time, fixed, and proportional. Dividend size at the given time is then expected to be equal to fixed + proportional * S / S0. Fixed dividends will be converted to proportional for purposes of this algorithm.

Details

Note that if the default_intensity is set larger than zero then put-call parity still holds. Greeks are reduced according to cumulated default probability.

All inputs must either be scalars or have the same nonscalar shape.

Value

A list with elements

Price The present value(s)

Delta Sensitivity to underlying price

Vega Sensitivity to volatility

See Also

Other European Options: black_scholes_on_term_structures(), implied_volatilities(), implied_volatilities_with_rates_struct(), implied_volatility(), implied_volatility_with_term_struct()

Other Equity Independent Default Intensity: american(), american_implied_volatility(), black_scholes_on_term_structures(), equivalent_bs_vola_to_jump(), equivalent_jump_vola_to_bs(), implied_volatilities(), implied_volatilities_with_rates_struct(), implied_volatility(), implied_volatility_with_term_struct()

Examples

```
blackscholes(callput=-1, S0=100, K=90, r=0.03, time=1, # -1 is a PUT
             vola=0.5, default_intensity=0.07)
```

---

black_scholes_on_term_structures

Black-Scholes pricing of european-exercise options with term structure arguments

---

Description

Price an option according to the famous Black-Scholes formula, with the optional addition of a jump-to-default intensity and discrete dividends. Volatility and rates may be provided as constants or as 2+ parameter functions with first argument T corresponding to maturity and second argument t corresponding to model date.

Usage

```
black_scholes_on_term_structures(
  callput,
  S0,
  K,
  time,
  const_volatility = 0.5,
  const_short_rate = 0,
  const_default_intensity = 0,
  discount_factor_fcn = function(T, t, ...) {
     exp(-const_short_rate * (T - t))
 },
  survival_probability_fcn = function(T, t, ...) {
     exp(-const_default_intensity * (T
    - t))
 },
  variance_cumulation_fcn = function(T, t) {
     const_volatility^2 * (T - t)
 },
  dividends = NULL,
  borrow_cost = 0,
  dividend_rate = 0
)
```

Arguments

`callput`	`1` for calls, `-1` for puts
`S0`	initial underlying price
`K`	strike
`time`	Time from `0` until expiration
`const_volatility`	A constant to use for volatility in case `variance_cumulation_fcn` is not given
`const_short_rate`	A constant to use for the instantaneous interest rate in case `discount_factor_fcn` is not given
`const_default_intensity`	A constant to use for the instantaneous default intensity in case `default_intensity_fcn` is not given
`discount_factor_fcn`	A function for computing present values to time `t` of various cashflows occurring during this timestep, with arguments `T`, `t`
`survival_probability_fcn`	A function for probability of survival, with arguments `T`, `t` and `T>t`. E.g. with a constant volatility s this takes the form $(T-t)s^2$.

`variance_cumulation_fcn`
: A function for computing total stock variance occurring during this timestep, with arguments `T`, `t`. E.g. with a constant volatility s this takes the form $(T-t)s^2$.

`dividends`
: A `data.frame` with columns `time`, `fixed`, and `proportional`. Dividend size at the given `time` is then expected to be equal to `fixed + proportional * S / S0`. Fixed dividends will be converted to proportional for purposes of this algorithm.

`borrow_cost`
: A continuous rate for stock borrow costs

`dividend_rate`
: A continuous rate for dividends and other cashflows such as foreign interest rates

Details

Any term structures will be converted to equivalent constant arguments by calling them with the arguments `(time, 0)`.

See Also

Other European Options: `blackscholes()`, `implied_volatilities()`, `implied_volatilities_with_rates_struct()`, `implied_volatility()`, `implied_volatility_with_term_struct()`

Other Equity Independent Default Intensity: `american()`, `american_implied_volatility()`, `blackscholes()`, `equivalent_bs_vola_to_jump()`, `equivalent_jump_vola_to_bs()`, `implied_volatilities()`, `implied_volatilities_with_rates_struct()`, `implied_volatility()`, `implied_volatility_with_term_struct()`

Examples

```
black_scholes_on_term_structures(callput=-1, S0=100, K=90, time=1,
                                 discount_factor_fcn = function(T, t, ...) {
                                   exp(-0.03 * (T - t))
                                 },
                                 survival_probability_fcn = function(T, t, ...) {
                                   exp(-0.07 * (T - t))
                                 },
                                 variance_cumulation_fcn = function(T, t) {
                                   0.45 ^ 2 * (T - t)
                                 })
```

CALL *Constant CALL for defining option contracts*

Description

Constant CALL for defining option contracts

Usage

```
CALL
```

CallableBond-class *Callable (and putable) corporate or government bond.*

Description

When a bond is *callable*, the issuer may choose to pay the call price to the bond holder and end the life of the contract.

Details

When a bond is *putable*, the bond holder may choose to force the issuer pay the put price to the bond holder thus ending the life of the contract.

Fields

`calls` A data.frame of details for each call. It should have the columns `call_price` and `effective_time`.

`puts` A data.frame of details for each put. It should have the columns `put_price` and `effective_time`.

Methods

`critical_times()` Important times in the life of this instrument for simulation and grid solvers

check_discount_factor_fcn *Check that a discount factor function is internally consistent*

Description

Ensure discount factor function satisfies the time-reversal identity

$$df(t, T) = 1/df(T, t)$$

. Emits a warning, whenever that identity is violated, returns TRUE if no problems are found.

Usage

```
check_discount_factor_fcn(discount_factor_fcn, time_pts, tolerance = 1e-06)
```

Arguments

`discount_factor_fcn`	A function for computing present values, with arguments `T`, `t`, returning the discount factor from time `T` back to time `t`
`time_pts`	A vector of time points at which to test the consistency of `discount_factor_fcn`
`tolerance`	Relative tolerance within which the reversal identity is considered to hold

Details

Discount factors that have underflowed to zero or overflowed to infinity carry no usable information about the reversed factor, so such pairs are silently ignored rather than flagged.

Value

`TRUE` if the discount factor function appears consistent at every tested pair, otherwise `FALSE`

See Also

Other Input Checking: `check_survival_probability_fcn()`, `check_variance_cumulation_fcn()`, `warn_once_negative_default_intensity()`

`check_survival_probability_fcn` *Check that a survival probability function is well-behaved*

Description

Survival probabilities accumulated from a fixed origin can never increase as the horizon moves forward in time, and must always lie in the closed interval $[0, 1]$. This routine evaluates the supplied `survival_probability_fcn` on the ordered `time_pts`. Emits a warning whenever the probabilities are increasing or fall outside $[0, 1]$.

Usage

```
check_survival_probability_fcn(survival_probability_fcn, time_pts)
```

Arguments

`survival_probability_fcn`	A function for computing survival probabilities, with arguments `T`, `t`, returning the probability of survival from time `t` to time `T`
`time_pts`	A vector of time points at which to test the behavior of `survival_probability_fcn`

Value

`TRUE` if the survival probabilities are nonincreasing and within $[0, 1]$ across `time_pts`, otherwise `FALSE`

See Also

Other Input Checking: `check_discount_factor_fcn()`, `check_variance_cumulation_fcn()`, `warn_once_negative_default_intensity()`

`check_variance_cumulation_fcn` *Check that a variance cumulation function is nondecreasing*

Description

Total variance accumulated from a fixed origin can never decrease as the horizon moves forward in time. This routine evaluates the supplied `variance_cumulation_fcn` on the ordered `time_pts`. Emits a warning whenever the cumulated variance fails to be nondecreasing.

Usage

```
check_variance_cumulation_fcn(variance_cumulation_fcn, time_pts)
```

Arguments

`variance_cumulation_fcn`	A function for computing total stock variance, with arguments `T`, `t`, returning the variance cumulated from time `t` to time `T`
`time_pts`	A vector of time points at which to test that `variance_cumulation_fcn` is nondecreasing

Value

`TRUE` if the cumulated variance is nondecreasing across `time_pts`, otherwise `FALSE`

See Also

Other Input Checking: `check_discount_factor_fcn()`, `check_survival_probability_fcn()`, `warn_once_negative_default_intensity()`

`construct_implicit_grid_structure` *Structure of implicit numerical integration grid*

Description

Infer a reasonable structure for our implicit grid solver based on the voltime, structure constant, and requested grid width in standard deviations.

Usage

```
construct_implicit_grid_structure(
  tenors,
  M,
  S0,
  K,
  c,
  sigma,
  structure_constant,
  std_devs_width,
  min_z_width = 0
)
```

Arguments

`tenors`	Tenors of instruments to be treated on this grid
`M`	Minimum number of timesteps on this grid
`S0`	An initial stock price, for setting grid scale
`K`	An instrument reference stock price, for setting grid scale
`c`	A continuous stock drift rate
`sigma`	Volatility of diffusion process (without jumps to default)
`structure_constant`	The maximum ratio between time intervals `dt` and the square of space intervals `dz^2`
`std_devs_width`	The number of standard deviations, in `sigma * sqrt(T)` units, to incorporate into the grid
`min_z_width`	Minimum grid width, in log space

Details

Generally speaking pricing will be good to about 10bp of relative accuracy when the ratio of timesteps to voltime (in annualized units) is over 200.

Cases with pathologically low volatility may go awry (in the sense of yielding ultimately inaccurate PDE solutions), as the `structure_constant` will force a step in `z` space much bigger than the width in standard deviations.

Value

A list with elements

`T` The maximum time for this grid

`dt` Largest permissible timestep size

`dz` Distance between space grid points

`z0` Center of space grid

`z_width` Width in z space

`half_N` A misnomer, actually $(N-1)/2$

`N` The number of space points

`z` Locations of space points

See Also

Other Implicit Grid Solver: `find_present_value()`, `form_present_value_grid()`, `infer_conforming_time_grid()`, `integrate_pde()`, `iterate_grid_from_timestep()`, `pde_matrix_solve()`, `take_implicit_timestep()`, `timestep_instruments()`

---

construct_tridiagonals

Matrix entries for implicit numerical differentiation using Neumann boundary conditions

---

Description

Matrix entries for implicit numerical differentiation using Neumann boundary conditions

Usage

```
construct_tridiagonals(sigma, structure_constant, drift)
```

Arguments

`sigma`	Volatility of diffusion process (without jumps to default)
`structure_constant`	The ratio between time interval `dt` and the square of space interval `dz^2`
`drift`	Vector of drift rate of underlying equity grid points, including induced drift from default intensity

Value

A list with elements `super`, `diag` and `sub` containing the superdiagonal, diagonal and subdiagonal of the implicit timestep differencing matrix

control_variate_pairs *Form instrument objects for vanilla options*

Description

Form a list twice as long as the longest of the arguments `callput`, `K`, `time` whose first half consists of AmericanOption objects and second half consists of EuropeanOption objects having the same exercise specification

Usage

```
control_variate_pairs(callput, K, time)
```

Arguments

`callput`	`1` for calls, `-1` for puts
`K`	strike
`time`	Time from `0` until expiration

See Also

Other American Exercise Equity Options: `american()`, `american_implied_volatility()`

ConvertibleBond-class *Convertible bond with exercise into stock*

Description

Convertible bond with exercise into stock

Fields

`conversion_ratio` The number of shares, per bond, that result from exercise

`dividend_ceiling` The level of dividend protection (if any) specified in terms and conditions

Methods

`exercise_decision(v, S, t, discount_factor_fctn = discount_factor_fcn, ...)` Find indexes where hold value `v` will be inferior to conversion value at each stock price level in `S`, adjusted to include all past coupons

`optionality_fcn(v, S, t, discount_factor_fctn = discount_factor_fcn, ...)` Return the greater of hold value `v` or exercise value at each stock price level in `S`. If the given date is beyond maturity, return value at maturity.

`reset_caches()` Clear the accrued-coupon cache in addition to inherited caches.

terminal_values(v, ...) Return a terminal value. defaults to simply calling optionality_fcn.

update_cashflows(small_t, big_t, discount_factor_fctn = discount_factor_fcn, include_notional = TRUE, . Update last_computed_cash and return cashflow information for the given time period, valued at big_t

---

CouponBond-class *Standard corporate or government bond*

Description

A coupon bond is treated here as the entire collection of cashflows. In particular, coupons are included in the package even after they have been paid, accruing at the risk-free rate.

Fields

coupons A data.frame of details for each coupon. It should have the columns payment_time and payment_size.

Methods

accumulate_coupon_values_before(t, discount_factor_fctn = discount_factor_fcn) Compute the sum of coupon present values as of t according to discount_factor_fctn

critical_times() Important times in the life of this instrument for simulation and grid solvers

optionality_fcn(v, S, t, ...) Return the notional value in the shape of S at any time on or after maturity, otherwise just return v

reset_caches() Clear any memoization caches so this instrument can be repriced afresh. Subclasses extend this to clear their own caches.

total_coupon_values_between(small_t, big_t, discount_factor_fctn = discount_factor_fcn) Compute the sum (as of big_t) of present values of coupons paid between small_t and big_t

update_cashflows(small_t, big_t, discount_factor_fctn = discount_factor_fcn, include_notional = TRUE, . Update last_computed_cash and return cashflow information for the given time period, valued at big_t

---

coupon_value_at_exercise

Present value of coupons according to an acceleration schedule

---

Description

Compute "present" value as of time t for coupons that would otherwise have been paid up to time acceleration_t, in the case of accelerated coupon provisions for forced conversions (or sometimes even unforced ones).

Usage

```
coupon_value_at_exercise(
  t,
  coupons_df,
  discount_factor_fcn,
  model_t = 0,
  accelerate_future_coupons = FALSE,
  acceleration_discount_factor_fcn = discount_factor_fcn,
  acceleration_t = Inf
)
```

Arguments

t	The time toward which all coupons should be present valued
coupons_df	A data.frame of details for each coupon. It should have the columns payment_time and payment_size.
discount_factor_fcn	A function specifying how future cashflows should generally be discounted for this instrument
model_t	Model timestamp passed to value_from_prior_coupons
accelerate_future_coupons	If TRUE, future coupons will be accelerated on exercise to pad present value
acceleration_discount_factor_fcn	A function specifying how future coupons should be discounted for this instrument under coupon acceleration conditions
acceleration_t	The maximum time up to which future coupons will be counted for acceleration, passed on to accelerated_coupon_value

Value

A scalar equal to the present value

See Also

Other Bond Coupons: accelerated_coupon_value(), value_from_prior_coupons()

Other Bond Coupon Acceleration: accelerated_coupon_value()

---

detail_from_AnnivDates

Convert output of BondValuation::AnnivDates to inputs for Bond

---

Description

The BondValuation package provides day count convention treatments superior to quantmod or any other R package known (as of May 2019). This function takes output from BondValuation::AnnivDates(...) and parses it into notionals, maturity time, and coupon times and sizes.

Usage

```
detail_from_AnnivDates(
  anvdates,
  as_of = Sys.time(),
  normalization_factor = 365.25
)
```

Arguments

anvdates	Output of BondValuation::AnnivDates(), which must have included a 'Coup' argument so that the resulting list contains an entry for 'PaySched'
as_of	Date or time from whose perspective times should be computed
normalization_factor	Factor by which raw R time differences should be multiplied. If volatilites are going to be annualized, then this should typically be 365 or so.

Details

Note: volatilities used in 'ragtop' must have compatible time units to these times.

Value

A list with some of the arguments appropriate for defining a Bond as follows: maturity - maturity notional - notional amount coupons - 'data.frame' with 'payment_time', 'payment_size'

EquityOption-class *An option contract with call or put terms*

Description

An option contract with call or put terms

Fields

`strike` A decision price for the contract

`callput` Either `1` for a call or `-1` for a put

`equivalent_bs_vola_to_jump` *Find straight Black-Scholes volatility equivalent to jump process with a given default risk*

Description

Find Black-Scholes volatility based on known interest rates and hazard rates, using an at-the-money put option at the given tenor to set the standard price.

Usage

```
equivalent_bs_vola_to_jump(
  jump_process_vola,
  time,
  const_short_rate = 0,
  const_default_intensity = 0,
  discount_factor_fcn = function(T, t, ...) {
     exp(-const_short_rate * (T - t))
 },
  survival_probability_fcn = function(T, t, ...) {
     exp(-const_default_intensity * (T
    - t))
 },
  dividends = NULL,
  borrow_cost = 0,
  dividend_rate = 0,
  relative_tolerance = 1e-06,
  max.iter = 100
)
```

Arguments

`jump_process_vola`
: Volatility of default-free process

`time`
: Time to expiration of associated option contracts

`const_short_rate`
: A constant to use for the instantaneous interest rate in case `discount_factor_fcn` is not given

`const_default_intensity`
: A constant to use for the instantaneous default intensity in case `survival_probability_fcn` is not given

`discount_factor_fcn`
: A function for computing present values to time `t` of various cashflows occurring during this timestep, with arguments `T`, `t`

survival_probability_fcn
: (Implied argument) A function for probability of survival, with arguments `T`, `t` and `T>t`.

dividends
: A `data.frame` with columns `time`, `fixed`, and `proportional`. Dividend size at the given `time` is then expected to be equal to `fixed + proportional * S / S0`. Fixed dividends will be converted to proportional for purposes of this algorithm.

borrow_cost
: A continuous rate for stock borrow costs

dividend_rate
: A continuous accumulation rate for the stock, affecting the drift

relative_tolerance
: Relative tolerance in instrument price defining the root-finder halting condition

max.iter
: Maximum number of root-finder iterations allowed

Value

A scalar defaultable volatility of an option

See Also

Other Implied Volatilities: `american_implied_volatility()`, `equivalent_jump_vola_to_bs()`, `fit_variance_cumulation()`, `implied_jump_process_volatility()`, `implied_volatilities()`, `implied_volatilities_with_rates_struct()`, `implied_volatility()`, `implied_volatility_with_term_struct()`

Other Equity Independent Default Intensity: `american()`, `american_implied_volatility()`, `black_scholes_on_term_structures()`, `blackscholes()`, `equivalent_jump_vola_to_bs()`, `implied_volatilities()`, `implied_volatilities_with_rates_struct()`, `implied_volatility()`, `implied_volatility_with_term_struct()`

---

equivalent_jump_vola_to_bs

Find jump process volatility with a given default risk from a straight Black-Scholes volatility

---

Description

Find default-free volatility (i.e. volatility of a Wiener process with a companion jump process to default) based on known interest rates and hazard rates, using and at-the-money put option at the given tenor to set the standard price.

Usage

```
equivalent_jump_vola_to_bs(
  bs_vola,
  time,
  const_short_rate = 0,
  const_default_intensity = 0,
  discount_factor_fcn = function(T, t, ...) {
     exp(-const_short_rate * (T - t))
```

```
 },
  survival_probability_fcn = function(T, t, ...) {
     exp(-const_default_intensity * (T
    - t))
 },
  dividends = NULL,
  borrow_cost = 0,
  dividend_rate = 0,
  relative_tolerance = 1e-06,
  max.iter = 100
)
```

Arguments

bs_vola	BlackScholes volatility of an option with no default assumption
time	Time to expiration of associated option contracts
const_short_rate	A constant to use for the instantaneous interest rate in case discount_factor_fcn is not given
const_default_intensity	A constant to use for the instantaneous default intensity in case survival_probability_fcn is not given
discount_factor_fcn	A function for computing present values to time t of various cashflows occurring during this timestep, with arguments T, t
survival_probability_fcn	(Implied argument) A function for probability of survival, with arguments T, t and T>t.
dividends	A data.frame with columns time, fixed, and proportional. Dividend size at the given time is then expected to be equal to fixed + proportional * S / S0
borrow_cost	Stock borrow cost, affecting the drift rate
dividend_rate	A continuous accumulation rate for the stock, affecting the drift
relative_tolerance	Relative tolerance in instrument price defining the root-finder halting condition
max.iter	Maximum number of root-finder iterations allowed

Value

A scalar volatility

See Also

Other Implied Volatilities: american_implied_volatility(), equivalent_bs_vola_to_jump(), fit_variance_cumulation(), implied_jump_process_volatility(), implied_volatilities(), implied_volatilities_with_rates_struct(), implied_volatility(), implied_volatility_with_term_struct()

Other Equity Independent Default Intensity: american(), american_implied_volatility(), black_scholes_on_term_structures(), blackscholes(), equivalent_bs_vola_to_jump(), implied_volatilities(), implied_volatilities_with_rates_struct(), implied_volatility(), implied_volatility_with_term_struct()

EuropeanOption-class *A standard option contract*

Description

At maturity, the call option holder will "exercise", i.e. choose stock, with value S, if the stock price is above the strike K, paying K to the option issuer, realizing value S-K. The put option holder will exercise, receiving K while surrendering stock worth S, if the stock price is below K.

Details

Therefore the value at maturity is equal to max(0,callput*(S-K))

Methods

optionality_fcn(v, ...) Return a version of v at time t corrected for any optionality conditions.

recovery_fcn(v, S, t, ...) Return recovery value, given non-default values v at time t. Subclasses may be more elaborate, this method simply returns 0.0.

---

find_greeks *Find present value and selected greeks for given set of instruments*

Description

Price the given instruments using find_present_value, and also compute the specified greeks via finite-difference calculations. The finite differences are higher-order 2-sided estimates where feasible, otherwise they are 1-sided estimates. Bump sizes are automatically chosen by resolve_bumps, unless the corresponding dS, dr, dvola or dhazard override has been given a non-NULL value.

Usage

```
find_greeks(
  S0,
  num_time_steps,
  instruments,
  greeks = c("delta", "gamma"),
  const_volatility = 0.5,
  const_short_rate = 0,
  const_default_intensity = 0,
```

```
  discount_factor_fcn = function(T, t, ...) {
     exp(-const_short_rate * (T - t))
 },
  default_intensity_fcn = function(t, S, ...) {
     const_default_intensity + 0 * S
 },
  variance_cumulation_fcn = function(T, t) {
     const_volatility^2 * (T - t)
 },
  dvola = NULL,
  dr = NULL,
  dhazard = NULL,
  ...
)
```

Arguments

S0	An initial stock price, for setting grid scale
num_time_steps	Minimum number of time steps in the grid
instruments	A list of instruments to be priced. Each one must have a strike and a optionality_fcn, as with GridPricedInstrument and its subclasses.
greeks	Sequence of names of greeks to be estimated (see GREEK_NAMES for the available values). Delta and Gamma are essentially zero-cost to include since they are read directly off the pricing grid as spline derivatives. All other greeks require repeated calls to the pricing engine and multiply computation expense accordingly.
const_volatility	A constant to use for volatility in case variance_cumulation_fcn is not given
const_short_rate	A constant to use for the instantaneous interest rate in case discount_factor_fcn is not given
const_default_intensity	A constant to use for the instantaneous default intensity in case default_intensity_fcn is not given
discount_factor_fcn	A function for computing present values to time t of various cashflows occurring during this timestep, with arguments T, t
default_intensity_fcn	A function for computing default intensity occurring during this timestep, dependent on time and stock price, with arguments t, S.
variance_cumulation_fcn	A function for computing total stock variance occurring during this timestep, with arguments T, t. E.g. with a constant volatility s this takes the form $(T - t)s^2$.
dvola	User-specified change in volatility to use for Vega estimation by multiple pricing calls. Usually left NULL, for automatic selection.

dr — User-specified change in risk-free rates to use for Rates DV01 estimation by multiple pricing calls. Usually left NULL, for automatic selection.

dhazard — User-specified change in default intensity to use for Credit DV01 estimation by multiple pricing calls. Usually left NULL, for automatic selection.

... — Arguments passed on to find_present_value

grid_center A reasonable central value for the grid, defaults to S0 or an instrument strike

borrow_cost Stock borrow cost, affecting the drift rate

dividend_rate Continuous dividend rate, affecting the drift rate

override_Tmax A different maximum time on the grid to enforce

structure_constant The maximum ratio between time intervals dt and the square of space intervals dz^2

std_devs_width The number of standard deviations, in sigma * sqrt(T) units, to incorporate into the grid

dividends A data.frame with columns time, fixed, and proportional. Dividend size at the given time is then expected to be equal to fixed + proportional * S / S0

Details

If interest rates, default intensity, or volatility is zero then the corresponding greek will *not* be computed without a specifically-chosen user override for dr, dvola or dhazard.

Bump sizes ultimately used, and central/one-sided estimate status, are available as as attributes() of the result

Value

A list, with at least one entry for price resulting from find_present_value(). For each greek requested, the list will have another entry.

See Also

Other Greeks: GREEK_NAMES, construct_descending_bumps(), greek_by_fd(), grid_delta_gamma(), resolve_bumps(), robust_greek(), safe_reprice()

Examples

```
S0=241.80
varc = variance_cumulation_from_vols(
  data.frame(
    time=c(0.13, 0.38, 0.63, 0.72, 1.72),
    volatility=c(0.48, 0.46, 0.46, 0.45, 0.45)
  ))
disct_fcn = spot_to_df_fcn(
  data.frame(time=c(1, 5, 10, 15),rate=c(0.01, 0.02, 0.03, 0.05))
)
cb = ConvertibleBond(
maturity=2.87, conversion_ratio=2.7788, notional=1000,
```

```
coupons=data.frame(
payment_time=seq(2.8, 0, by=-0.25),
    payment_size=1000*0.0025/4),
  discount_factor_fcn = disct_fcn
)
price_and_greeks = find_greeks(
  greeks=c("delta", "vega", "credit_dv01"),
  S0=S0,
instruments=list(CB=cb),
  num_time_steps=55,
  default_intensity_fcn = function(t, S, ...){0.03 + 0.01 * (S0/S)^1.5},
  discount_factor_fcn = disct_fcn,
  variance_cumulation_fcn=varc)
round(unlist(price_and_greeks),4)
```

`find_present_value` *Use a model to estimate the present value of financial derivatives*

Description

Use a finite difference scheme to form estimates of present values for a variety of stock prices. Once the grid has been created, interpolate to obtain the value of each instrument at the present stock price `S0`

Usage

```
find_present_value(
  S0,
  num_time_steps,
  instruments,
  const_volatility = 0.5,
  const_short_rate = 0,
  const_default_intensity = 0,
  override_Tmax = NA,
  discount_factor_fcn = function(T, t, ...) {
     exp(-const_short_rate * (T - t))
 },
  default_intensity_fcn = function(t, S, ...) {
     const_default_intensity + 0 * S
 },
  variance_cumulation_fcn = function(T, t) {
     const_volatility^2 * (T - t)
 },
  dividends = NULL,
  borrow_cost = 0,
  dividend_rate = 0,
  structure_constant = 2,
  std_devs_width = 3,
```

```
  grid_center = NA
)
```

Arguments

`S0`	An initial stock price, for setting grid scale
`num_time_steps`	Minimum number of time steps in the grid
`instruments`	A list of instruments to be priced. Each one must have a `strike` and a `optionality_fcn`, as with `GridPricedInstrument` and its subclasses.
`const_volatility`	A constant to use for volatility in case `variance_cumulation_fcn` is not given
`const_short_rate`	A constant to use for the instantaneous interest rate in case `discount_factor_fcn` is not given
`const_default_intensity`	A constant to use for the instantaneous default intensity in case `default_intensity_fcn` is not given
`override_Tmax`	A different maximum time on the grid to enforce
`discount_factor_fcn`	A function for computing present values to time `t` of various cashflows occurring during this timestep, with arguments `T`, `t`
`default_intensity_fcn`	A function for computing default intensity occurring during this timestep, dependent on time and stock price, with arguments `t`, `S`.
`variance_cumulation_fcn`	A function for computing total stock variance occurring during this timestep, with arguments `T`, `t`. E.g. with a constant volatility s this takes the form $(T - t)s^2$.
`dividends`	A `data.frame` with columns `time`, `fixed`, and `proportional`. Dividend size at the given `time` is then expected to be equal to `fixed + proportional * S / S0`
`borrow_cost`	Stock borrow cost, affecting the drift rate
`dividend_rate`	Continuous dividend rate, affecting the drift rate
`structure_constant`	The maximum ratio between time intervals `dt` and the square of space intervals `dz^2`
`std_devs_width`	The number of standard deviations, in `sigma * sqrt(T)` units, to incorporate into the grid
`grid_center`	A reasonable central value for the grid, defaults to S0 or an instrument strike

Value

A list of present values, with the same names as `instruments`

See Also

Other Equity Dependent Default Intensity: fit_to_option_market_df(), fit_variance_cumulation(), form_present_value_grid(), implied_jump_process_volatility()

Other Implicit Grid Solver: construct_implicit_grid_structure(), form_present_value_grid(), infer_conforming_time_grid(), integrate_pde(), iterate_grid_from_timestep(), pde_matrix_solve(), take_implicit_timestep(), timestep_instruments()

`fit_to_option_market` *Calibrate volatilities and equity-linked default intensity*

Description

Given derivative instruments (subclasses of GridPricedInstrument, though typically either AmericanOption or EuropeanOption objects), along with their prices and spreads, calibrate variance cumulation (the at-the-money volatility of the continuous process) and equity linked default intensity of the form $h(s + (1-s)(S0/S_t)^p)$.

Usage

```
fit_to_option_market(
  variance_instruments,
  variance_instrument_prices,
  variance_instrument_spreads,
  fit_instruments,
  fit_instrument_prices,
  fit_instrument_spreads,
  fit_instrument_weights,
  S0,
  num_time_steps = 30,
  const_short_rate = 0,
  discount_factor_fcn = function(T, t) {
     exp(-const_short_rate * (T - t))
 },
  ...,
  base_default_intensity = 0.05,
  relative_spread_tolerance = 0.15,
  num_variance_time_steps = 30
)
```

Arguments

`variance_instruments`
: A list of instruments in strictly increasing order of maturity, from which the volatility term structure will be inferred. Once the calibration is finished, the chosen parameters will reproduce the prices of these instruments with fairly high precision.

`variance_instrument_prices`
: Central price targets for the variance instruments

`variance_instrument_spreads`
: Bid-offer spreads used to normalize errors in variance instrument prices during term structure fitting

`fit_instruments`
: A list of instruments in any order, from which the mispricing penalties used for judging fit quality will be computed

`fit_instrument_prices`
: Central price targets for the variance instruments

`fit_instrument_spreads`
: Bid-offer spreads used to normalize errors in fit instrument prices during default intensity

`fit_instrument_weights`
: Weights applied to relative errors in fit instrument prices before summing to form the penalty

`S0`
: Current underlying price

`num_time_steps`
: Time step count passed on to `find_present_value` while fitting instrument values

`const_short_rate`
: A constant to use for the instantaneous interest rate in case `discount_factor_fcn` is not given

`discount_factor_fcn`
: A function for computing present values to time `t` of various cashflows occurring during this timestep, with arguments `T`, `t`

`...`
: Further arguments passed to `penalty_with_intensity_link`

`base_default_intensity`
: Overall default intensity (in natural units)

`relative_spread_tolerance`
: Tolerance to apply in calling `fit_variance_cumulation`

`num_variance_time_steps`
: Number of time steps to use in calling `fit_variance_cumulation`

Details

In its present form, this function uses a brain-dead grid search.

See Also

`penalty_with_intensity_link` for the penalty function used as an optimization target

`fit_to_option_market_df` *Calibrate volatilities and equity-linked default intensity making many assumptions*

Description

This is a convenience function for calibrating variance cumulation (the at-the-money volatility of the continuous process) and equity linked default intensity of the form $h(s + (1-s)(S0/S_t)^p)$, using a `data.frame` of option market data.

Usage

```
fit_to_option_market_df(
  S0 = ragtop::TSLAMarket$S0,
  discount_factor_fcn = spot_to_df_fcn(ragtop::TSLAMarket$risk_free_rates),
  options_df = ragtop::TSLAMarket$options,
  min_maturity = 1/12,
  min_moneyness = 0.8,
  max_moneyness = 1.2,
  base_default_intensity = 0.05
)
```

Arguments

`S0`	Current underlying price
`discount_factor_fcn`	A function for computing present values to time `t` of various cashflows occurring during this timestep, with arguments `T`, `t`
`options_df`	A data frame of American option details. It should have columns `callput`, `K`, `time`, `mid`, `bid`, and `ask`,
`min_maturity`	Minimum option maturity to allow in calibration
`min_moneyness`	Minimum option strike as a proportion of S0 to allow in calibration
`max_moneyness`	Maximum option strike as a proportion of S0 to allow in calibration
`base_default_intensity`	Overall default intensity (in natural units)

See Also

`fit_to_option_market` the underlying fit algorithm

Other Equity Dependent Default Intensity: `find_present_value()`, `fit_variance_cumulation()`, `form_present_value_grid()`, `implied_jump_process_volatility()`

`fit_variance_cumulation` *Fit piecewise constant volatilities to a set of equity options*

Description

Given a set of equity options with increasing tenors, along with target prices for those options, and a set of equity-lined default SDE parameters, fit a vector of piecewise constant volatilities and an associated cumulative variance function to them.

Usage

```
fit_variance_cumulation(
  S0,
  eq_options,
  mid_prices,
  spreads = NULL,
  initial_vols_guess = 0.55 + 0 * mid_prices,
  use_impvol = TRUE,
  relative_spread_tolerance = 0.01,
  force_same_grid = FALSE,
  num_time_steps = 40,
  const_short_rate = 0,
  const_default_intensity = 0,
  discount_factor_fcn = function(T, t, ...) {
     exp(-const_short_rate * (T - t))
 },
  survival_probability_fcn = function(T, t, ...) {
     exp(-const_default_intensity * (T
    - t))
 },
  default_intensity_fcn = function(t, S, ...) {
     const_default_intensity + 0 * S
 },
  dividends = NULL,
  borrow_cost = 0,
  dividend_rate = 0,
  ...
)
```

Arguments

`S0`	Current stock price
`eq_options`	A list of options to find prices for. Each must have fields `callput`, `maturity`, and `strike`. This list must be in strictly increasing order of maturity.
`mid_prices`	Prices to match

`spreads` Spreads within which any match is tolerable

`initial_vols_guess`
Initial set of volatilities to try in the root finder

`use_impvol` Judge fit quality on implied vol distance rather than price distance

`relative_spread_tolerance`
Tolerance multiplier on bid-ask spreads taken from vol normalization

`force_same_grid`
Price all options on the same grid, rather than having smaller timestep sizes for earlier maturities

`num_time_steps` Minimum number of time steps in the grid

`const_short_rate`
A constant to use for the instantaneous interest rate in case `discount_factor_fcn` is not given

`const_default_intensity`
A constant to use for the instantaneous default intensity in case `default_intensity_fcn` is not given

`discount_factor_fcn`
A function for computing present values to time `t` of various cashflows occurring, with arguments `T`, `t`

`survival_probability_fcn`
A function for probability of survival, with arguments `T`, `t` and `T>t`. E.g. with a constant volatility s this takes the form $(T-t)s^2$. This argument is only used in normalization of prices to vols for root finder tolerance, and is therefore entirely optional

`default_intensity_fcn`
A function for computing default intensity occurring during this timestep, dependent on time and stock price, with arguments `t`, `S`. Should be consistent with `survival_probability_fcn` if specified

`dividends` A `data.frame` with columns `time`, `fixed`, and `proportional`. Dividend size at the given `time` is

`borrow_cost` Stock borrow cost, affecting the drift rate

`dividend_rate` Continuous dividend rate, affecting the drift rate

`...` Further arguments to `find_present_value`

Details

By default, the fitting happens in implied Black-Scholes volatility space for better normalization. That is to say, the fitting does pricing using the *full* SDE and PDE solver via `find_present_value`, but judges fit quality on the basis of running resulting prices through a nonlinear transformation that just happens to come from the straight Black-Scholes model.

Value

A list with three elements, `volatilities`, `times` and `cumulation_function`. The `cumulation_function` will be a 2-parameter function giving cumulated variances, as created by `variance_cumulation_from_vols`

See Also

Other Implied Volatilities: american_implied_volatility(), equivalent_bs_vola_to_jump(), equivalent_jump_vola_to_bs(), implied_jump_process_volatility(), implied_volatilities(), implied_volatilities_with_rates_struct(), implied_volatility(), implied_volatility_with_term_struct()

Other Equity Dependent Default Intensity: find_present_value(), fit_to_option_market_df(), form_present_value_grid(), implied_jump_process_volatility()

`form_present_value_grid` *Use a model to estimate the present value of financial derivatives on a grid of initial underlying values*

Description

Use a finite difference scheme to form estimates of present values for a variety of stock prices on a grid of initial underlying prices, determined by constructing a logarithmic equivalent conforming to the grid parameters `structure_constant` and `structure_constant`

Usage

```
form_present_value_grid(
  S0,
  num_time_steps,
  instruments,
  const_volatility = 0.5,
  const_short_rate = 0,
  const_default_intensity = 0,
  override_Tmax = NA,
  discount_factor_fcn = function(T, t, ...) {
     exp(-const_short_rate * (T - t))
 },
  default_intensity_fcn = function(t, S, ...) {
     const_default_intensity + 0 * S
 },
  variance_cumulation_fcn = function(T, t) {
     const_volatility^2 * (T - t)
 },
  dividends = NULL,
  borrow_cost = 0,
  dividend_rate = 0,
  structure_constant = 2,
  std_devs_width = 3,
  grid_center = NA
)
```

Arguments

`S0`	An initial stock price, for setting grid scale
`num_time_steps`	Minimum number of time steps in the grid
`instruments`	A list of instruments to be priced. Each one must have a `strike` and a `optionality_fcn`, as with `GridPricedInstrument` and its subclasses.
`const_volatility`	A constant to use for volatility in case `variance_cumulation_fcn` is not given
`const_short_rate`	A constant to use for the instantaneous interest rate in case `discount_factor_fcn` is not given
`const_default_intensity`	A constant to use for the instantaneous default intensity in case `default_intensity_fcn` is not given
`override_Tmax`	A different maximum time on the grid to enforce
`discount_factor_fcn`	A function for computing present values to time `t` of various cashflows occurring during this timestep, with arguments `T`, `t`
`default_intensity_fcn`	A function for computing default intensity occurring during this timestep, dependent on time and stock price, with arguments `t`, `S`.
`variance_cumulation_fcn`	A function for computing total stock variance occurring during this timestep, with arguments `T`, `t`. E.g. with a constant volatility s this takes the form $(T-t)s^2$.
`dividends`	A `data.frame` with columns `time`, `fixed`, and `proportional`. Dividend size at the given `time` is then expected to be equal to `fixed + proportional * S / S0`
`borrow_cost`	Stock borrow cost, affecting the drift rate
`dividend_rate`	Continuous dividend rate, affecting the drift rate
`structure_constant`	The maximum ratio between time intervals `dt` and the square of space intervals `dz^2`
`std_devs_width`	The number of standard deviations, in `sigma * sqrt(T)` units, to incorporate into the grid
`grid_center`	A reasonable central value for the grid, defaults to S0 or an instrument strike

Details

If any instrument in the `instruments` has a strike, then the grid will be normalized to the last such instrument's strike.

See Also

Other Equity Dependent Default Intensity: `find_present_value()`, `fit_to_option_market_df()`, `fit_variance_cumulation()`, `implied_jump_process_volatility()`

Other Implicit Grid Solver: `construct_implicit_grid_structure()`, `find_present_value()`, `infer_conforming_time_grid()`, `integrate_pde()`, `iterate_grid_from_timestep()`, `pde_matrix_solve()`, `take_implicit_timestep()`, `timestep_instruments()`

GREEK_NAMES *Greeks available for computation, presently "delta", "gamma", "theta", "vega", "rates_dv01", and "credit_dv01". Note in particular that we use the term* `rates_dv01` *rather than* `rho`

Description

Greeks available for computation, presently "delta", "gamma", "theta", "vega", "rates_dv01", and "credit_dv01". Note in particular that we use the term `rates_dv01` rather than `rho`

Usage

```
GREEK_NAMES
```

See Also

Other Greeks: `construct_descending_bumps()`, `find_greeks()`, `greek_by_fd()`, `grid_delta_gamma()`, `resolve_bumps()`, `robust_greek()`, `safe_reprice()`

GridPricedInstrument-class *Representation of financial instrument amenable to grid pricing schemes*

Description

Our basic instrument defines a tenor/maturity, a method to provide values in case of default, and a method to correct instrument prices in light of exercise decisions.

Fields

`maturity` The tenor, expiration date or terminal date by which the value of this security will be certain.

`last_computed_grid` The most recently computed set of values from a grid pricing scheme. Used internally for pricing chains of derivatives.

`name` A mnemonic name for the instrument, not used by ragtop

Methods

`optionality_fcn(v, ...)` Return a version of `v` at time `t` corrected for any optionality conditions.

`recovery_fcn(v, S, t, ...)` Return recovery value, given non-default values `v` at time `t`. Subclasses may be more elaborate, this method simply returns 0.0.

`reset_caches()` Clear any memoization caches so this instrument can be repriced afresh. Subclasses extend this to clear their own caches.

`terminal_values(v, ...)` Return a terminal value. defaults to simply calling `optionality_fcn`.

`implied_jump_process_volatility`
Implied volatility of any instrument

Description

Use the grid solver to generate instrument prices via `find_present_value` and run them through a bisective root search method until a constant volatility matching the provided instrument price has been found.

Usage

```
implied_jump_process_volatility(
  instrument_price,
  instrument,
  ...,
  starting_volatility_estimate = 0.85,
  relative_tolerance = 0.005,
  max.iter = 100,
  max_vola = 4
)
```

Arguments

`instrument_price`	Target price for root finder
`instrument`	Instrument to search for the target price on, passed as the sole instrument to `find_present_value`
`...`	Additional arguments to be passed on to `find_present_value`
`starting_volatility_estimate`	Bisection method original guess
`relative_tolerance`	Relative tolerance in instrument price defining the root-finder halting condition
`max.iter`	Maximum number of root-finder iterations allowed
`max_vola`	Maximum volatility to try

Details

Unlike `american_implied_volatility`, this routine allows for any legal term structures and equity-linked default intensities. For that reason, it eschews the control variate tricks that make `american_implied_volatility` so much faster.

Note that equity-linked default intensities can result in instrument prices that are not monotonic in volatility. This bisective root finder will find a solution but not necessarily any particular one.

Value

A list of present values, with the same names as `instruments`

See Also

`find_present_value` for the underlying pricing algorithm, `implied_volatility_with_term_struct` for European options without equity dependence of default intensity, `american_implied_volatility` for the same on American options

Other Implied Volatilities: `american_implied_volatility()`, `equivalent_bs_vola_to_jump()`, `equivalent_jump_vola_to_bs()`, `fit_variance_cumulation()`, `implied_volatilities()`, `implied_volatilities_w`
`implied_volatility()`, `implied_volatility_with_term_struct()`

Other Equity Dependent Default Intensity: `find_present_value()`, `fit_to_option_market_df()`, `fit_variance_cumulation()`, `form_present_value_grid()`

Examples

```
implied_jump_process_volatility(
    25, AmericanOption(maturity=1.1, strike=100, callput=-1),
    S0=100, num_time_steps=50, relative_tolerance=1.e-3)
```

---

implied_volatilities *Implied volatilities of european-exercise options under Black-Scholes or a jump-process extension*

Description

Find default-free volatilities based on known interest rates and hazard rates, using a given option price.

Usage

```
implied_volatilities(
  option_price,
  callput,
  S0,
  K,
  r,
```

```
  time,
  const_default_intensity = 0,
  divrate = 0,
  borrow_cost = 0,
  dividends = NULL,
  relative_tolerance = 1e-06,
  max.iter = 100,
  max_vola = 4
)
```

Arguments

option_price	Present option values (may be a vector)
callput	1 for calls, -1 for puts (may be a vector)
S0	initial underlying price (may be a vector)
K	strike (may be a vector)
r	risk-free interest rate (may be a vector)
time	Time from 0 until expiration (may be a vector)
const_default_intensity	hazard rate of underlying default (may be a vector)
divrate	A continuous rate for dividends and other cashflows such as foreign interest rates (may be a vector)
borrow_cost	A continuous rate for stock borrow costs (may be a vector)
dividends	A data.frame with columns time, fixed, and proportional. Dividend size at the given time is then expected to be equal to fixed + proportional * S / S0. Fixed dividends will be converted to proprtional for purposes of this algorithm.
relative_tolerance	Relative tolerance in option price to achieve before halting the search
max.iter	Number of iterations to try before abandoning the search
max_vola	Maximum volatility to try in the search

Value

Scalar volatilities

See Also

Other Implied Volatilities: american_implied_volatility(), equivalent_bs_vola_to_jump(), equivalent_jump_vola_to_bs(), fit_variance_cumulation(), implied_jump_process_volatility(), implied_volatilities_with_rates_struct(), implied_volatility(), implied_volatility_with_term_struct()

Other European Options: black_scholes_on_term_structures(), blackscholes(), implied_volatilities_with_rat implied_volatility(), implied_volatility_with_term_struct()

Other Equity Independent Default Intensity: american(), american_implied_volatility(), black_scholes_on_term_structures(), blackscholes(), equivalent_bs_vola_to_jump(), equivalent_jump_vola_to_bs(), implied_volatilities_with_rates_struct(), implied_volatility(), implied_volatility_with_term_struct()

`implied_volatilities_with_rates_struct` *Find the implied volatility of european-exercise options with a term structure of interest rates*

Description

Use the provided discount factor function to infer constant short rates applicable to each expiration time, then use the Black-Scholes formula to generate European option values and run them through Newton's method until a constant volatility matching each provided option price has been found.

Usage

```
implied_volatilities_with_rates_struct(
  option_price,
  callput,
  S0,
  K,
  discount_factor_fcn,
  time,
  const_default_intensity = 0,
  divrate = 0,
  borrow_cost = 0,
  dividends = NULL,
  relative_tolerance = 1e-06,
  max.iter = 100,
  max_vola = 4
)
```

Arguments

`option_price`	Present option values (may be a vector)
`callput`	`1` for calls, `-1` for puts (may be a vector)
`S0`	initial underlying prices (may be a vector)
`K`	strikes (may be a vector)
`discount_factor_fcn`	A function for computing present values to time `t`, with arguments `T`, `t`
`time`	Time from `0` until expirations (may be a vector)
`const_default_intensity`	hazard rates of underlying default (may be a vector)
`divrate`	A continuous rate for dividends and other cashflows such as foreign interest rates (may be a vector)
`borrow_cost`	A continuous rate for stock borrow costs (may be a vector)

dividends	A data.frame with columns time, fixed, and proportional. Dividend size at the given time is then expected to be equal to fixed + proportional * S / S0. Fixed dividends will be converted to proprtional for purposes of this algorithm.
relative_tolerance	Relative tolerance in option price to achieve before halting the search
max.iter	Number of iterations to try before abandoning the search
max_vola	Maximum volatility to try in the search

Details

Differs from implied_volatility_with_term_struct by first computing constant interest rates for each option, and then calling implied_volatilities

Value

Scalar volatilities

See Also

implied_volatility for simpler cases with constant parameters, implied_volatilities for the underlying algorithm with constant rates, implied_volatility_with_term_struct when volatilities or survival probabilities also have a nontrivial term structure

Other Implied Volatilities: american_implied_volatility(), equivalent_bs_vola_to_jump(), equivalent_jump_vola_to_bs(), fit_variance_cumulation(), implied_jump_process_volatility(), implied_volatilities(), implied_volatility(), implied_volatility_with_term_struct()

Other European Options: black_scholes_on_term_structures(), blackscholes(), implied_volatilities(), implied_volatility(), implied_volatility_with_term_struct()

Other Equity Independent Default Intensity: american(), american_implied_volatility(), black_scholes_on_term_structures(), blackscholes(), equivalent_bs_vola_to_jump(), equivalent_jump_vola_to_bs(), implied_volatilities(), implied_volatility(), implied_volatility_with_ter

Examples

```
d_fcn = function(T,t) {exp(-0.03*(T-t))}
implied_volatilities_with_rates_struct(c(23,24,25),
       c(-1,1,1), 100, 100,
       discount_factor_fcn=d_fcn, time=c(4,4,5))
```

---

implied_volatility *Implied volatility of european-exercise option under Black-Scholes or a jump-process extension*

Description

Find default-free volatility (not necessarily just Black-Scholes) based on known interest rates and hazard rates, using a given option price.

Usage

```
implied_volatility(
  option_price,
  callput,
  S0,
  K,
  r,
  time,
  const_default_intensity = 0,
  divrate = 0,
  borrow_cost = 0,
  dividends = NULL,
  relative_tolerance = 1e-06,
  max.iter = 100,
  max_vola = 4
)
```

Arguments

`option_price`	Present option value
`callput`	`1` for calls, `-1` for puts
`S0`	initial underlying price
`K`	strike
`r`	risk-free interest rate
`time`	Time from `0` until expiration
`const_default_intensity`	hazard rate of underlying default
`divrate`	A continuous rate for dividends and other cashflows such as foreign interest rates
`borrow_cost`	A continuous rate for stock borrow costs
`dividends`	A `data.frame` with columns `time`, `fixed`, and `proportional`. Dividend size at the given `time` is then expected to be equal to `fixed + proportional * S / S0`. Fixed dividends will be converted to proportional for purposes of this algorithm. To handle truly fixed dividends, see `implied_jump_process_volatility`
`relative_tolerance`	Relative tolerance in option price to achieve before halting the search
`max.iter`	Number of iterations to try before abandoning the search
`max_vola`	Maximum volatility to try in the search

Details

To get a straight Black-Scholes implied volatility, simply call this function with `const_default_intensity` set to zero (the default).

Value

A scalar volatility

See Also

Other Implied Volatilities: american_implied_volatility(), equivalent_bs_vola_to_jump(), equivalent_jump_vola_to_bs(), fit_variance_cumulation(), implied_jump_process_volatility(), implied_volatilities(), implied_volatilities_with_rates_struct(), implied_volatility_with_term_struct

Other Equity Independent Default Intensity: american(), american_implied_volatility(), black_scholes_on_term_structures(), blackscholes(), equivalent_bs_vola_to_jump(), equivalent_jump_vola_to_bs(), implied_volatilities(), implied_volatilities_with_rates_struct(), implied_volatility_with_term_struct()

Other European Options: black_scholes_on_term_structures(), blackscholes(), implied_volatilities(), implied_volatilities_with_rates_struct(), implied_volatility_with_term_struct()

Examples

```
implied_volatility(2.5, 1, 100, 105, 0.01, 0.75)
implied_volatility(option_price = 17,
                   callput = CALL, S0 = 250,  K=245,
                   r = 0.005, time = 2,
                   const_default_intensity = 0.03)
```

`implied_volatility_with_term_struct` *Find the implied volatility of a european-exercise option with term structures*

Description

Use the Black-Scholes formula to generate European option values and run them through Newton's method until a constant volatility matching the provided option price has been found.

Usage

```
implied_volatility_with_term_struct(
  option_price,
  callput,
  S0,
  K,
  time,
  ...,
  starting_volatility_estimate = 0.5,
  relative_tolerance = 1e-06,
  max.iter = 100,
  max_vola = 4
)
```

Arguments

option_price	Option price to match
callput	1 for calls, -1 for puts
S0	initial underlying price
K	strike
time	Time to expiration
...	Further arguments to be passed on to black_scholes_on_term_structures
starting_volatility_estimate	The Newton method's original guess
relative_tolerance	Relative tolerance in instrument price defining the root-finder halting condition
max.iter	Maximum number of root-finder iterations allowed
max_vola	Maximum volatility to try

Details

Differs from implied_volatility by calling black_scholes_on_term_structures for pricing, thereby allowing term structures of rates, and a nontrivial survival_probability_fcn

Value

Estimated volatility

See Also

implied_volatility for simpler cases with constant parameters, black_scholes_on_term_structures for the underlying pricing algorithm, implied_volatilities_with_rates_struct when neither volatilities nor survival probabilities have a nontrivial term structure

Other Implied Volatilities: american_implied_volatility(), equivalent_bs_vola_to_jump(), equivalent_jump_vola_to_bs(), fit_variance_cumulation(), implied_jump_process_volatility(), implied_volatilities(), implied_volatilities_with_rates_struct(), implied_volatility()

Other Equity Independent Default Intensity: american(), american_implied_volatility(), black_scholes_on_term_structures(), blackscholes(), equivalent_bs_vola_to_jump(), equivalent_jump_vola_to_bs(), implied_volatilities(), implied_volatilities_with_rates_struct(), implied_volatility()

Other European Options: black_scholes_on_term_structures(), blackscholes(), implied_volatilities(), implied_volatilities_with_rates_struct(), implied_volatility()

Examples

```
## Dividends
divs = data.frame(time=seq(from=0.11, to=2, by=0.25),
                  fixed=seq(1.5, 1, length.out=8),
                  proportional = seq(1, 1.5, length.out=8))
surv_prob_fcn = function(T, t, ...) {
  exp(-0.07 * (T - t)) }
```

```
disc_factor_fcn = function(T, t, ...) {
  exp(-0.03 * (T - t)) }
implied_volatility_with_term_struct(
    option_price = 12, S0 = 150, callput=PUT,
    K = 147.50, time=1.5,
    discount_factor_fcn=disc_factor_fcn,
    survival_probability_fcn=surv_prob_fcn,
    dividends=divs)
```

`infer_conforming_time_grid` *A time grid with extra times inserted for coupons, calls and puts*

Description

At its base, this function chooses a time grid with `1+min_num_time_steps` elements from 0 to `Tmax`. Any coupon, call, or put times occurring in one of the supplied instruments are also inserted.

Usage

```
infer_conforming_time_grid(min_num_time_steps, Tmax, instruments = NULL)
```

Arguments

`min_num_time_steps`	The minimum number of timesteps the output vector should have
`Tmax`	The maximum time on the grid
`instruments`	A set of instruments whose maturity and terms and conditions can introduce extra timesteps. Each will be queried for the output of a `critical_times` function.

Value

A vector of times at which the grid should have nodes

See Also

Other Implicit Grid Solver: `construct_implicit_grid_structure()`, `find_present_value()`, `form_present_value_grid()`, `integrate_pde()`, `iterate_grid_from_timestep()`, `pde_matrix_solve()`, `take_implicit_timestep()`, `timestep_instruments()`

integrate_pde *Numerically integrate the pricing differential equation*

Description

Use an implicit integration scheme to numerically integrate the pricing differential equation for each of the given instruments, backwardating from time `Tmax` to time 0.

Usage

```
integrate_pde(
  z,
  min_num_time_steps,
  S0,
  Tmax,
  instruments,
  stock_level_fcn,
  discount_factor_fcn,
  default_intensity_fcn,
  variance_cumulation_fcn,
  dividends = NULL
)
```

Arguments

`z`	Space grid value morphable to stock prices using `stock_level_fcn`
`min_num_time_steps`	The minimum number of timesteps used. Calls, puts and coupons may result in extra timesteps taken.
`S0`	Time zero price of the base equity
`Tmax`	The maximum time on the grid, from which all backwardation steps will take place.
`instruments`	A list of instruments to be priced. Each one must have a `strike` and a `optionality_fcn`, as with `GridPricedInstrument` and its subclasses.
`stock_level_fcn`	A function for changing space grid value to stock prices, with arguments `z` and `t`
`discount_factor_fcn`	A function for computing present values to time `t` of various cashflows occurring during this timestep, with arguments `T`, `t`
`default_intensity_fcn`	A function for computing default intensity occurring during this timestep, dependent on time and stock price, with arguments `t`, `S`.

variance_cumulation_fcn
: A function for computing total stock variance occurring during this timestep, with arguments T, t. E.g. with a constant volatility s this takes the form $(T - t)s^2$.

dividends
: A data.frame with columns time, fixed, and proportional. Dividend size at the given time is then expected to be equal to fixed + proportional * S / S0

Value

A grid of present values of derivative prices, adapted to z at each timestep. Time zero value will appear in the first index.

See Also

Other Implicit Grid Solver: construct_implicit_grid_structure(), find_present_value(), form_present_value_grid(), infer_conforming_time_grid(), iterate_grid_from_timestep(), pde_matrix_solve(), take_implicit_timestep(), timestep_instruments()

---

is.blank *Return* TRUE *if the argument is empty,* NULL *or* NA

Description

Return TRUE if the argument is empty, NULL or NA

Usage

```
is.blank(x, false.triggers = FALSE)
```

Arguments

x
: Argument to test

false.triggers
: Whether to allow nonempty vectors of all FALSE to trigger this condition

---

iterate_grid_from_timestep *Iterate over a set of timesteps to integrate the pricing differential equation*

Description

Timestep an implicit integration scheme to numerically integrate the pricing differential equation for each of the given instruments, backwardating from time Tmax to time 0.

Usage

```
iterate_grid_from_timestep(
  starting_time_step,
  time_pts,
  z,
  S0,
  instruments,
  stock_level_fcn,
  discount_factor_fcn,
  default_intensity_fcn,
  variance_cumulation_fcn,
  dividends = NULL,
  grid = NULL,
  original_grid_values = as.matrix(grid[1 + starting_time_step, , ])
)
```

Arguments

`starting_time_step`	The index into time_pts of the first timestep to be employed. This must be no larger than the length of time_pts, minus one
`time_pts`	Time nodes to be treated on the grid
`z`	Space grid value morphable to stock prices using `stock_level_fcn`
`S0`	Time zero price of the base equity
`instruments`	A list of instruments to be priced. Each one must have a `strike` and a `optionality_fcn`, as with `GridPricedInstrument` and its subclasses.
`stock_level_fcn`	A function for changing space grid value to stock prices, with arguments `z` and `t`
`discount_factor_fcn`	A function for computing present values to time `t` of various cashflows occurring during this timestep, with arguments `T`, `t`
`default_intensity_fcn`	A function for computing default intensity occurring during this timestep, dependent on time and stock price, with arguments `t`, `S`.
`variance_cumulation_fcn`	A function for computing total stock variance occurring during this timestep, with arguments `T`, `t`. E.g. with a constant volatility s this takes the form $(T-t)s^2$.
`dividends`	A `data.frame` with columns `time`, `fixed`, and `proportional`. Dividend size at the given `time` is then expected to be equal to `fixed + proportional * S / S0`
`grid`	An optional grid into which results at each timestep will be written. Its size should be at least `(1+starting_time_step, length(z), length(instruments))`
`original_grid_values`	Grid values to timestep from

Value

Either a populated grid of present values of derivative prices, or a matrix of values at the first time point, adapted to z at each timestep. Time zero value will appear in the first index of any grid.

See Also

Other Implicit Grid Solver: `construct_implicit_grid_structure()`, `find_present_value()`, `form_present_value_grid()`, `infer_conforming_time_grid()`, `integrate_pde()`, `pde_matrix_solve()`, `take_implicit_timestep()`, `timestep_instruments()`

`pde_matrix_solve` *Solve the implicit-timestep linear system*

Description

Solve the linear system $Ax = rhs$ arising from one implicit finite-difference timestep, where A is the matrix assembled by `construct_tridiagonals` and b is the vector of price space-grid values from the previously calculated timestep (which corresponds to one step further in our future).

Usage

```
pde_matrix_solve(tridiag_matrix_entries, rhs, method = "auto")
```

Arguments

`tridiag_matrix_entries`	Diagonal, superdiagonal and subdiagonal of the matrix from the numerical integrator, as a list with elements `sub`, `diag` and `super` (see `construct_tridiagonals`)
`rhs`	A vector forming the right-hand side of the linear system, typically the space grid values from the previously calculated timestep
`method`	Character string selecting the linear-system solver. One of `'lapack'` (use `limSolve::Solve.tridiag()`, LAPACK DGTSV), `'banded'` (assemble a sparse banded matrix and solve with the **Matrix** package) or `'thomas'` (a pure-R Thomas tridiagonal solve, always available). Any other value (including the default `'auto'`) selects the first available method in that same logical order.

Details

By default (`method='auto'`) the solver prefers `limSolve::Solve.tridiag()`, which calls LAPACK DGTSV Gaussian elimination with partial pivoting, then falls back to a sparse banded solve from the **Matrix** package, and finally to a pure-R Thomas algorithm that requires no additional packages. Passing an explicit `method` forces that solver; forcing `'lapack'` or `'banded'` requires the corresponding package to be installed.

Value

The solution vector x of the linear system

See Also

Other Implicit Grid Solver: `construct_implicit_grid_structure()`, `find_present_value()`, `form_present_value_grid()`, `infer_conforming_time_grid()`, `integrate_pde()`, `iterate_grid_from_timestep()`, `take_implicit_timestep()`, `timestep_instruments()`

`penalty_with_intensity_link` *Helper function (volatility-normalized pricing error) for calibration of equity-linked default intensity*

Description

Given a set SDE parameters, form a volatility term structure that fairly precisely matches the supplied prices of the `variance_instruments`. Then use that term structure and the default intensity to price all the `fit_instruments`, and compare them to the `fit_instrument_prices`.

Usage

```
penalty_with_intensity_link(
  p,
  s,
  h,
  variance_instruments,
  variance_instrument_prices,
  variance_instrument_spreads,
  fit_instruments,
  fit_instrument_prices,
  fit_instrument_spreads,
  fit_instrument_weights,
  S0,
  num_time_steps = 30,
  const_short_rate = 0,
  discount_factor_fcn = function(T, t) {
     exp(-const_short_rate * (T - t))
 },
  ...,
  relative_spread_tolerance = 0.15,
  num_variance_time_steps = 30
)
```

Arguments

`p`	Power of default intensity
`s`	Proportion of constant default intensity
`h`	Base default intensity

variance_instruments
: A list of instruments in strictly increasing order of maturity, from which the volatility term structure will be inferred. Once the calibration is finished, the chosen parameters will reproduce the prices of these instruments with fairly high precision.

variance_instrument_prices
: Central price targets for the variance instruments

variance_instrument_spreads
: Bid-offer spreads used to normalize errors in variance instrument prices during term structure fitting

fit_instruments
: A list of instruments in any order, from which the mispricing penalties used for judging fit quality will be computed

fit_instrument_prices
: Central price targets for the variance instruments

fit_instrument_spreads
: Bid-offer spreads used to normalize errors in fit instrument prices during default intensity

fit_instrument_weights
: Weights applied to relative errors in fit instrument prices before summing to form the penalty

S0
: Current underlying price

num_time_steps
: Time step count passed on to find_present_value while fitting instrument values

const_short_rate
: A constant to use for the instantaneous interest rate in case discount_factor_fcn is not given

discount_factor_fcn
: A function for computing present values to time t of various cashflows occurring during this timestep, with arguments T, t

...
: Further arguments passed to price_with_intensity_link

relative_spread_tolerance
: Tolerance to apply in calling fit_variance_cumulation

num_variance_time_steps
: Number of time steps to use in calling fit_variance_cumulation

Details

Forms implied Black-Scholes volatilities from all supplied mid prices, and their implied bid and offer prices, as well as from the prices computed by the grid solver. Each instrument is then assigned an error term component in proportion to its weight and the pricing error (in implied vol terms) divided by the spread (also in implied vol terms).

See Also

price_with_intensity_link for the pricing function

price_with_intensity_link *Helper function (instrument pricing) for calibration of equity-linked default intensity*

Description

Given derivative instruments (subclasses of GridPricedInstrument, though typically either AmericanOption or EuropeanOption objects), along with their prices and spreads, calibrate variance cumulation (the at-the-money volatility of the continuous process) and then price the instruments via equity linked default intensity of the form $h(s + (1-s)(S0/S_t)^p)$.

Usage

```
price_with_intensity_link(
  p,
  s,
  h,
  variance_instruments,
  variance_instrument_prices,
  variance_instrument_spreads,
  fit_instruments,
  S0,
  num_time_steps = 30,
  ...,
  relative_spread_tolerance = 0.15,
  num_variance_time_steps = 30
)
```

Arguments

p	Power of default intensity
s	Proportion of constant default intensity
h	Base default intensity
variance_instruments	A list of instruments in strictly increasing order of maturity, from which the volatility term structure will be inferred. Once the calibration is finished, the chosen parameters will reproduce the prices of these instruments with fairly high precision.
variance_instrument_prices	Central price targets for the variance instruments
variance_instrument_spreads	Bid-offer spreads used to normalize errors in variance instrument prices during term structure fitting

fit_instruments
: A list of instruments in any order, from which the mispricing penalties used for judging fit quality will be computed

S0
: Current underlying price

num_time_steps
: Time step count passed on to find_present_value while fitting instrument values

...
: Further arguments passed to both fit_variance_cumulation and to find_present_value

relative_spread_tolerance
: Tolerance to apply in calling fit_variance_cumulation

num_variance_time_steps
: Number of time steps to use in calling fit_variance_cumulation

---

PUT *Constant PUT for defining option contracts*

Description

Constant PUT for defining option contracts

Usage

```
PUT
```

---

shift_for_dividends *Shift a set of grid values for dividends paid, using spline interpolation*

Description

Shift a set of grid values for dividends paid, using spline interpolation

Usage

```
shift_for_dividends(grid_values_before_shift, stock_prices, div_sum)
```

Arguments

grid_values_before_shift
: Values on grid before accounting for expected dividends

stock_prices
: Stock prices for which to shift the grid

div_sum
: Sum of dividend values at each grid point

Value

An object like grid_values_before_shift with entries shifted according to the dividend sums

See Also

Other Dividends: adjust_for_dividends(), time_adj_dividends()

`spot_to_df_fcn` *Create a discount factor function from a yield curve*

Description

Use a piecewise constant approximation to the given spot curve to generate a function capable of returning corresponding discount factors

Usage

```
spot_to_df_fcn(yield_curve)
```

Arguments

`yield_curve`	A data.frame with numeric columns `time` (in increasing order) and `rate` (in natural units)

Value

A function taking two time arguments, which returns the discount factor from the second to the first

Examples

```
disct_fcn = ragtop::spot_to_df_fcn(
  data.frame(time=c(1, 5, 10, 15),
             rate=c(0.01, 0.02, 0.03, 0.05)))
print(disct_fcn(1, 0.5))
```

`take_implicit_timestep` *Backwardate grid values one timestep*

Description

Take one timestep of an implicit solver for a given instrument

Usage

```
take_implicit_timestep(
  t,
  S,
  full_discount_factor,
  local_discount_factor,
  discount_factor_fcn,
  prev_grid_values,
  survival_probabilities,
```

```
  tridiag_matrix_entries,
  instrument = NULL,
  dividends = NULL,
  instr_name = "this instrument"
)
```

Arguments

`t`	Time after this timestep has been taken
`S`	Underlying equity values for the grid
`full_discount_factor`	A discount factor for the transform from grid values to actual derivative prices
`local_discount_factor`	A discount factor to apply to recovery values
`discount_factor_fcn`	A function for computing present values to time `t` of various cashflows occurring during this timestep, with arguments `T`, `t`
`prev_grid_values`	A vector of space grid values from the previously calculated timestep
`survival_probabilities`	Vector of probabilities of survival for each space grid node
`tridiag_matrix_entries`	Diagonal, superdiagonal and subdiagonal of tridiagonal matrix from the numerical integrator
`instrument`	If not `NULL`/`NA`, must have a `recovery_fcn` and an `optionality_fcn` though those properties are themselves allowed to be `NA`.
`dividends`	A `data.frame` with columns `time`, `fixed`, and `proportional`. Dividend size at the given `time` is then expected to be equal to `fixed + proportional * S / S0`
`instr_name`	Name of instrument to use in log messages

Value

Grid values for the instrument after taking the implicit timestep

See Also

Other Implicit Grid Solver: `construct_implicit_grid_structure()`, `find_present_value()`, `form_present_value_grid()`, `infer_conforming_time_grid()`, `integrate_pde()`, `iterate_grid_from_timestep()`, `pde_matrix_solve()`, `timestep_instruments()`

`timestep_instruments` *Take an implicit timestep for all the given instruments*

Description

Backwardate grid values for all the given instruments from a set of grid values matched to time `t+dt` to form a new set of grid value as of time `t`.

Usage

```
timestep_instruments(
  z,
  prev_grid_values,
  t,
  dt,
  S0,
  instruments,
  stock_level_fcn,
  discount_factor_fcn,
  default_intensity_fcn,
  variance_cumulation_fcn,
  dividends = NULL
)
```

Arguments

`z`	Space grid value morphable to stock prices using `stock_level_fcn`
`prev_grid_values`	A matrix with one column for each instrument and one row for each of the N values of `z`
`t`	Time after this timestep has been taken
`dt`	Interval to the end of this timestep
`S0`	Time zero price of the base equity
`instruments`	Instruments corresponding to layers of the value grid in `prev_grid_values`
`stock_level_fcn`	A function for changing space grid value to stock prices, with arguments `z` and `t`
`discount_factor_fcn`	A function for computing present values to time `t` of various cashflows occurring during this timestep, with arguments `T`, `t`
`default_intensity_fcn`	A function for computing default intensity occurring during this timestep, dependent on time and stock price, with arguments `t`, `S`.

variance_cumulation_fcn
: A function for computing total stock variance occurring during this timestep, with arguments T, t. E.g. with a constant volatility s this takes the form $(T - t)s^2$.

dividends
: A data.frame with columns time, fixed, and proportional. Dividend size at the given time is then expected to be equal to fixed + proportional * S / S0

Value

Grid values after applying an implicit timestep

See Also

Other Implicit Grid Solver: construct_implicit_grid_structure(), find_present_value(), form_present_value_grid(), infer_conforming_time_grid(), integrate_pde(), iterate_grid_from_timestep(), pde_matrix_solve(), take_implicit_timestep()

---

time_adj_dividends *Find the sum of time-adjusted dividend values*

Description

For each of the N elements of S/h find the sum of the given M dividends, discounted to t_final by r and h

Usage

```
time_adj_dividends(relevant_divs, t_final, r, h, S, S0)
```

Arguments

relevant_divs
: A data.frame with columns time, fixed, and proportional. Dividend size at the given time is then expected to be equal to fixed + proportional * S / S0

t_final
: Time beyond which to ignore dividends

r
: risk-free interest rate

h
: Default intensities

S
: Stock prices

S0
: initial underlying price

Value

Sum of dividends, at each grid node

See Also

Other Dividends: adjust_for_dividends(), shift_for_dividends()

`TIME_RESOLUTION_FACTOR` *Constant to define when times are considered so close to each other that they should be treated as simultaneous*

Description

Constant to define when times are considered so close to each other that they should be treated as simultaneous

Usage

```
TIME_RESOLUTION_FACTOR
```

`TIME_RESOLUTION_SIGNIF_DIGITS` *Constant to define when times are considered so close to each other that they should be treated as simultaneous, in terms of significant digits*

Description

Constant to define when times are considered so close to each other that they should be treated as simultaneous, in terms of significant digits

Usage

```
TIME_RESOLUTION_SIGNIF_DIGITS
```

`treasury_df` *Get a US Treasury curve discount factor function*

Description

This is a caching wrapper for `treasury_df_raw`

Usage

```
treasury_df(..., envir = parent.frame())
```

Arguments

`...`	Arguments passed to `treasury_df_raw`
`envir`	Environment passed to `treasury_df_raw`

Value

A function taking two time arguments, which returns the discount factor from the second to the first (see `spot_to_df_fcn`)

`treasury_df_raw` *Get a US Treasury curve discount factor function*

Description

Get a US Treasury curve discount factor function

Usage

```
treasury_df_raw(on_date)
```

Arguments

`on_date`	Date for which to query for the curve, year-month-day format

Value

A function taking two time arguments, which returns the discount factor from the second to the first

---

`TSLAMarket` *Market information snapshot for TSLA options*

Description

A dataset containing option contract details and a snapshot of market prices for Tesla Motors (TSLA) equity options, interest rates and an equity price.

Usage

```
TSLAMarket
```

Format

A list with these prices and rates:

`S0` The stock price as of snapshot time

`risk_free_rates` The spot risk-free rate curve as of snapshot time

`options` A data frame with details of the options market

Examples

```
TSLAMarket
```

value_from_prior_coupons

Present value of past coupons paid

Description

Present value as of time `t` for coupons paid since the `model_t`

Usage

```
value_from_prior_coupons(t, coupons_df, discount_factor_fcn, model_t = 0)
```

Arguments

`t`	The time toward which all coupons should be present valued
`coupons_df`	A data.frame of details for each coupon. It should have the columns `payment_time` and `payment_size`.
`discount_factor_fcn`	A function specifying how the contract says future coupons should be discounted for this instrument in case the acceleration clause is triggered
`model_t`	The payment time beyond which coupons will be included in this computation

See Also

Other Bond Coupons: `accelerated_coupon_value()`, `coupon_value_at_exercise()`

variance_cumulation_from_vols

Create a variance cumulation function from a volatility term structure

Description

Given a volatility term structure, create a corresponding variance cumulation function. The function assumes piecewise constant forward volatility, with the final such forward volatility extending to infinity.

Usage

```
variance_cumulation_from_vols(vols_df)
```

Arguments

`vols_df`	A data.frame with numeric columns `time` (in increasing order) and `volatility` (not decreasing so quickly as to give negative forward variance)

Value

A function taking two time arguments, which returns the cumulated variance from the second to the first

Examples

```
vc = variance_cumulation_from_vols(
  data.frame(time=c(0.1,2,3),
  volatility=c(0.2,0.5,1.2)))
vc(1.5, 0)
```

---

warn_once_negative_default_intensity

Wrap a default intensity function so it warns at most once about negatives

---

Description

The default intensity (hazard rate) must be nonnegative. A solver such as `integrate_pde` calls the intensity function many times, on various stock-price grids and at various times, so a naive check would emit a flood of redundant warnings. This routine returns a function that behaves exactly like `default_intensity_fcn` but latches a flag in its enclosing environment so that, across all of its invocations, at most one warning about negative intensities is emitted in the usual `futile.logger` manner.

Usage

```
warn_once_negative_default_intensity(default_intensity_fcn)
```

Arguments

`default_intensity_fcn`
: A function for computing default intensity, with arguments `t`, `S`

Details

Create a fresh wrapper for each top-level pricing call so that the "already warned" state is scoped to that call and resets on the next.

Value

A function with the same interface as `default_intensity_fcn` that warns at most once about negative default intensities

See Also

Other Input Checking: `check_discount_factor_fcn()`, `check_survival_probability_fcn()`, `check_variance_cumulation_fcn()`

`ZeroCouponBond-class` *A simple contract paying the* `notional` *amount at the* `maturity`

Description

A simple contract paying the `notional` amount at the `maturity`

Fields

`notional` The amount that will be paid at `maturity`, conditional on survival

`recovery_rate` The proportion of notional that would be expected to be paid to bond holders after bankruptcy court proceedings

`discount_factor_fcn` A function specifying how cashflows should generally be discounted for this instrument

Methods

`optionality_fcn(v, ...)` Return a version of `v` at time `t` corrected for any optionality conditions.

`recovery_fcn(v, S, t, ...)` Return recovery value, given non-default values `v` at time `t`. Subclasses may be more elaborate, this method simply returns 0.0.

Index

∗ **American Exercise Equity Options**
american, 5
american_implied_volatility, 7
control_variate_pairs, 18
∗ **Black-Scholes**
fit_variance_cumulation, 33
implied_volatility, 42
∗ **Bond Coupon Acceleration**
accelerated_coupon_value, 3
coupon_value_at_exercise, 19
∗ **Bond Coupons**
accelerated_coupon_value, 3
coupon_value_at_exercise, 19
value_from_prior_coupons, 61
∗ **Dividends**
adjust_for_dividends, 4
shift_for_dividends, 54
time_adj_dividends, 58
∗ **Equity Dependent Default Intensity**
find_present_value, 28
fit_to_option_market_df, 32
fit_variance_cumulation, 33
form_present_value_grid, 35
implied_jump_process_volatility, 38
∗ **Equity Independent Default Intensity**
american, 5
american_implied_volatility, 7
black_scholes_on_term_structures, 10
blackscholes, 9
equivalent_bs_vola_to_jump, 22
equivalent_jump_vola_to_bs, 23
implied_volatilities, 39
implied_volatilities_with_rates_struct, 41
implied_volatility, 42
implied_volatility_with_term_struct, 44
∗ **European Options**
black_scholes_on_term_structures, 10
blackscholes, 9
implied_volatilities, 39
implied_volatilities_with_rates_struct, 41
implied_volatility, 42
implied_volatility_with_term_struct, 44
∗ **Greeks**
find_greeks, 25
GREEK_NAMES, 37
∗ **Implicit Grid Solver**
construct_implicit_grid_structure, 15
find_present_value, 28
form_present_value_grid, 35
infer_conforming_time_grid, 46
integrate_pde, 47
iterate_grid_from_timestep, 48
pde_matrix_solve, 50
take_implicit_timestep, 55
timestep_instruments, 57
∗ **Implied Volatilities**
american_implied_volatility, 7
equivalent_bs_vola_to_jump, 22
equivalent_jump_vola_to_bs, 23
fit_variance_cumulation, 33
implied_jump_process_volatility, 38
implied_volatilities, 39
implied_volatilities_with_rates_struct, 41
implied_volatility, 42
implied_volatility_with_term_struct, 44
∗ **Input Checking**
check_discount_factor_fcn, 13

check_survival_probability_fcn, 14
check_variance_cumulation_fcn, 15
warn_once_negative_default_intensity, 62
* **bond**
CallableBond-class, 13
* **calibration**
american_implied_volatility, 7
fit_variance_cumulation, 33
implied_jump_process_volatility, 38
* **callable**
CallableBond-class, 13
* **datasets**
TSLAMarket, 60
* **implied volatility**
american_implied_volatility, 7
fit_variance_cumulation, 33
implied_jump_process_volatility, 38
* **putable**
CallableBond-class, 13

accelerated_coupon_value, 3, *20*
accelerated_coupon_value(), *20*, *61*
adjust_for_dividends, 4
adjust_for_dividends(), *54*, *58*
american, 5, *8*
american(), *9, 10*, *12*, *18*, *23*, *25*, *40*, *42*, *44, 45*
american_implied_volatility, 7, *39*
american_implied_volatility(), *6*, *10*, *12*, *18*, *23–25*, *35*, *39, 40*, *42*, *44, 45*
AmericanOption, *30*, *53*
AmericanOption *(*AmericanOption-class*)*, 7
AmericanOption-class, 7

black_scholes_on_term_structures, 10, *45*
black_scholes_on_term_structures(), *6*, *9, 10*, *23*, *25*, *40*, *42*, *44, 45*
blackscholes, 9
blackscholes(), *6*, *9*, *12*, *23*, *25*, *40*, *42*, *44, 45*

CALL, 12
CallableBond *(*CallableBond-class*)*, 13
CallableBond-class, 13
check_discount_factor_fcn, 13
check_discount_factor_fcn(), *14, 15*, *62*
check_survival_probability_fcn, 14
check_survival_probability_fcn(), *14, 15*, *62*
check_variance_cumulation_fcn, 15
check_variance_cumulation_fcn(), *14*, *62*
construct_descending_bumps(), *27*, *37*
construct_implicit_grid_structure, 15
construct_implicit_grid_structure(), *30*, *37*, *46*, *48*, *50, 51*, *56*, *58*
construct_tridiagonals, 17, *50*
control_variate_pairs, 18
control_variate_pairs(), *6*, *9*
ConvertibleBond *(*ConvertibleBond-class*)*, 18
ConvertibleBond-class, 18
coupon_value_at_exercise, 19
coupon_value_at_exercise(), *3*, *61*
CouponBond *(*CouponBond-class*)*, 19
CouponBond-class, 19

detail_from_AnnivDates, 20

EquityOption *(*EquityOption-class*)*, 21
EquityOption-class, 21
equivalent_bs_vola_to_jump, 22
equivalent_bs_vola_to_jump(), *6*, *8–10*, *12*, *24, 25*, *35*, *39, 40*, *42*, *44, 45*
equivalent_jump_vola_to_bs, 23
equivalent_jump_vola_to_bs(), *6*, *8–10*, *12*, *23*, *35*, *39, 40*, *42*, *44, 45*
EuropeanOption, *30*, *53*
EuropeanOption *(*EuropeanOption-class*)*, 25
EuropeanOption-class, 25

find_greeks, 25
find_greeks(), *37*
find_present_value, *6*, *25*, *27*, 28, *31*, *34*, *38, 39*, *52*, *54*
find_present_value(), *17*, *32*, *35*, *37*, *39*, *46*, *48*, *50, 51*, *56*, *58*
fit_to_option_market, 30, *32*
fit_to_option_market_df, 32
fit_to_option_market_df(), *30*, *35*, *37*, *39*
fit_variance_cumulation, *31*, 33, *52*, *54*
fit_variance_cumulation(), *8*, *23, 24*, *30*, *32*, *37*, *39, 40*, *42*, *44, 45*
form_present_value_grid, 35

form_present_value_grid(), *17*, *30*, *32*, *35*, *39*, *46*, *48*, *50, 51*, *56*, *58*

greek_by_fd(), *27*, *37*
GREEK_NAMES, *26, 27*, 37
grid_delta_gamma(), *27*, *37*
GridPricedInstrument, *26*, *29*, *36*, *47*, *49*
GridPricedInstrument *(*GridPricedInstrument-class*)*, *37*
GridPricedInstrument-class, 37

implied_jump_process_volatility, 38, *43*
implied_jump_process_volatility(), *8*, *23, 24*, *30*, *32*, *35*, *37*, *40*, *42*, *44, 45*
implied_volatilities, 39, *42*
implied_volatilities(), *6*, *8–10*, *12*, *23–25*, *35*, *39*, *42*, *44, 45*
implied_volatilities_with_rates_struct, 41, *45*
implied_volatilities_with_rates_struct(), *6*, *8–10*, *12*, *23–25*, *35*, *39, 40*, *44, 45*
implied_volatility, *42*, 42, *45*
implied_volatility(), *6*, *8–10*, *12*, *23–25*, *35*, *39, 40*, *42*, *45*
implied_volatility_with_term_struct, *8*, *39*, *42*, 44
implied_volatility_with_term_struct(), *6*, *8–10*, *12*, *23–25*, *35*, *39, 40*, *42*, *44*
infer_conforming_time_grid, 46
infer_conforming_time_grid(), *17*, *30*, *37*, *48*, *50, 51*, *56*, *58*
integrate_pde, 47, *62*
integrate_pde(), *17*, *30*, *37*, *46*, *50, 51*, *56*, *58*
is.blank, 48
iterate_grid_from_timestep, 48
iterate_grid_from_timestep(), *17*, *30*, *37*, *46*, *48*, *51*, *56*, *58*

pde_matrix_solve, 50
pde_matrix_solve(), *17*, *30*, *37*, *46*, *48*, *50*, *56*, *58*
penalty_with_intensity_link, *31*, 51
price_with_intensity_link, *52*, 53
PUT, 54

resolve_bumps, *25*
resolve_bumps(), *27*, *37*
robust_greek(), *27*, *37*

safe_reprice(), *27*, *37*
shift_for_dividends, 54
shift_for_dividends(), *4*, *58*
spot_to_df_fcn, 55

take_implicit_timestep, 55
take_implicit_timestep(), *17*, *30*, *37*, *46*, *48*, *50, 51*, *58*
time_adj_dividends, 58
time_adj_dividends(), *4*, *54*
TIME_RESOLUTION_FACTOR, 59
TIME_RESOLUTION_SIGNIF_DIGITS, 59
timestep_instruments, 57
timestep_instruments(), *17*, *30*, *37*, *46*, *48*, *50, 51*, *56*
treasury_df, 59
treasury_df_raw, *59*, 60
TSLAMarket, 60

value_from_prior_coupons, *20*, 61
value_from_prior_coupons(), *3*, *20*
variance_cumulation_from_vols, *34*, 61

warn_once_negative_default_intensity, 62
warn_once_negative_default_intensity(), *14, 15*

ZeroCouponBond *(*ZeroCouponBond-class*)*, 63
ZeroCouponBond-class, 63