Financial Instruments Toolbox™ Release Notes

How to Contact MathWorks

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`www.mathworks.com/contact_TS.html`	Technical Support
`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

Financial Instruments Toolbox™ Release Notes

Contents

R2014a

R2013b

R2013a

R2012b

R2014a

Version: 1.3

New Features: Yes

Bug Fixes: Yes

Dual curve construction

Support for bootstrapping an interest rate curve using a different curve for discounting the cash flows with the following enhancements:

- `bootstrap` accepts a new optional input argument for `DiscountCurve`.
- `bootstrap` accepts a new bootstrapping instrument type called `FRA` for a forward rate agreement instrument.

For more information on using `bootstrap` for dual curve construction, see the example: “Dual Curve Bootstrapping”.

Dual curve pricing of caps, floors, and swaptions using the Black model

`capbyblk`, `floorbyblk`, and `swaptionbyblk` accept an optional input argument for `ProjectionCurve`.

Dual curve pricing of swaps and floating-rate notes

`swapbyzero` and `floatbyzero` have new examples to demonstrate pricing a swap and a floating-rate note with two curves.

Monte Carlo and analytical pricing of lookback options

Support for lookback options using closed-form solutions or Monte Carlo simulations.

Function	Purpose
`lookbackbycvgsg`	Calculate prices of European lookback fixed- and floating-strike options using the Conze-Viswanathan and Goldman-Sosin-Gatto models.
`lookbacksensbycvgsg`	Calculate prices and sensitivities of European fixed- and floating-strike lookback options using the Conze-Viswanathan and Goldman-Sosin-Gatto models.
`lookbackbyls`	Calculate prices of lookback fixed- and floating-strike options using the Longstaff-Schwartz model.
`lookbacksensbyls`	Calculate prices and sensitivities of lookback fixed- and floating-strike options using the Longstaff-Schwartz model.

Implied Black volatility computation for the SABR stochastic volatility model

Support for `blackvolbysabr` to calibrate the SABR model parameters and to compute SABR implied Black volatilities.

User-specified simulation paths for Longstaff-Schwartz pricing functions

Support for `optpricebysim` to calculate the price and sensitivities of European or American call or put options based on simulation results of the underlying asset. For American options, the Longstaff-Schwartz least squares method is used to calculate the early exercise premium.

`creditexposures` function to compute credit exposures from mark-to-market OTC contract values

Support for computing credit exposures as a part of a counterparty credit risk workflow. For more information, see `creditexposures`.

`exposureprofiles` function to derive credit exposure profiles from credit exposures

Support for computing various credit exposure profiles, including potential future exposure and expected exposure. For more information, see `exposureprofiles`.

Enhanced pricing functions for instruments and portfolios with cash flows between tree levels

The pricing algorithms for vanilla stock options have been enhanced to support `ExerciseDates` between tree levels. While `ExerciseDates` previously allowed only values that coincided with tree dates, the new pricing algorithm allows arbitrary `ExerciseDates` between the tree valuation date and tree maturity. For more information see the Bermuda option examples in `optstockbycrr`, `optstockbyeqp`, and `optstockbyitt`.

Swing option pricing example

New example for "Pricing Swing Options using the Longstaff-Schwartz Method".

R2013b

Version: 1.2

New Features: Yes

Bug Fixes: No

Support for Numerix CrossAsset Integration Layer (CAIL) API

Support for accessing Numerix® instruments and risk models.

Class	Purpose
`numerix`	Create a numerix object to set up the Numerix CrossAsset Integration Layer (CAIL) environment.

Method	Purpose
`numerix.parseResults`	Converts Numerix CAIL data to MATLAB® data types.

Kirk's approximation and Bjerksund-Stensland closed-form pricing models for spread options

Support pricing and sensitivity of spread options for the energy market using closed-form solutions.

Function	Purpose
`spreadbykirk`	Price European spread options using the Kirk pricing model.
`spreadsensbykirk`	Calculate European spread option prices and sensitivities using the Kirk pricing model.
`spreadbybjs`	Price European spread options using the Bjerksund-Stensland pricing model.
`spreadsensbybjs`	Calculate European spread option prices and sensitivities using the Bjerksund-Stensland pricing model.

Finite difference and Monte Carlo simulation pricing for American spread options

Support pricing and sensitivity of spread options for the energy market using Monte Carlo simulation.

Function	Purpose
spreadbyfd	Price European or American spread options using the Alternate Direction Implicit (ADI) finite difference method.
spreadsensbyfd	Calculate price and sensitivities of European or American spread options using the Alternate Direction Implicit (ADI) finite difference method.
spreadbyls	Price European or American spread options using Monte Carlo simulations.
spreadsensbyls	Calculate price and sensitivities for European or American spread options using Monte Carlo simulations.

Levy and Kemna-Vorst closed-form pricing and Monte Carlo simulation pricing for Asian options

Support pricing and sensitivity of Asian options for the energy market using Monte Carlo simulation and closed-form solutions.

Function	Purpose
asianbyls	Price European or American Asian options using the Longstaff-Schwartz model.
asiansensbyls	Calculate prices and sensitivities of European or American Asian options using the Longstaff-Schwartz model.
asianbykv	Price European geometric Asian options using the Kemna-Vorst model.

Function	Purpose
asiansensbykv	Calculate prices and sensitivities of European geometric Asian options using the Kemna-Vorst model.
asianbylevy	Price European arithmetic Asian options using the Levy model.
asiansensbylevy	Calculate prices and sensitivities of European arithmetic Asian options using the Levy model.

Additional CDS option pricing functionality for index swaptions

New example for Pricing a CDS Index Option.

Pricing functions for vanilla options using Monte Carlo simulation

Support pricing and sensitivity of vanilla options for the energy market using Monte Carlo simulation.

Function	Purpose
optstockbyls	Price European, Bermudan, or American vanilla options using the Longstaff-Schwartz model.
optstocksensbyls	Calculate European, Bermudan, or American vanilla option prices and sensitivities using the Longstaff-Schwartz model.

Hedging strategies using spread options example

New example for Hedging Strategies Using Spread Options.

Pricing European and American spread options example

New example for Pricing European and American Spread Options.

First-to-default (FTD) swaps example

New example for First-to-Default Swaps.

New function for risky present value of a basis point

Compatibility Considerations: Yes

`cdsrpv01` computes risky present value of a basis point (RPV01) for a credit default swap (CDS) and conforms to the industry standards (ISDA CDS Standard Model).

Compatibility Considerations

Compared with the previous version of Financial Instruments Toolbox™, there are minor changes in the values computed by `cdsbootstrap`, `cdsspread`, `cdsprice`, and `cdsoptprice` when the starting dates do not fall on a payment date. The affected output arguments are as follows:

- `cdsbootstrap`: `ProbData`, `HazData`
- `cdsspread`: `Spread`
- `cdsprice`: `Price`
- `cdsoptprice`: `Payer`, `Receiver`

While the magnitudes of the value changes are very small, they might affect users who depend on exact matches to previous values. These changes are caused by the modification of the way risky present value of a basis point (RPV01) is computed and these changes were made to better reflect the industry practice of paying CDS premiums only on specific payment dates.

`optimoptions` support

`optimoptions` support for `IRFitOptions`, `fitFunction` method, `hwcalbycap`, and `hwcalbyfloor`.

Functions moved from Financial Instruments Toolbox to Financial Toolbox

The following functions are moved from Financial Instruments Toolbox to Financial Toolbox™:

- `cdai`
- `cdprice`
- `cdyield`
- `tbilldisc2yield`
- `tbillprice`
- `tbillrepo`
- `tbillval01`
- `tbillyield`
- `tbillyield2disc`

R2013a

Version: 1.1

New Features: Yes

Bug Fixes: No

Pricing functions for options on floating-rate notes (FRNs)

Support for pricing a floating-rate note instrument with an option using tree models.

Function	Purpose
`optfloatbybdt`	Price an option for a floating-rate note using a Black-Derman-Toy interest-rate tree.
`optfloatbyhjm`	Price an option for a floating-rate note using a Heath-Jarrow-Morton interest-rate tree.
`optfloatbyhw`	Price an option for a floating-rate note using a Hull-White interest-rate tree.
`optfloatbybk`	Price an option for a floating-rate note using a Black-Karasinski interest-rate tree.
`instoptfloat`	Define the option instrument for a floating-rate note.

Pricing functions for FRNs with embedded options

Support for pricing a floating-rate note instrument with an embedded option using tree models.

Function	Purpose
`optemfloatbybdt`	Price an embedded option for a floating-rate note using a Black-Derman-Toy interest-rate tree.
`optemfloatbybk`	Price an embedded option for a floating-rate note using a Black-Karasinski interest-rate tree.
`optemfloatbyhjm`	Price an embedded option for a floating-rate note using a Heath-Jarrow-Morton interest-rate tree.
`optemfloatbyhw`	Price an embedded option for a floating-rate note using a Hull-White interest-rate tree.
`instoptemfloat`	Define the floating-rate note with an embedded option instrument.

Performance enhancements in implied volatility calculations

Improved performance for calculating implied volatility when using `impvbybjs` and `impvbyrgw`.

Calibration and Monte Carlo simulation of single-factor and multifactor interest-rate models, including Hull-White, Linear Gaussian, and LIBOR Market Models

Support for pricing interest-rate instruments for caps, floors, and swaptions using Monte Carlo simulation with Hull-White, Shifted Gaussian, and LIBOR Market Models. There are three new classes, three new methods, and four new functions.

Class	Purpose
`HullWhite1F`	Create a Hull-White one-factor model.
`LinearGaussian2F`	Create a two-factor additive Gaussian interest-rate model.
`LiborMarketModel`	Create a LIBOR Market Model.

Method	Purpose
`HullWhite1F.simTermStructs`	Simulate term structures for a Hull-White one-factor model.
`LinearGaussian2F.simTermStructs`	Simulate term structures for a two-factor additive Gaussian interest-rate model.
`LiborMarketModel.simTermStructs`	Simulate term structures for a LIBOR Market Model.

Function	Purpose
capbylg2f	Price caps using a Linear Gaussian two-factor model.
floorbylg2f	Price floors using a Linear Gaussian two-factor model.
swaptionbylg2f	Price European swaptions using a Linear Gaussian two-factor model.
blackvolbyrebonato	Compute the Black volatility for a LIBOR Market Model using the Rebonato formula.

R2012b

Version: 1.0

New Features: Yes

Bug Fixes: No

Merge of Fixed-Income Toolbox and Financial Derivatives Toolbox to Financial Instruments Toolbox

Compatibility Considerations: Yes

Fixed-Income Toolbox™ and Financial Derivatives Toolbox™ are merged into the new product Financial Instruments Toolbox.

Cap and floor floating-rate note pricing using trees

Support for pricing capped, collared, and floored floating-rate notes using the `CapRate` and `FloorRate` arguments.

Function	Purpose
`floatbybdt`	Price a capped floating-rate note using a Black-Derman-Toy interest-rate tree.
`floatbyhjm`	Price a capped floating-rate note using a Heath-Jarrow-Morton interest-rate tree.
`floatbyhw`	Price a capped floating-rate note using a Hull-White interest-rate tree.
`floatbybk`	Price a capped floating-rate note using a Black-Karasinski interest-rate tree.
`instfloat`	Create a capped floating-rate note instrument.
`instadd`	Add capped floating-rate note instruments to a portfolio.

Forward-swap pricing using trees or term structure

Support for interest-rate forward swaps using the new `StartDate` argument to define the future date for the swap instrument.

Function	Purpose
`swapbyzero`	Price a bond using a set of zero curves.
`swapbybdt`	Price a forward swap using a Black-Derman-Toy interest-rate tree.

Function	Purpose
swapbyhjm	Price a forward swap using a Heath-Jarrow-Morton interest-rate tree.
swapbyhw	Price a forward swap using a Hull-White interest-rate tree.
swapbybk	Price a forward swap using a Black-Karasinski interest-rate tree.
instswap	Create a forward swap instrument.
instadd	Add forward swap instruments to a portfolio.

Functions for fitting and extracting calibrated parameters from `IRFunctionCurve` objects

New enhancements for `IRFunctionCurve` object, including the ability to get calibrated parameters, the ability to specify linear inequality parameter constraints, and support for curve type in `fitSmoothingSpline` to be forward, zero, and discount.

LIBOR market model example

New example for mortgage prepayment that uses a LIBOR market model to generate interest-rate evolutions. For more information, see Prepayment Modeling with a Two Factor Hull White Model and a LIBOR Market Model.

Counterparty credit risk example

New example for computing the unilateral Credit Value (Valuation) Adjustment (CVA) for a bank holding a portfolio of vanilla interest-rate swaps with several counterparties. For more information, see Counterparty Credit Risk and CVA.

Conversion of error and warning message identifiers

Compatibility Considerations: Yes

For R2012b, error and warning message identifiers have changed in Financial Instruments Toolbox.

Compatibility Considerations

If you have scripts or functions that use message identifiers that changed, you must update the code to use the new identifiers. Typically, message identifiers are used to turn off specific warning messages, or in code that uses a `try/catch` statement and performs an action based on a specific error identifier.

For example, because Fixed-Income Toolbox and Financial Derivatives Toolbox merged to become Financial Instruments Toolbox, the `finfixed` and `finderiv` message identifiers have changed to `fininst`. If your code checks for `finfixed` or `finderiv` message identifiers, you must update it to check for `finisnt` instead.

To determine the identifier for an error, run the following command just after you see the error:

```
exception = MException.last;
MSGID = exception.identifier;
```

To determine the identifier for a warning, run the following command just after you see the warning:

```
[MSG,MSGID] = lastwarn;
```

This command saves the message identifier to the variable `MSGID`.