Credit Trader Suite User/Developer Guide

Lakshmi Krishnamurthy

v2.3 22 January 2014

Introduction

Credit Trader Suite of libraries aims to provide open source analytics and trading/valuation system solution suite for credit and fixed income products. To this end, it implements its functionality in a single library over 6 main components - CreditAnalytics, CreditProduct, CurveBuilder, FixedPointFinder, RegressionSuite, and SplineLibrary.

Overview and Problem Space Coverage

The main challenges that Credit Trader Suite attempts to address are:

- Implementation of day count conventions, holidays calendars, and rule-based period generators
- Abstract the functionality behind curves, parameters, and products onto defined interfaces
- Unified/standardized methods for curve calibrations, parameter and product implementers and constructors
- Environmental system to hold live ticks, closing marks, and reference data containers
- Enhanced analytics testing
- Ease of usage, installation, and deployment

While a number of other libraries - both Open Source implementations and proprietary systems such as Fincad, NumeriX, Algorithmics exist, they typically cater to the needs of the wider financial mathematics community, thus diluting their value towards treating credit products. Further, few of them inherently export a curve/product/product models that work well with products quotes, marks, and reference data sources, thereby forcing development teams to build their own integration layers from scratch. Finally, building

the components of functional credit-trading system requires additional layers of development over analytics.

Credit Trader Suite is an attempt to overcome these shortcomings. It aims to bring the aspects mentioned above together in one Open Source implementation that readily integrates onto existing systems.

Main Libraries and their Purpose

The libraries the constitute the Credit Trader Suite are:

- CreditAnalytics – Concerned with the construction and the implementation of the interfaces defined in CreditProduct, analytics environment management, and functional distribution.
- CreditProduct – Focused on the core analytics, and the curve, the parameter, and the product definitions.
- CurveBuilder – Provides the functionality for highly customized discount, forward, credit, and FX curve construction, customized with wide variety of basis splines, calibration instrument types and measures.
- SplineLibrary – Provides the functionality for building, calibrating, and evaluating different kinds of splines for use in latent state representation.
- FixedPointFinder – Provides the implmentation of all the standard bracketing and open root finding techniques, along with a customizable and configurable framework that separates the initilization/bracketing functionality from the eventual root search.
- RegressionSuite – This aims to ease testing of analytics, measurement and generation of the execution time distribution, as well as release performance characterization.

CreditAnalytics Description and Problem Space Coverage

CreditAnalytics provides the functionality behind creation, calibration, and implementation of the curve, the parameter, and the product interfaces defined in CreditProduct. It also implements a curve/parameter/product/analytics management environment, and has packaged samples and testers.

CreditAnalytics library achieves its design goal by implementing its functionality over several packages:

- Curve calibration and creation: Functional implementation and creation factories for rates curves, credit curves, and FX curves of al types
- Market Parameter implementation and creation: Implementation and creation of quotes, component/basket market parameters, as well as scenario parameters.
- Product implementation and creation: Implementation and creation factories for rates products (cash/EDF/IRS), credit products (bonds/CDS), as well as basket products.
- Reference data/marks loaders: Loaders for bond/CDX, as well a sub-universe of closing marks
- Calculation Environment Manager: Implementation of the market parameter container, manager for live/closing curves, stub/client functionality for serverization/distribution, input/output serialization.
- Samples: Samples for curve, parameters, product, and analytics creation and usage
- Unit functional testers: Detailed unit scenario test of various analytics, curve, parameter, and product functionality.

CreditProduct Description and Problem Space Coverage

CreditProduct aims to define the functional and behavioral interfaces behind curves, products, and different parameter types (market, valuation, pricing, and product parameters). To facilitate this, it implements various day count conventions, holiday sets, period generators, and calculation outputs.

CreditProduct library achieves its design goal by implementing its functionality over several packages:

- Dates and holidays coverage: Covers a variety of day count conventions, 120+ holiday locations, as well as custom user-defined holidays
- Curve and analytics definitions: Defines the base functional interfaces for the variants of discount curves, credit curves, and FX curves
- Market Parameter definitions: Defines quotes, component/basket market parameters, and custom scenario parameters
- Valuation and Pricing Parameters: Defines valuation, settlement/work-out, and pricing parameters of different variants
- Product and product parameter definitions: Defines the product creation and behavior interfaces for Cash/EDF/IRS (all rates), bonds/CDS (credit), and basket bond/CDS, and their feature parameters.
- Output measures container: Defines generalized component and basket outputs, as well customized outputs for specific products

CurveBuilder Description and Problem Space Coverage

CurveBuilder provides the functionality for highly customized discount, forward, credit, and FX curve construction, customized with wide variety of basis splines, calibration instrument types and measures.

CurveBuilder library achieves its design goal by implementing its functionality over several packages:

- Latent State Representation Package: The latent state representation package implements the latent state, the quantification metric/manifest measure, its labels, the merge stretch and its manager.
- Latent State Estimator Package: The latent state estimator package provides functionality to estimate the latent state, local/global state construction controls, constraint representation, and linear/non-linear calibrator routines.

- Latent Curve State Package: The latent curve state package provides implementations of latent state representations of discount curve, forward curve, zero curve, credit curve, FX Basis curve, and FX forward curve.
- Latent State Creator Package: The latent curve state package provides implementations of the constructor factories that create discount curve, forward curve, zero curve, credit curve, FX Basis curve, and FX forward curve.
- Analytics Definition Package: The analytics definition package provides definitions of the generic curve, discount curve, forward curve, zero curve, credit curve, FX Basis curve, and FX forward curve, turns list, and their construction inputs.
- Rates Analytics Package: The rates analytics package provides definitions of the discount curve, the forward curve, the zero curve, the discount factor and the forward rate estimators, the turns list, and their construction inputs.

FixedPointFinder Description and Problem Space Coverage

RootFinder achieves its design goal by implementing its functionality over several packages:

- Framework: Framework to accommodate bracketing/open convergence initialization, execution customization/configuration, iterative variate evolution, and search termination detection
- Bracket Initialization Techniques: Implementation of the different techniques for the initial bracket extraction.
- Open Method Convergence Zone Initialization Techniques: Implementation of the different techniques for the convergence zone starting variate extraction.
- Iterative Open Methods: Implementation of the iterative Open Methods - Newton-Raphson and Secant Methods
- Iterative Bracketing Primitive Methods: Implementation of the iterative bracketing primitives – Bisection, False Position, Quadratic Interpolation, Inverse Quadratic Interpolation, and Ridder.

- Iterative Bracketing Compound Methods: Implementation of the iterative bracketing compound methodologies – Brent's and Zheng's methods.
- Search Initialization Heuristics: Implementation of a number of search heuristics to make the search targeted and customized.
- Samples: Samples for the various bracketing and the open methods, their customization, and configuration.
- Documentation: Literature review, framework description, mathematical and formulation details of the different components, root finder synthetic knowledge unit (SKU) composition, and module and API usage guide.
- Regression Tests: Statistical regression analysis and dispersion metric evaluation for the initialization and the iteration components of the different bracketing and open root finder methodologies.

RegressionSuite Description and Problem Space Coverage

RegressionSuite aims to incorporate measurement of the startup lag, measurement of accurate execution times, generating execution statistics, customized input distributions, and processable regression specific details as part of the regular unit tests.

RegressionSuite library achieves its design goal by implementing its functionality over several packages:

- Regression Engine: Provides control for distribution set, invocation strategy, and load.
- Unit Regression Executor: Framework that implements set up and tear-down, as well as generate run details
- Regression Statistics: Execution time distribution, start-up and other event delay measurements, and system load monitoring
- Regression Output: Fine grained regressor level output, module aggregated output, sub-element execution time estimation.

- Regressor Set: Module containing set of regressors, group level turn on/off and execution control
- Regression Utilities: Formatting and tolerance checking.

SplineLibrary Description and Problem Space Coverage

SplineLibrary provides the functionality for building, calibrating, and evaluating different kinds of splines for use in latent state representation. It implements the functionality behind spline design, spline constructions, customization, calibration, and evaluation of a wide variety of spline types and basis functions.

SplineLibrary achieves its design goals by implementing its functionality over several packages the perform the following:

- Univariate Function Package: The univariate function package implements the individual univariate functions, their convolutions, and reflections.
- Univariate Calculus Package: The univariate calculus package implements univariate difference based arbitrary order derivative, implements differential control settings, implements several integrand routines, and multivariate Wengert Jacobian.
- Spline Parameters Package: The spline parameters package implements the segment and stretch level construction, design, penalty, and shape control parameters.
- Splinbe Basis Function Set Package: The spline basis function set package implements the basis set, parameters for the different basis functions, parameters for basis set construction, and parameters for B Spline sequence construction.
- Spline Segment Package: The spline segment package implements the segment's inelastic state, the segment basis evaluator, the segment flexure penalizer, computes the segment monotonicity behavior, and implements the segment's complete constitutive state.
- Spline Stretch Package: The spline stretch package provides single segment and multi segment interfaces, builders, and implementations, along with custom boundary settings.

- Spline Grid/Span Package: The spline grid/span package provides the multi-stretch spanning functionality. It specifies the span interface, and provides implementations of the overlapping and the non-overlapping span instances. It also implements the transition splines with custom transition zones.
- Spline PCHIP Package: The spline PCHIP package implements most variants of the local piece-wise cubic Hermite interpolating polynomial smoothing functionality. It provides a number of tweaks for smoothing customization, as well as providing enhanced implementations of Akima, Preuss, and Hagan-West smoothing interpolators.
- Spline B Spline Package: The spline B Spline package implements the raw and the processed basis B Spline hat functions. It provides the standard implementations for the monic and the multic B Spline Segments. It also exports functionality to generate higher order B Spline Sequences.
- Tension Spline Package: The tension spline package implements closed form family of cubic tension splines laid out in the basic framework outlined in Koch and Lyche (1989), Koch and Lyche (1993), and Kvasov (2000).

Design Objectives

This section covers the design objectives across several facets – functional, software, system, usage, and deployment aspects.

Financial Feature Design Attributes

The chief design aims from a financial functionality angle are:

- Interface representations of curve, parameter, and products
- Separation of the creation and the implementation modules from the exposed functional interface behavior
- Re-usable functional and behavioral abstractions around financial products
- Provide “open” public implementations of the standard analytics functionality such as day count conventions, holidays, date representations, rule based period generation etc
- Abstraction of the quote, the market parameter and the pricing structures
- Abstraction and implementation of the standard curve calibration

Software Feature Design Attributes

The chief design aims from a software design angle are:

- Logical functionality identification/localization and functional group partitioning
- Clearly defined interface structure
- Implementation and creation factory bodies
- Reach and interaction through interfaces only

System Feature Design Attributes

The key system design aims are:

- Functionality needs to be readily serverizable and distributable
- Provide built in serialization, marshalling, and persistence of all the main components
- Management containers around the products, the curves, and the parameter sets, and establishing the execution control environment

Analytics Usage Design Objectives

The key usage design goals are:

- The analytics modules should provide comprehensive credit product risk, valuation, and pricing functionality from a set of functional API
- Ease of use
- Flexible
- When direct object access is needed, use only through the object model interface (and amend the interface as appropriate)

Test Design Objectives

The key testing design goals in this area are:

- Comprehensive unit testing of curve, parameters, and product implementation
- Extensive composite scenario testing
- Environment and server components testing
- Release time performance characterization and execution time and execution resource statistics calculation

Installation, Dependency, and Deployment Design Objectives

The key design goals in this area are:

- Minimize dependency on external modules
- Ease of installation and deployment
- Customizability – for non-standard setups – through the supplied configuration file.

Credit Product

Credit Product Library consists of the following 14 packages:

1. Date & Time Manipulators: This contains functionality for creating, manipulating, and adjusting dates, as well as time instants (to nano-second granularity).

2. Day-count Parameters, Conventions, and Date Adjustment Operations: This contains the functionality for day count generation and date adjustment according to specific rules. It also holds parameters needed for specific day count conventions.

3. Location Specific Standard Holiday Set: This contains all the non-weekend holidays that correspond to a specific location jurisdiction, and its description. Each location implements it holidays in a separate class.

4. Custom Holidays: This provides the ability to specify custom holidays, if the standard ones provided earlier are insufficient. Different types of holidays can be added – variable, fixed, static, as well as weekends for a given location.

5. Cash flow Period: This contains the cash flow period functionality, as well as place holders for the period related different curve factors.

6. Analytics Support Utilities: This contains utility functions for manipulating the core Credit Product modules, generic utility functions, and a logger.

7. Quotes, Market, and Scenario Parameters Definitions: This contains the classes that implement the definitions for all parameters except product feature parameters – quotes, calibration parameters, market parameters, tweak parameters, and the scenario curves.

8. Pricer Parameters: This contains the pricing parameters corresponding to a given product and model.

9. Quoting Parameters: This contains the quoting parameters needed to interpret a product quote.

10. Valuation Parameters: This contains all the non-market and non-product parameters needed for valuing a product at a given date.

11. Product Definitions: This contains interface definitions for all products, along with definitions for credit, rates, and FX components and specific credit/rates/FX products, and baskets.

12. Product Parameters: This contains the implementations of the features required for a complete construction of an instance of the product.

13. Product RV and Batch Calculation Outputs: This contains the bulk results of pricing and relative value calculation for the products.

14. Serializer: This interface defines the core object serialization methods – serialization into and de-serialization out of byte arrays, as well as the object serializer version.

Credit Product: Date Time Manipulators

Date Time Manipulators are implemented in the package *org.drip.analytics.date*. It contains functionality for creating, manipulating, and adjusting dates, as well as time instants (to nano-second granularity).

The functionality is implemented in 2 classes: DateTime and JulianDate, and both are serializable.

JulianDate

This class provides a comprehensive representation of Julian date and date manipulation functionality. It exports the following functionality:

- Explicit date construction, as well as date construction from several input string formats/today
- Date Addition/Adjustment/Elapsed/Difference, add/subtract days/weeks/months/years and tenor codes
- Leap Year Functionality (number of leap days in the given interval, is the given year a leap year etc.)
- Generate the subsequent IMM date (EDF/CME IMM date, CDS/Credit ISDA IMM date etc)
- Year/Month/Day in numbers/characters
- Days Elapsed/Remaining, is EOM
- Comparison with the “Other”, equals/hash-code/comparator
- Export the date to a variety of date formats (Oracle, Julian, Bloomberg)
- Serialization/De-serialization to and from Byte Arrays

DateTime

This class provides the representation of the instantiation-time date and time objects. It provides the following functionality:

- Instantiation-time and Explicit Date/Time Construction
- Retrieval of Date/Time Fields
- Serialization/De-serialization to and from Byte Arrays

Credit Product: Day Count Parameters, Conventions, and Date Adjustment Operations

Day Count Calculators are implemented in the package *org.drip.analytics.daycount*. It contains the functionality for day count generation and date adjustment according to specific rules. It also holds parameters needed for specific day count conventions.

The functionality is implemented across 19 classes: ActActDCParams, Convention, and DateAdjustParams, DateEOMAdjustment, DC28_360, DC30_360, DC30_365, DC30_Act, DC30E_360, DCAct_360, DC30_364, DC30_365, DC30_365L, DCAct_Act_ISDA, DCAct_Act, DCFCalculator, DCNL_360, DCNL_365, and DCNL_Act.

ActActDCParams

This class contains parameters to represent Act/Act day count. It exports the following functionality:

- Frequency/Start/End Date Fields
- Serialization/De-serialization to and from Byte Arrays

Convention

This class contains flags that indicate where the holidays are loaded from, as well as the holiday types and load rules. It exports the following date related functionality:

- Add business days according to the specified calendar
- The Year Fraction between any 2 days given the day count type and the holiday calendar

- Adjust/roll to the next working day according to the adjustment rule
- Holiday Functions - is the given day a holiday/business day, the number and the set of holidays/business days between 2 days.
- Calendars and Day counts - Available set of day count conventions and calendars, and the weekend days corresponding to a given calendar.

DateAdjustParams

This class contains the parameters needed for adjusting dates. It exports the following functionality:

- Accessor for holiday calendar and adjustment type
- Serialization/De-serialization to and from Byte Arrays

DateEOMAdjustment

This class holds the applicable adjustments for a given date pair. It exposes the following functionality:

- Static Methods for creating 30/360, 30/365, and EOMA Date Adjustments
- Export Anterior and Posterior EOM Adjustments

DCFCalculator

This interface is the stub for all the day count convention functionality. It exposes the base/alternate day count convention names, the year-fraction and the days accrued.

DC28_360

This class implements the 28/360 day count convention.

DC30_360

This class implements the 30/360 day count convention.

DC30_365

This class implements the 30/365 day count convention.

DC30_Act

This class implements the 30/Act day count convention.

DC30E_360

This class implements the 30E/360 day count convention.

DCAct_360

This class implements the Act/360 day count convention.

DCAct_364

This class implements the Act/364 day count convention.

DCAct_365

This class implements the Act/365 day count convention.

DCAct_365L

This class implements the Act/365L day count convention.

DCAct_Act_ISDA

This class implements the Act/Act ISDA day count convention.

DCAct_Act

This class implements the Act/Act day count convention.

DCNL_360

This class implements the NL/360 day count convention.

DCNL_365

This class implements the NL/365 day count convention.

DCNL_Act

This class implements the NL/Act day count convention.

Credit Product: Location Specific Standard Holiday Set

Location Specific Holidays are implemented in the package *org.drip.analytics.holset*. It contains all the holidays that correspond to a specific location jurisdiction, and its description.

The functionality is implemented in its own location qualified class instance - each of which is an instance of the LocationHoliday interface.

LocationHoliday

LocationHoliday is an interface that is implemented by all the Location Holiday classes. It exposes the specific holiday location, as well as the set of location-specific holidays.

Other classes in this package provide explicit holidays and the locale name. So far, Credit Product has about 130 locales implemented – please consult the Credit Analytics site for what they are.

Credit Product: Custom Holidays

Custom Holiday creators are implemented in the package *org.drip.analytics.holiday*. It provides the ability to add holidays, it the standard ones provided earlier are insufficient. Different types of holidays can be added – variable, fixed, static, as well as weekends for a given location.

Different holiday types are implemented in their own classes – they are Static, Fixed, and Variable, each of which extends the Base holiday class. Weekend is implemented in a separate class. All holiday instances for a given Locale are maintained on a named holiday container.

Base

Base is an abstraction around holiday and description. Abstract function generates an optional adjustment for weekends in a given year.

Weekend

Weekend holds the left and the right weekend days. It provides functionality to retrieve them, check if the given day is a weekend, and serialize/de-serialize weekend days.

Static

Static implements a complete date as a specific holiday.

Fixed

Fixed contains the fixed holiday's date and month. Holidays are generated on a per-year basis by applying the year, and by adjusting the date generated.

Variable

Variable class contains the rule characterizing the variable holiday's month, day in week, week in month, and the weekend days. Specific holidays in the given year are generated using these rules.

Locale

Locale contains the set of regular holidays and the weekend holidays for a location. It also provides the functionality to add custom holidays and weekends.

Credit Product: Cash flow Period

Cash flow period functionality is implemented in the package *org.drip.analytics.period*. It contains the cash flow period functionality, as well as place holders for the period related different curve factors.

Functionality in this package is implemented across 4 classes – Period, CouponPeriod, CouponPeriodCurveFactors, and LossPeriodCurveFactors.

Period

Period serves as a holder for the period dates: period start/end, period accrual start/end, pay, and full period day count fraction.

CouponPeriod

CouponPeriod extends the period class with the following coupon day-count specific parameters: frequency, reset date, and accrual day-count convention. It also exposes static methods to construct coupon period sets starting backwards/forwards, as well as merge coupon periods.

CouponPeriodCurveFactors

CouponPeriodCurveFactors is an enhancement of the period class using the following period measures: start/end survival probabilities, start/end notionals, and period start/end discount factor.

LossPeriodCurveFactors

LossPeriodCurveFactors is an implementation of the period class enhanced by the following period measures:

- Start/end survival probabilities
- Period effective notional/recovery/discount factor

Credit Product: Analytics Support Utilities

Analytics Support functionality is implemented in the package *org.drip.analytics.support*. It contains utility functions for manipulating the Credit Product modules, case insensitive maps, and a logger.

Functionality in this package is implemented across 4 classes – AnalyticsHelper, CaseInsensitiveHashMap, CaseInsensitiveTreeMap, and Logger.

AnalyticsHelper

AnalyticsHelper contains the collection of the analytics related utility functions used by the modules. The following is the functionality that it exposes:

- Yield to Discount Factor, and vice versa.
- Map Bloomberg Codes to CreditAnalytics Codes.
- Generate rule-based curve bumped nodes.
- Generate loss periods using a variety of different schemes.
- Aggregate/disaggregate coupon period lists.

CaseInsensitiveHashMap

CaseInsensitiveMap implements a case insensitive key in a hash map.

CaseInsensitiveTreeMap

CaseInsensitiveMap implements a case insensitive key in a hash map

Logger

The Logger class implements level-set logging, backed by either the screen or a file. Logging always includes time-stamps, and happens according to the level requested.

Credit Product: Quote, Market, and Scenario Parameters

Quote, Market, Tweak, and Scenario parameter definitions are specified in the package *org.drip.param.definition*. It contains the classes that implement the definitions for all parameters except product feature parameters – quotes, calibration parameters, market parameters, tweak parameters, and the scenario curves.

Functionality in this package is implemented across 10 classes and 5 groups – CalibrationParams, the quote parameters group (Quote, and ComponentQuote), the tweak parameters group (NodeTweakParams, and CreditNodeTweakParams), and the scenario curves group (RatesScenarioCurve and CreditScenarioGroup), and the market parameters group (ComponentMarketParams, BasketMarketParams, MarketParams).

CalibrationParams

CalibrationParams the calibration parameters - the measure to be calibrated, the type/nature of the calibration to be performed, and the work-out date to which the calibration is done.

Quote

Quote interface contains the stubs corresponding to a product quote. It contains the quote value and quote instant for the different quote sides (bid/ask/mid).

ComponentQuote

ComponentQuote abstract class holds the different types of quotes for a given component. It contains a single market field/quote pair, but multiple alternate named quotes (to accommodate quotes on different measures for the component).

NodeTweakParams

NodeTweakParams is the place-holder for the scenario tweak parameters, for either a specific curve node, or the entire curve (flat). Parameter bumps can be parallel or proportional.

CreditNodeTweakParams

CreditNodeTweakParams is the place-holder for the credit curve scenario tweak parameters: the measure, the curve node, and the nodal calibration type (entire curve/flat or a given tenor point).

RatesScenarioCurve

RatesScenarioCurve abstract class exposes the interface the constructs scenario discount curves. The following curve construction scenarios are supported:

- Base, flat/tenor up/down by arbitrary bumps.
- Tenor bumped discount curve set - keyed using the tenor.
- NTP-based custom scenario curves.

CreditScenarioCurve

CreditScenarioCurve abstract class exposes the bump parameters and the curves for the following credit curve scenarios:

- Base, Flat Spread/Recovery bumps.
- Spread/Recovery Tenor bumped up/down credit curve sets keyed using the tenor.
- NTP-based custom scenario curves.

ComponentMarketParams

ComponentMarketParams abstract class provides stub for the ComponentMarketParamsRef interface. It is a place-holder for the market parameters needed to value the component object – the discount curve, the forward curve, the treasury curve, the EDSF curve, the credit curve, the component quote, the treasury quote map, and the fixings map.

BasketMarketParams

BasketMarketParams class extends the BaketMarketParamsRef for a specific scenario. It provides access to maps holding named discount curves, named credit curves, named treasury quote, named component quote, and fixings object.

MarketParams

MarketParams is the place-holder for the comprehensive suite of the market set of curves for the given date. It exports the following functionality:

- Add/remove/retrieve scenario discount curve.
- Add/remove/retrieve scenario zero curve.
- Add/remove/retrieve scenario credit curve.

- Add/remove/retrieve scenario recovery curve.
- Add/remove/retrieve scenario FXForward curve.
- Add/remove/retrieve scenario FXBasis curve.
- Add/remove/retrieve scenario fixings.
- Add/remove/retrieve Treasury/component quotes.
- Retrieve scenario CMP/BMP.
- Retrieve map of flat rates/credit/recovery CMP/BMP.
- Retrieve double map of tenor rates/credit/recovery CMP/BMP.
- Retrieve rates/credit scenario generator.

Credit Product: Pricing Parameters

Pricing parameter is implemented in the package org.drip.param.pricer. It contains the pricing parameters corresponding to a given product and model.

Currently only the credit-pricing model is implemented – it is implemented in PricerParams.

PricerParams

PricerParams contains the pricer parameters - the discrete unit size, calibration mode on/off, survival to pay/end date, and the discretization scheme.

Credit Product: Quoting Parameters

Pricing parameter is implemented in the package org.drip.param.quoting. This contains the quoting parameters needed to interpret a product quote.

Functionality in this package is implemented across 3 classes – MeasureInterpreter, QuotedSpreadInterpreter, and YieldInterpreter.

MeasureInterpreter

MeasureInterpreter is the abstract shell stub class from which all product measure quoting parameters are derived. It contains fields needed to interpret a measure quote.

QuotedSpreadInterpreter

QuotedSpreadInterpreter holds the fields needed to interpret a Quoted Spread Quote. It contains the contract type and the coupon.

YieldInterpreter

YieldInterpreter holds the fields needed to interpret a Yield Quote. It contains the quote day count, quote frequency, quote EOM Adjustment, quote Act/Act parameters, and quote Calendar.

Credit Product: Valuation Parameters

Valuation parameters are implemented in the package *org.drip.param.valuation*. It contains all the non-market and non-product parameters needed for valuing a product at a given date.

Functionality in this package is implemented across 4 classes – QuotingParams, CashSettleParams, WorkoutInfo, and ValuationParams.

CashSettleParams

CashSettleParams is the place-holder for the cash settlement parameters for a given product. It contains the cash settle lag, the calendar, and the date adjustment mode.

QuotingParams

QuotingParams holds the parameters needed to interpret the input quotes. It contains the quote day count, quote frequency, quote EOM Adjustment, quote Act/Act parameters, and quote Calendar. It also indicates if the native quote is spread based.

ValuationParams

ValuationParams is the place-holder for the valuation parameters for a given product. It contains the valuation and the cash pay/settle dates, as well as the calendar. It also exposes a number of methods to construct standard valuation parameters.

WorkoutInfo

WorkoutInfo is the place-holder for the work-out parameters. It contains the date, the factor, the type, and the yield of the work-out.

Credit Product: Product Definitions

Product definitions are implemented in the package *org.drip.product.definition*. It contains interface definitions for all products, along with definitions for credit, rates, and FX components and specific credit/rates/FX products, and baskets.

Product definitions are implemented in different groups – base component group (ComponentMarketParamsRef, Component, CalibrateComponent), base basket group (BasketMarketParamRef, BasketProduct), RatesComponent, Credit Component Group (CreditComponent, CreditDefaultSwap, BondProduct, Bond), and FX Component group (FXSpot and FXForward).

ComponentMarketParamRef

ComponentMarketParamRef interface provides stubs for component name, IR curve, forward curve, credit curve, TSY curve, and EDSF curve needed to value the component.

Component

Component abstract class extends ComponentMarketParamRef and provides the following methods:

- Get the component's initial notional, notional, and coupon.
- Get the Effective date, Maturity date, First Coupon Date.
- List the coupon periods.
- Set the market curves - discount, TSY, forward, Credit, and EDSF curves.
- Retrieve the component's settlement parameters.
- Value the component using standard/custom market parameters.

- Retrieve the component's named measures and named measure values.

CalibratableComponent

CalibratableComponent abstract class provides implementation of Component's calibration interface. It exposes stubs for getting/setting the component’s calibration code, generate calibrated measure values from the market inputs, and compute micro-Jacobians (QuoteDF and PVDF micro-Jacks).

BasketMarketParamRef

BasketMarketParamRef interface provides stubs for component's IR and credit curves that constitute the basket.

BasketProduct

BasketProduct abstract class extends BasketMarketParamRef. It provides methods for getting the basket’s components, notional, coupon, effective date, maturity date, coupon amount, and list of coupon periods.

RatesComponent

RatesComponent is the abstract class that extends CalibratableComponent on top of which all rates components are implemented.

CreditComponent

CreditComponent is the base abstract class on top of which all credit components are implemented. Its methods expose Credit Valuation Parameters, and coupon/loss cash flows.

CreditDefaultSwap

CreditDefaultSwap is the base abstract class implements the pricing, the valuation, and the RV analytics functionality for the CDS product.

BondProduct

BondProduct interface implements the product static data behind bonds of all kinds. Bond static data is captured in a set of 11 container classes – BondTSYParams, BondCouponParams, BondNotionalParams, BondFloaterParams, BondCurrencyParams, BondIdentifierParams, ComponentValuationParams, ComponentRatesValuationParams, ComponentCreditValuationParams, ComponentTerminationEvent, BondFixedPeriodParams, and one EmbeddedOptionSchedule object instance each for the call and the put objects. Each of these parameter sets can be set separately.

Bond

Bond abstract class implements the pricing, the valuation, and the RV analytics functionality for the bond product.

FXSpot

FXSpot is the abstract class exposes the functionality behind the FXSpot Contract. Derived implementations return the spot date and the currency pair.

FXForward

FXForward is the abstract class exposes the functionality behind the FXForward Contract. Derived implementations expose the primary/secondary codes, the effective/maturity dates, the currency pair, imply the discount curve basis and the FX Forward from a set of market parameters. The value function carries out a full valuation.

Credit Product: Product Parameters

Product parameter definitions are implemented in the package org.drip.product.params. It contains the implementations of the features required for a complete construction of an instance of the product.

Product parameters are implemented across 20 classes. Validatable is the base interface that underpins most of them. Others are identifier parameters (CDXIdentifier, CDXRefDataParams, IdentifierSet, StandardCDXParams), CouponSetting, CreditSetting, CurrencySet, EmbeddedOptionSchedule, FactorSchedule, NotionalSetting, PeriodGenerator, PeriodSet, FloaterSetting, RatesSeting, TerminationSetting, QuoteConvention, Treasury Parameters (TreasuryBenchmark, TsyBmkSet), and CurrencyPair.

Validatable

Validatable interface defines the validate function, which validates the current object state.

CDXIdentifier

CDXIdentifier implements the creation and the static details of the all the NA, EU, SovX, EMEA, and ASIA standardized CDS indexes. It contains the index, the tenor, the series, and the version of a given CDX.

CDXRefDataParams

CDXRefDataParams contains the complete set of reference data that corresponds to the contract of a standard CDX. It consists of the following category and fields:

- Descriptive => Index Label, Index Name, Curve Name, Index Class, Index Group Name, Index Short Group.
- Name, Index Short Name, Short Name.
- Issuer ID => Curve ID, Red ID, Series, Version, Curvy Curve ID, Location, Bloomberg Ticker.
- Quote Details => Quote As CDS.
- Date => Issue Date, Maturity Date.
- Coupon Parameters => Coupon Rate, Currency, Day Count, Full First Stub, Frequency.
- Component Details => Original Count, Defaulted Count.
- Payoff Details => Knock out on Default, Pay Accrued Amount, Recovery on Default.
- Other => Index Life Span, Index Factor

IdentifierSet

IdentifierSet contains the component's identifier parameters - ISIN, CUSIP, ID, and ticker.

StandardCDXParams

StandardCDXParams implements the parameters used to create the standard CDX - the coupon, the number of components, and the currency.

CouponSetting

CouponSetting contains the coupon type, schedule, and the coupon amount for the component. If available, the floor and/or the ceiling may also be applied to the coupon, in a pre-determined order of precedence.

CreditSetting

CreditSetting contains the credit related valuation parameters - use default pay lag, use curve or the component recovery, component recovery, credit curve name, and whether there is accrual on default.

CurrencySet

CurrencySet contains the component's trade, the coupon, and the redemption currencies.

EmbeddedOptionSchedule

EmbeddedOptionSchedule is a place-holder for the embedded option schedule for the component. It contains the schedule of exercise dates and factors, the exercise notice period, and the option is to call or put. Further, if the option is of the type fix-to-float on exercise, contains the post-exercise floater index and floating spread. If the exercise is not discrete (American option), the exercise dates/factors are discretized according to a pre-specified discretization grid.

FactorSchedule

FactorSchedule contains the array of dates and factors.

NotionalSetting

NotionalSetting contains the product's notional schedule and the amount. It also incorporates hints on how the notional factors are to be interpreted - off of the original or the current notional. Further flags tell whether the notional factor is to be applied at the start/end/average of the coupon period.

PeriodGenerator

PeriodGenerator generates the component coupon periods from flexible inputs. Periods can be generated forwards or backwards, with long/short stubs. For customization, date adjustment parameters can be applied to each cash flow date of the period - effective, maturity, period start start/end, accrual start/end, pay and reset can each be generated according to the date adjustment rule applied to nominal period start/end.

PeriodSet

PeriodSet is the place holder for the component’s period generation parameters. It contains the component's date adjustment parameters for period start/end, period accrual start/end, effective, maturity, pay and reset, first coupon date, and interest accrual start date.

FloaterSetting

FloaterSetting contains the component's floating rate parameters. It holds the rate index, floater day count, and one of either the coupon spread or the full current coupon.

RatesSetting

RatesSetting contains the rate related valuation parameters - the discount curves to be used for discounting the coupon, the redemption, the principal, and the settle cash flows.

TerminationSetting

TerminationSetting class contains the current "liveness" state of the component, and, if inactive, how it entered that state.

QuoteConvention

QuoteConvention contains the Component Market Convention Parameters - the quote convention, the calculation type, the first settle date, and the redemption amount.

TreasuryBenchmark

TreasuryBenchmark contains component treasury benchmark parameters - the treasury benchmark set, and the names of the treasury and the EDSF IR curves.

TsyBmkSet

TsyBmkSet contains the treasury benchmark set - the primary treasury benchmark, and an array of secondary treasury benchmarks.

CurrencyPair

CurrencyPair class contains the numerator currency, the denominator currency, the quote currency, and the PIP Factor.

Credit Product: Product RV and Batch Calculation Outputs

Product bulk outputs are implemented in the package *org.drip.analytics.output*. It contains the bulk results of pricing and relative value calculation for the products.

Outputs are implemented in 6 classes – ComponentMeasures, bond specific calculation outputs (ExerciseInfo, BondCouponMeasures, BondWorkoutMeasures, BondRVMeasures), and BasketMeasures.

ComponentMeasures

ComponentMeasures is the place-holder for analytical single component output measures, optionally across scenarios. It contains measure maps for the following scenarios:

- Unadjusted Base IR/credit curves
- Flat delta/gamma bump measure maps for IR/credit bump curves
- Tenor bump double maps for IR/credit curves
- Flat/recovery bumped measure maps for recovery bumped credit curves
- Measure Maps generated for Custom Scenarios

ExerciseInfo

ExerciseInfo is a place-holder for the full set of exercise information. It contains the exercise date, the exercise factor, and the exercise type.

BondCouponMeasures

This class encapsulates the parsimonius but complete set of the cash-flow oriented coupon measures generated out of a full bond analytics run to a given work-out. These are:

- DV01
- PV Measures (Coupon PV, Index Coupon PV, PV)

BondWorkoutMeasures

BondWorkoutMeasures encapsulates the parsimonius yet complete set of measures generated out of a full bond analytics run to a given work-out. It contains the following:

- Credit Risky/Credit Riskless Clean/Dirty Coupon Measures
- Credit Risky/Credit Riskless Par/Principal PV
- Loss Measures such as expected Recovery, Loss on instantaneous default, and default exposure with/without recovery
- Unit Coupon measures such as Accrued 01, first coupon/index rate

BondRVMeasures

BondRVMeasures encapsulates the comprehensive set of RV measures calculated for the bond to the appropriate exercise:

- Workout Information.
- Price, Yield, and Yield01.
- Spread Measures: Asset Swap/Credit/G/I/OAS/PECS/TSY/Z.
- Basis Measures: Bond Basis, Credit Basis, Yield Basis.
- Duration Measures: Macaulay/Modified Duration, Convexity

BasketMeasures

BasketMeasures is the place holder for the analytical basket measures, optionally across scenarios. It contains the following scenario measure maps:

- Unadjusted Base Measures
- Flat delta/gamma bump measure maps for IR/credit/RR bump curves
- Component/tenor bump double maps for IR/credit/RR curves
- Flat/component recovery bumped measure maps for recovery bumped credit curves
- Custom scenario measure map

Credit Product: Serializer

Serializer interface are implemented in the package *org.drip.service.stream*. The interface defines methods for serializing out of and de-serializing into a byte stream, as well as the object serialization version.

There is just one interface in this package – Serializer.

Serializer

Serializer interface defines the core object serializer methods – serialization into and de-serialization out of byte arrays, as well as the object version.

Credit Analytics Library

Credit Analytics Library consists of the following 12 packages:

1. Reference Data Loaders: This package contains functionality that loads the bond and the CDS reference data, as well as closing marks for a few date ranges.

2. Analytics Configurator: This package contains functionality to configure various aspects of Credit Analytics.

3. Market, Quote, and Scenario Parameter Implementations: This contains the implementations of the Credit Product interfaces representing the quotes, the basket/component market parameters, and the scenario curve containers.

4. Market, Quote, and Scenario Parameter Creators: This contains the builder factories for the quotes, market parameters, and the scenario curves.

5. Rates Component Implementations: This contains the implementations of the Credit Product interfaces for Cash, Euro-dollar future, fixed/floating streams, interest rate swap instruments, and rates basket products.

6. Credit Product Implementations: This contains the implementations of the Credit Product interfaces for Bonds, CDS, basket CDS, and bond baskets.

7. FX Product Implementations: This contains the implementation of the Credit Product interface for FX products.

8. Product Creators: This contains the creators for the various rates, credit, and FX component and basket products.

9. Analytics Environment Manager: This provides functionality for loading products from their reference data and managing them, as well as creating/accessing live/closing curves.

10. Analytics Bridge: This provides the stub and proxy functionality for invoking Credit Analytics functionality in a remote server and extracting the results.

11. Analytics API: This provides a unified and comprehensive functional, static interface of all the main Credit Analytics functionality.

12. Functional Testers: This contains a fairly extensive set of unit and composite testers for the curve, products, serialization, and analytics functionality provided by the Credit Analytics suite, with a special focus on bonds.

Credit Analytics: Reference Data Loaders

Data loaders are implemented in the package *org.drip.feed.loaders*. This package contains functionality that loads the bond and the CDS reference data, as well as closing marks for a few date ranges.

Functionality in this package is implemented over 3 classes – BondRefData, CDXRefData, and CreditStaticAndMarks.

BondRefData

BondRefData contains functionality to load a variety of Bond Product reference data and closing marks. It exposes the following functionality:

- Load the bond valuation-based reference data, amortization schedule and EOS
- Build the bond instance entities from the valuation-based reference data
- Load the bond non-valuation-based reference data

BondRefData assumes the appropriate connections are available to load the data.

CDXRefData

CDXRefData contains the functionality to load the standard CDX reference data and definitions, and create compile time static classes for these definitions.

CreditStaticAndMarks

CreditStaticAndMarks contains functionality to load a variety of Credit and Rates Product reference data and closing marks. It exposes the following functionality:

- Load the bond reference data, static data, amortization schedule and EOS.
- Build the bond instance entities from the reference data.
- Load the bond, CDS, and Rates product Closing Marks.
- Load and build the Holiday Calendars.

CreditStaticAndMarks assumes the appropriate connections are available to load the data.

Credit Analytics: Analytics Configurator

Credit Analytics configurator is implemented in the package *org.drip.param.config*. This package contains functionality to configure various aspects of Credit Analytics.

Functionality in this package is implemented in a single class – ConfigLoader.

ConfigLoader

ConfigLoader implements the configuration functionality. It exposes the following:

- Parses the XML configuration file and extract the tag pairs information.
- Retrieve the logger.
- Load the holiday calendars and retrieve the location holidays.
- Connect to analytics server and the database server.

Depending on the configuration setting, ConfigLoader loads the above from either a file or the specified database.

Credit Analytics: Market Parameters, Quotes, and Scenario Parameter Implementations

Quotes and Market Parameters are implemented in the package org.drip.param.market. This contains the implementations of the Credit Product interfaces representing the quotes, the basket/component market parameters, and the scenario curve containers.

Functionality in this package is implemented over 8 classes – MultiSidedQuote, ComponentTickQuote, ComponentMultiMeasureQuote, RatesCurveScenarioContainer, CreditCurveScenarioContainer, ComponentMarketParamsSet, BasketMarketParamSet, and MarketParamsContainer.

MultiSidedQuote

MultiSidedQuote implements the Quote interface, which contains the stubs corresponding to a product quote. It contains the quote value and the quote time-snap for the different quote sides (bid/ask/mid).

ComponentTickQuote

ComponentTickQuote holds the tick related component parameters - it contains the product ID, the quote composite, the source, the counter party, and whether the quote can be treated as a mark.

ComponentMultiMeasureQuote

ComponentMultiMeasureQuote holds the different types of quotes for a given component. It contains a single market field/quote pair, but multiple alternate named quotes (to accommodate quotes on different measures for the component).

RatesCurveScenarioContainer

RatesCurveScenarioContainer implements the RatesScenarioCurve abstract class that exposes the interface the constructs scenario discount curves. The following curve construction scenarios are supported:

- Base, flat/tenor up/down by arbitrary bumps
- Tenor bumped discount curve set - keyed using the tenor
- NTP-based custom scenario curves

CreditCurveScenarioContainer

CreditCurveScenarioContainer is the place-holder for the bump parameters and the curves for the different credit curve scenarios. Contains the spread and the recovery bumps, and the credit curve scenario generator object that wraps the calibration instruments. It also contains the base credit curve, spread bumped up/down credit curves, recovery bumped up/down credit curves, and the tenor mapped up/down credit curves.

ComponentMarketParamSet

ComponentMarketParamSet provides implementation of the ComponentMarketParamsRef interface. It is the place-holder for the market parameters needed to value the component object – discount curve, forward curve, treasury curve, EDSF curve, credit curve, component quote, treasury quote map, and fixings map.

BasketMarketParamSet

BasketMarketParamSet provides an implementation of BasketMarketParamsRef for a specific scenario. It contains maps holding named discount curves, named credit curves, named component quote, and fixings object.

MarketParamsContainer

MarketParamsContainer extends MarketParams abstract class, and is the place-holder for the comprehensive suite of the market set of curves for the given date. It exports the following functionality:

- Add/remove/retrieve scenario discount curve.
- Add/remove/retrieve scenario zero curve.
- Add/remove/retrieve scenario credit curve.
- Add/remove/retrieve scenario recovery curve.
- Add/remove/retrieve scenario FXForward curve.
- Add/remove/retrieve scenario FXBasis curve.
- Add/remove/retrieve scenario fixings.
- Add/remove/retrieve Treasury/component quotes.
- Retrieve scenario CMP/BMP.
- Retrieve map of flat rates/credit/recovery CMP/BMP.
- Retrieve double map of tenor rates/credit/recovery CMP/BMP.
- Retrieve rates/credit scenario generator.

Credit Analytics: Market Parameters, Quotes, and Scenario Parameter Creators

Builders for quotes, market parameters, and scenario curves are implemented in the package *org.drip.param.creator*. This contains the builder factories for the quotes, market parameters, and the scenario curves.

Functionality in this package is implemented over 8 classes – QuoteBuilder, ComponentQuoteBuilder, ComponentTickQuoteBuilder, RatesScenarioCurveBuilder, CreditScenarioCurveBuilder, ComponentMarketParamsBuilder, BasketMarketParamsBuilder, MarketParamsBuilder.

QuoteBuilder

QuoteBuilder contains the quote builder object. It contains static functions that build two-sided quotes from inputs, as well as from a byte stream.

ComponentQuoteBuilder

ComponentQuoteBuilder contains the component quote builder object. It contains static functions that build component quotes from the quote inputs, as well as from byte streams.

ComponentTickQuoteBuilder

ComponentTickQuoteBuilder implements the component tick quote builder object. It contains static functions that build component quotes from the inputs, as well as from byte array.

RatesScenarioCurveBuilder

RatesScenarioCurveBuilder implements the the construction of the scenario discount curve using the input discount curve instruments.

CreditScenarioCurveBuilder

CreditScenarioCurveBuilder implements the construction, de-serialization, and building of the custom scenario based credit curves.

ComponentMarketParamsBuilder

ComponentMarketParamsBuilder implements the various ways of constructing, de-serializing, and building the Component Market Parameters.

BasketMarketParamsBuilder

BasketMarketParamsBuilder implements the various ways of constructing, de-serializing, and building the Basket Market Parameters.

MarketParamsBuilder

MarketParamsBuilder implements the functionality for constructing, de-serializing, and building the Market Universe Curves Container.

Credit Analytics: Rates Component Implementations

Rates components are implemented in the package *org.drip.product.rates*. This contains the implementations of the Credit Product interfaces for Cash, Euro-dollar future, fixed/floating streams, interest rate swap instruments, and rates basket products.

Functionality in this package is implemented over 6 classes – CashComponent, EDFComponent, FixedStream, FloatingStream, IRSComponent, and RatesBasket.

CashComponent

CashComponent contains the implementation of the Cash product and its contract/valuation details.

EDFComponent

EDFComponent contains the implementation of the Euro-dollar future contract/valuation (EDF).

FixedStream

FixedStream contains an implementation of the fixed leg cash flow stream product.

FloatingStream

FloatingStream contains an implementation of the Floating leg cash flow stream.

IRSComponent

IRSComponent contains the implementation of the Interest Rate Swap product contract/valuation details. It is made off one fixed stream and one floating stream.

RatesBasket

RatesBasket contains the implementation of the Basket of Rates Component legs. RatesBasket is made from zero/more fixed and floating streams.

Credit Analytics: Credit Product Implementations

Credit product definitions are implemented in the package *org.drip.product.credit*. This contains the implementations of the Credit Product interfaces for Bonds, CDS, basket default swaps, and bond baskets.

Functionality in this package is implemented over 4 classes – BondComponent, BondBasket, CDSComponent, and CDSBasket.

BondComponent

BondComponent is the base class that extends CreditComponent abstract class and implements the functionality behind bonds of all kinds. Bond static data is captured in a set of 11 container classes – BondTSYParams, BondCouponParams, BondNotionalParams, BondFloaterParams, BondCurrencyParams, BondIdentifierParams, BondIRValuationParams, CompCRValParams, BondCFTerminationEvent, BondFixedPeriodGenerationParams, and one EmbeddedOptionSchedule object instance each for the call and the put objects. Each of these parameter sets can be set separately.

BondBasket

BondBasket implements the bond basket product contract details. Contains the basket name, basket notional, component bonds, and their weights.

CDSComponent

CDSComponent implements the credit default swap product contract details. Contains effective date, maturity date, coupon, coupon day count, coupon frequency, contingent credit, currency, basket notional, credit valuation parameters, and optionally the outstanding notional schedule.

CDSBasket

CDSBasket implements the basket default swap product contract details. Contains effective date, maturity date, coupon, coupon day count, coupon frequency, basket components, basket notional, loss pay lag, and optionally the outstanding notional schedule and the flat basket recovery.

Credit Analytics: FX Component Implementations

FX components are implemented in the package *org.drip.product.fx*. This contains the implementations of the Credit Product interfaces for FX spot and forward contracts.

Functionality in this package is implemented over 2 classes – FXSpotContract and FXForwardContract.

FXForwardContract

FXForwardContract contains the FX forward product contract details - the effective date, the maturity date, the currency pair and the product code.

FXSpotContract

FXSpotContract contains the FX spot contract parameters - the spot date and the currency pair.

Credit Analytics: Product Creators

Product creators are implemented in the package *org.drip.product.creator*. This contains the creators for the various rates, credit, and FX component and basket products.

Functionality in this package is implemented over 12 classes – CashBuilder, EDFutureBuilder, RatesStreamBuilder, CDSBuilder, bond creator classes (BondRefDataBuilder, BondProductBuilder, BondBuilder), CDS basket creator classes (CDSBasketBuilder, CDXRefDataHolder), BondBasketBuilder, and FX product builder classes (FXSpotBuilder and FXForwardBuilder). Of these CDXRefDataHolder is generated from the CDX reference/static information.

BondBasketBuilder

BondBasketBuilder contains the suite of helper functions for creating the bond Basket Product from different kinds of inputs and byte streams.

BondBuilder

BondBuilder contains the suite of helper functions for creating simple fixed/floater bonds, user defined bonds, optionally with custom cash flows and embedded option schedules (European or American). It also constructs bonds by de-serializing the byte stream.

BondProductBuilder

BondProductBuilder holds the static parameters of the bond product needed for the full bond valuation. It contains:

- Bond identifier parameters (ISIN, CUSIP)
- Issuer level parameters (Ticker, SPN or the credit curve string)
- Coupon parameters (coupon rate, coupon frequency, coupon type, day count)
- Maturity parameters (maturity date, maturity type, final maturity, redemption value)
- Date parameters (announce, first settle, first coupon, interest accrual start, and issue dates)
- Embedded option parameters (callable, putable, has been exercised)
- Currency parameters (trade, coupon, and redemption currencies)
- Floater parameters (floater flag, floating coupon convention, current coupon, rate index, spread)
- Whether the bond is perpetual or has defaulted

BondRefDataBuilder

BondRefDataBuilder holds the entire set of static parameters for the bond product. In particular, it contains

- Bond identifier parameters (ISIN, CUSIP, BBG ID, name short name)
- Issuer level parameters (Ticker, category, industry, issue type, issuer country, issuer country code, collateral type, description, security type, unique Bloomberg ID, long company name, issuer name, SPN or the credit curve string)
- Issue parameters (issue amount, issue price, outstanding amount, minimum piece, minimum increment, par amount, lead manager, exchange code, country of incorporation, country of guarantor, country of domicile, industry sector, industry group, industry sub-group, senior/sub)
- Coupon parameters (coupon rate, coupon frequency, coupon type, day count)
- Maturity parameters (maturity date, maturity type, final maturity, redemption value)

- Date parameters (announce, first settle, first coupon, interest accrual start, next coupon, previous coupon, penultimate coupon, and issue dates)
- Embedded option parameters (callable, putable, has been exercised)
- Currency parameters (trade, coupon, and redemption currencies)
- Floater parameters (floater flag, floating coupon convention, current coupon, rate index, spread)
- Trade status
- Ratings (S & P, Moody, and Fitch),
- Whether the bond is private placement, is registered, is a bearer bond, is reverse convertible, is a structured note, can be unit traded, is perpetual or has defaulted.

CashBuilder

CashBuilder contains the suite of helper functions for creating the Cash product from the parameters/codes/byte array streams.

CDSBasketBuilder

CDSBasketBuilder contains the suite of helper functions for creating the CDS Basket Product from different kinds of inputs and byte streams.

CDSBuilder

CDSBuilder contains the suite of helper functions for creating the CreditDefaultSwap product from the parameters/byte array streams. It also creates the standard EU, NA, ASIA contracts, CDS with amortization schedules, and CDS from product codes/tenors.

CDXRefDataHolder

CDXRefDataHolder contains all the generated standard CDX Products, returned as instances of CreditProduct's BasketProduct interface. Since this is a generated file, please do not delete this.

EDFutureBuilder

EDFutureBuilder contains the suite of helper functions for creating the EDFuture product from the parameters/codes/byte array streams.

FXForwardBuilder

FXForwardBuilder contains the suite of helper functions for creating the FXForwardBuilder product from the parameters/byte array streams.

FXSpotBuilder

FXSpotBuilder contains the suite of helper functions for creating the FXSpot from the corresponding parameters/byte array streams.

RatesStreamBuilder

RatesStreamBuilder contains the suite of helper functions for creating the Stream-based Rates Products from different kinds of inputs. In particular, it demonstrates the following:

- Construction of the custom/standard fixed/floating streams from parameters.
- Construction of the custom/standard IRS from parameters.
- Construction of the fixed/floating streams and IRS from byte arrays.

Credit Analytics: Analytics Environment Manager

Analytics Environment Manager component are implemented in the package *org.drip.service.env*. This contains the creators for the various rates, credit, and FX component and basket products.

Functionality in this package is implemented over 7 classes – BondManager, CDSManager, EnvManager, EODCurves, RatesManager, StandardCDXManager, and StaticBACurves.

BondManager

BondManager implements a container that holds the EOD and bond static information on a per issuer basis. It exposes the following functionality:

- Retrieve the available tickers, and all the ISIN's per ticker.
- Load the full set of bond reference data, embedded option schedules, and amortization schedules.
- Load the full set of bond marks.
- Calculate the bond RV/Value measures for a ticker/full bond set, given the EOD and the appropriate curves and market measures.
- Save the computed measures for a given EOD.
- (Optionally) Generate a Bond Creator File.

CDSManager

CDSManager is the container that retrieves the EOD and CDS/credit curve information on a per-issuer basis and populates the MPC.

EnvManager

EnvManager sets the environment and connection parameters, and populates the market parameters (quotes, curves, and fixings) for a given EOD.

EODCurves

EODCurves that creates the closing curves from the closing marks available in the DB for a given EOD and populates them onto the MPC. It builds the following:

- Discount Curve (from cash/future/swap - typical sequence), EDSF Curve, and TSY Curve
- Credit Curve from CDS quotes
- On-the-run TSY yield quotes

RatesManager

RatesManager manages the creation/loading of the rates curves of different kinds for a given EOD.

StaticBACurves

StaticBACurves that creates the closing curves from custom/user defined marks for a given EOD and populates them onto the MPC. It builds the following:

- Discount Curve (from cash/future/swap - typical sequence), EDSF Curve, and TSY Curve

- Credit Curve from CDS quotes
- On-the-run TSY yield quotes

StandardCDXManager

StandardCDXManager implements the creation and the static details of the all the NA, EU, SovX, EMEA, and ASIA standardized CDS indices. It exposes the following functionality:

- Retrieve the full set of pre-set/pre-loaded CDX names/descriptions.
- Retrieve all the CDX's given an index name.
- Get the index, index series, and the effective/maturity dates for a given CDX.
- Get all the on-the-runs for an index, date, and tenor.
- Retrieve the full basket product corresponding to NA/EU/ASIA IG/HY/EM and other available standard CDX.
- Build a custom CDX product.

Credit Analytics: Analytics Bridge

Analytics Bridge is implemented in the package *org.drip.service.bridge*. This provides the stub and proxy functionality for invoking Credit Analytics functionality in a remote server and extracting the results.

Functionality in this package is implemented over 4 classes – CreditAnalyticsRequest, CreditAnalyticsResponse, CreditAnalyticsStub, and CreditAnalyticsProxy.

CreditAnalyticsRequest

CreditAnalyticsRequest contains the requests for the Credit Analytics server from the client. It contains the following parameters:

- The GUID and the time-stamp of the request.
- The component that is being valued.
- The valuation, the pricer, and the quoting parameters.
- The market parameters assembled in the ComponentMarketParams.

Typical usage is: Client fills in the entities in the request, serializes them, and sends them to the server, and receives a serialized response back from the server.

CreditAnalyticsResponse

CreditAnalyticsResponse contains the response from the Credit Analytics server to the client. It contains the following parameters:

- The GUID and of the request.
- The type and time-stamp of the response.

- The string version of the response body.

CreditAnalyticsProxy

CreditAnalyticsProxy captures the requests for the Credit Analytics server from the client, formats them, and sends them to the Credit Analytics Stub.

CreditAnalyticsStub

CreditAnalyticsStub serves as a sample server that hosts the Credit Analytics functionality. It receives requests from the analytics client as a serialized message, and invokes the CreditAnalytics functionality, and sends the client the serialized results.

Credit Analytics: Analytics API

Analytics API is implemented in the package *org.drip.service.api*. This provides a unified and comprehensive functional, static interface of all the main Credit Analytics functionality.

Functionality in this package is implemented over a single class – CreditAnalytics.

CreditAnalytics

CreditAnalytics exposes all the CreditAnalytics API to clients – this class is the main functional interface. The functions exposed are too numerous to list, and can be roughly grouped into the following:

- Product Creation
- Curve Construction from Market Instruments
- Product Reference Data Examination
- Product Valuation from the Market Parameters
- Product Measure Extraction
- Product RV Measure Computation
- General Finance Math calculation (day count, date adjust etc.)
- Closing points extraction

Credit Analytics: Functional Testers

Credit Analytics functional testers are available in the package *org.drip.tester.functional*. This contains a fairly extensive set of unit and composite testers for the curve, products, serialization, and analytics functionality provided by the Credit Analytics suite, with a special focus on bonds.

Functionality in this package is implemented over 4 classes – BondTestSuite, CreditAnalyticsTestSuite, ProductTestSuite, and SerializerTestSuite.

BondTestSuite

BondTestSuite tests more-or-less the full suite of bond functionality exposed in CreditAnalytics API.

CreditAnalyticsTestSuite

CreditAnalyticsTestSuite tests more-or-less the full suite of functionality exposed in CreditAnalytics API across all products, curves, quotes, outputs, and parameters, and their variants.

ProductTestSuite

ProductTestSuite tests more-or-less the full suite of the product valuation functionality exposed in CreditAnalytics API. The following variants are tested.

- Full suite of products - rates, credit and FX, both components and baskets.

- Base flat/tenor bumped scenario tests.

SerializerTestSuite

SerializerTestSuite tests the serialization functionality across all products, curves, quotes, outputs, and parameters, and their variants.

Curve Builder

Curve Builder Library consists of the following 6 packages:

1. Latent State Representation: The latent state representation package implements the latent state, the quantification metric/manifest measure, its labels, the merge stretch and its manager.

2. Latent Curve State: The latent curve state package provides implementations of latent state representations of discount curve, forward curve, zero curve, credit curve, FX Basis curve, and FX forward curve.

3. Latent State Estimator: The latent state estimator package provides functionality to estimate the latent state, local/global state construction controls, constraint representation, and linear/non-linear calibrator routines.

4. Latent State Creator: The latent curve state package provides implementations of the constructor factories that create discount curve, forward curve, zero curve, credit curve, FX Basis curve, and FX forward curve.

5. Curve Analytics Definitions: The analytics definition package provides definitions of the generic curve, discount curve, forward curve, zero curve, credit curve, FX Basis curve, and FX forward curve, turns list, and their construction inputs.

6. Rates Analytics: The rates analytics package provides definitions of the discount curve, the forward curve, the zero curve, the discount factor and the forward rate estimators, the turns list, and their construction inputs.

Curve Builder: Latent State Representation

Curve Builder Latent State Representation functions are available in the package *org.drip.state.representation*. The latent state representation package implements the latent state, the quantification metric/manifest measure, its labels, the merge stretch and its manager.

Functionality in this package is implemented over 5 classes – LatentStateLabel, LatentStateMergeSubStretch, MergeSubStretchManager, LatentStateMetricMeasure, and LatentState.

LatentStateLabel

LatentStateLabel is the interface that contains the labels inside the sub-stretch of the alternate state. The functionality its derivations implement provide fully qualified label names and their matches.

LatentStateMergeSubStretch

LatentStateMergeSubStretch implements merged stretch that is common to multiple latent states. It is identified by the start/end date predictor ordinates, and the Latent State Label. Its methods provide the following functionality:

- Identify if the specified predictor ordinate belongs to the sub stretch
- Shift that sub stretch start/end
- Identify if the this overlaps the supplied sub stretch, and coalesce them if possible
- Retrieve the label, start, and end

MergeSubStretchManager

MergeSubStretchManager manages the different discount-forward merge stretches. It provides functionality to create, expand, or contract the merge stretches.

LatentStateMetricMeasure

LatentStateMetricMeasure holds the latent state that is estimated, its quantification metric, and the corresponding product manifest measure, and its value that it is estimated off of during the calibration run.

LatentState

LatentState exposes the functionality to manipulate the hidden Variable's Latent State. Specifically it exports functions to:

- Retrieve the Array of the LatentStateMetricMeasure
- Produce node shifted, parallel shifted, and custom manifest-measure tweaked variants of the Latent State
- Produce parallel shifted and custom quantification metric tweaked variants of the Latent State

Curve Builder: Latent Curve State

Curve Builder Latent Curve State functions are available in the package *org.drip.state.curve*. The latent curve state package provides implementations of latent state representations of discount curve, forward curve, zero curve, credit curve, FX Basis curve, and FX forward curve.

Functionality in this package is implemented over 9 classes – DiscountFactorDiscountCurve, NonlinearDiscountFactorDiscountCurve, ZeroRateDiscountCurve, DerivedZeroRate, FlatForwardDiscountCurve, BasisSplineForwardRate, ForwardHazardCreditCurve, DerivedFXForward, and DerivedFXBasis.

DiscountFactorDiscountCurve

DiscountFactorDiscountCurve manages the Discounting Latent State, using the Discount Factor as the State Response Representation. It exports the following functionality:

- Compute the discount factor, forward rate, or the zero rate from the Discount Factor Latent State
- Create a ForwardRateEstimator instance for the given Index
- Retrieve Array of the Calibration Components and their LatentStateMetricMeasure's
- Retrieve the Curve Construction Input Set
- Compute the Jacobian of the Discount Factor Latent State to the input Quote
- Synthesize scenario Latent State by parallel shifting/custom tweaking the quantification metric
- Synthesize scenario Latent State by parallel/custom shifting/custom tweaking the manifest measure
- Serialize into and de-serialize out of byte array

NonlinearDiscountFactorDiscountCurve

NonlinearDiscountFactorDiscountCurve manages the Discounting Latent State, using the Forward Rate as the State Response Representation. It exports the following functionality:

- Boot Methods - Set/Bump Specific Node Quantification Metric, or Set Flat Value
- Boot Calibration - Initialize Run, Compute Calibration Metric
- Compute the discount factor, forward rate, or the zero rate from the Forward Rate Latent State
- Create a ForwardRateEstimator instance for the given Index
- Retrieve Array of the Calibration Components and their LatentStateMetricMeasure's
- Retrieve the Curve Construction Input Set
- Compute the Jacobian of the Discount Factor Latent State to the input Quote
- Synthesize scenario Latent State by parallel shifting/custom tweaking the quantification metric
- Synthesize scenario Latent State by parallel/custom shifting/custom tweaking the manifest measure
- Serialize into and de-serialize out of byte array

ZeroDiscountCurve

ZeroRateDiscountCurve manages the Discounting Latent State, using the Zero Rate as the State Response Representation. It exports the following functionality:

- Compute the discount factor, forward rate, or the zero rate from the Zero Rate Latent State
- Create a ForwardRateEstimator instance for the given Index
- Retrieve Array of the Calibration Components and their LatentStateMetricMeasure's

- Retrieve the Curve Construction Input Set
- Compute the Jacobian of the Discount Factor Latent State to the input Quote
- Synthesize scenario Latent State by parallel shifting/custom tweaking the quantification metric
- Synthesize scenario Latent State by parallel/custom shifting/custom tweaking the manifest measure
- Serialize into and de-serialize out of byte array

DerivedZeroRate

DerivedZeroRate implements the delegated ZeroCurve functionality. Beyond discount factor/zero rate computation at specific cash pay nodes, all other functions are delegated to the embedded discount curve.

FlatForwardDiscountCurve

FlatForwardDiscountCurve manages the Discounting Latent State, using the Forward Rate as the State Response Representation. It exports the following functionality:

- Boot Methods - Set/Bump Specific Node Quantification Metric, or Set Flat Value
- Boot Calibration - Initialize Run, Compute Calibration Metric
- Compute the discount factor, forward rate, or the zero rate from the Forward Rate Latent State
- Create a ForwardRateEstimator instance for the given Index
- Retrieve Array of the Calibration Components and their LatentStateMetricMeasure's
- Retrieve the Curve Construction Input Set
- Compute the Jacobian of the Discount Factor Latent State to the input Quote
- Synthesize scenario Latent State by parallel shifting/custom tweaking the quantification metric

- Synthesize scenario Latent State by parallel/custom shifting/custom tweaking the manifest measure
- Serialize into and de-serialize out of byte array

BasisSplineForwardRate

BasisSplineForwardRate manages the Forward Latent State, using the Forward Rate as the State Response Representation. It exports the following functionality:

- Calculate implied forward rate / implied forward rate Jacobian
- Serialize into and de-serialize out of byte arrays

ForwardHazardCreditCurve

ForwardHazardCreditCurve manages the Survival Latent State, using the Hazard Rate as the State Response Representation. It exports the following functionality:

- Boot Methods - Set/Bump Specific Node Quantification Metric, or Set Flat Value
- Boot Calibration - Initialize Run, Compute Calibration Metric
- Compute the survival probability, recovery rate, or the hazard rate from the Hazard Rate Latent State
- Retrieve Array of the Calibration Components and their LatentStateMetricMeasure's
- Retrieve the Curve Construction Input Set
- Synthesize scenario Latent State by parallel shifting/custom tweaking the quantification metric
- Synthesize scenario Latent State by parallel/custom shifting/custom tweaking the manifest measure
- Serialize into and de-serialize out of byte array

DerivedFXForward

DerivedFXForward manages the constant forward based FX Forward Curve holder object. It exports the following functionality:

- Extract currency, currency pair, spot epoch and spot FX
- Compute Zero/boot-strap Basis, as well as boot-strap basis DC
- Compute the spot implied rate/implied rate nodes
- Retrieve Array of the Calibration Components and their LatentStateMetricMeasure's
- Retrieve the Curve Construction Input Set
- Synthesize scenario Latent State by parallel shifting/custom tweaking the quantification metric
- Synthesize scenario Latent State by parallel/custom shifting/custom tweaking the manifest measure
- Serialize into and de-serialize out of byte array

DerivedFXBasis

DerivedFXBasis manages the constant forward basis based FX Basis Curve holder object. It exports the following functionality:

- Extract currency, currency pair, spot epoch, spot FX, and whether the basis is boot-strapped
- Compute the FX Forward Array
- Retrieve Array of the Calibration Components and their LatentStateMetricMeasure's
- Retrieve the Curve Construction Input Set
- Synthesize scenario Latent State by parallel shifting/custom tweaking the quantification metric
- Synthesize scenario Latent State by parallel/custom shifting/custom tweaking the manifest measure
- Serialize into and de-serialize out of byte array

Curve Builder: Latent State Estimator

Curve Builder Latent State Estimator functions are available in the package *org.drip.state.estimator*. The latent state estimator package provides functionality to estimate the latent state, local/global state construction controls, constraint representation, and linear/non-linear calibrator routines.

Functionality in this package is implemented over 11 classes – StretchRepresentationSpec, PredictorResponseWeightConstraint, SmoothingCurveStretchParams, GlobalCurveControlParams, LocalCurveControlParams, CurveStretch, RatesSegmentSequenceBuilder, LinearCurveCalibrator, NonlinearCurveCalibrator, RatesCurveScenarioGenerator, and CreditCurveScenarioGenerator.

StretchRepresentationSpec

StretchRepresentationSpec carries the calibration instruments and the corresponding calibration parameter set in LSMM instances. Together, these inputs are used for constructing an entire latent state stretch. StretchRepresentationSpec exports the following functionality:

- Alternate ways of constructing custom Stretch representations
- Retrieve indexed instrument/LSMM
- Retrieve the full set calibratable instrument/LSMM

PredictorResponseWeightConstraint

PredictorResponseWeightConstraint holds the Linearized Constraints (and, optionally, their quote sensitivities) necessary needed for the Linear Calibration. Linearized Constraints are expressed as $C_j = \sum_i W_i y\left(x_{ij}\right)$ where x_{ij} is the predictor ordinate at node i, y is the response, W_i is the weight applied for the Response i, and C_j is the value of constraint j. The function can either be univariate function, or weighted spline basis set. To this end, it implements the following functionality:

- Update/Retrieve Predictor/Response Weights and their Quote Sensitivities
- Update/Retrieve Predictor/Response Constraint Values and their Quote Sensitivities
- Display the contents of PredictorResponseWeightConstraint

SmoothingCurveStretchParams

SmoothingCurveStretchParams contains the Parameters needed to hold the Stretch. It provides functionality to:

- The Stretch Best fit Response and the corresponding Quote Sensitivity
- The Calibration Detail and the Curve Smoothening Quantification Metric
- The Segment Builder Parameters

GlobalCurveControlParams

GlobalControlCurveParams enhances the SmoothingCurveStretchParams to produce globally customized curve smoothing. Currently, GlobalControlCurveParams uses custom boundary setting and spline details to implement the global smoothing pass.

LocalCurveControlParams

LocalControlCurveParams enhances the SmoothingCurveStretchParams to produce locally customized curve smoothing. Flags implemented by LocalControlCurveParams control the following:

- The C1 generator scheme to be used
- Whether to eliminate spurious extrema
- Whether or not to apply monotone filtering

CurveStretch

CurveStretch expands the regular Multi-Segment Stretch to aid the calibration of Boot-strapped Instruments. In particular, CurveStretch implements the following functions that are used at different stages of curve construction sequence:

- Mark the Range of the "built" Segments
- Clear the built range mark to signal the start of a fresh calibration run
- Indicate if the specified Predictor Ordinate is inside the "Built" Range
- Retrieve the MergeSubStretchManager

RatesSegmentSequenceBuilder

RatesSegmentSequenceBuilder holds the logic behind building the bootstrap segments contained in the given Stretch. It extends the SegmentSequenceBuilder interface by implementing/customizing the calibration of the starting as well as the subsequent segments.

LinearCurveCalibrator

LinearCurveCalibrator creates the discount curve span from the instrument cash flows. The span construction may be customized using specific settings provided in GlobalControlCurveParams.

NonlinearCurveCalibrator

NonlinearCurveCalibrator calibrates the discount and credit/hazard curves from the components and their quotes. NonlinearCurveCalibrator employs a set of techniques for achieving this calibration.

- It bootstraps the nodes in sequence to calibrate the curve
- In conjunction with splining estimation techniques, it may also be used to perform dual sweep calibration. The inner sweep achieves the calibration of the segment spline parameters, while the outer sweep calibrates iteratively for the targeted boundary conditions
- It may also be used to custom calibrate a single Interest Rate/Hazard Rate Node from the corresponding Component
- CurveCalibrator bootstraps/cooks both discount curves and credit curves

RatesCurveScenarioGenerator

RatesCurveScenarioGenerator uses the interest rate calibration instruments along with the component calibrator to produce scenario interest rate curves.
RatesCurveScenarioGenerator typically first constructs the actual curve calibrator instance to localize the intelligence around curve construction. It then uses this curve calibrator instance to build individual curves or the sequence of node bumped scenario curves. The curves in the set may be an array, or tenor-keyed.

CreditCurveScenarioGenerator

CreditCurveScenarioGenerator uses the hazard rate calibration instruments along with the component calibrator to produce scenario hazard rate curves.
CreditCurveScenarioGenerator typically first constructs the actual curve calibrator instance to localize the intelligence around curve construction. It then uses this curve calibrator instance to build individual curves or the sequence of node bumped scenario curves. The curves in the set may be an array, or tenor-keyed.

Curve Builder: Latent State Creator

Curve Builder Latent State Creator functions are available in the package *org.drip.state.creator*. The latent curve state package provides implementations of the constructor factories that create discount curve, forward curve, zero curve, credit curve, FX Basis curve, and FX forward curve.

Functionality in this package is implemented over 5 classes – DiscountCurveBuilder, ZeroCurveBuilder, CreditCurveBuilder, FXForwardCurveBuilder, and FXBasisCurveBuilder.

DiscountCurveBuilder

This class contains the builder functions that construct the discount curve (comprising both the rates and the discount factors) instance. It contains static functions that build different types of discount curve from 3 major types of inputs:

- From a variety of ordered DF-sensitive calibration instruments and their quotes
- From an array of ordered discount factors
- From a serialized byte stream of the discount curve instance

ZeroCurveBuilder

This class contains the builder functions that construct the zero curve instance. It contains static functions that build different types of zero curve from 2 major types of inputs:

- From a source discount curve, a set of coupon periods, and the Zero Bump
- From a serialized byte stream of the Zero curve instance

CreditCurveBuilder

This class contains the builder functions that construct the credit curve (comprising both survival and recovery) instance. It contains static functions that build different types of credit curve from 3 major types of inputs:

- From a variety of ordered credit-sensitive calibration instruments and their quotes
- From an array of ordered survival probabilities
- From a serialized byte stream of the credit curve instance

FXForwardCurveBuilder

This class contains the baseline FX Forward curve builder object. It contains static functions that build FX Forward curves from the 3 major inputs:

- An ordered array of Forward FX
- An ordered array of Forward Basis Points
- A byte Stream representing the serialized instance of the FXForwardCurve

FXBasisCurveBuilder

This class contains the baseline FX Basis curve builder object. It contains static functions that build FX Basis curves from the 3 major inputs:

- An ordered array of Forward FX
- An ordered array of Forward Basis Points
- A byte Stream representing the serialized instance of the FXBasisCurve

Curve Builder: Analytics Definition

Curve Builder Analytics Definition functions are available in the package *org.drip.analytics.definition*. The analytics definition package provides definitions of the generic curve, discount curve, forward curve, zero curve, credit curve, FX Basis curve, and FX forward curve, turns list, and their construction inputs.

Functionality in this package is implemented over 10 classes – CurveConstructionInputSet, CurveSpanConstructionInput, ShapePreservingCCIS, BootCurveConstructionInput, Curve, CreditCurve, ExplicitBootCurve, ExplicitBootCreditCurve, FXForwardCurve, and FXBasisCurve.

CurveConstructionInputSet

CurveConstructionInputSet interface contains the Parameters needed for the Curve Calibration/Estimation. It's methods expose access to the following:

- Calibration Valuation Parameters
- Calibration Quoting Parameters
- Array of Calibration Instruments
- Map of Calibration Quotes
- Map of Calibration Measures
- Double Map of the Date/Index Fixings

CurveSpanConstructionInput

CurveSpanConstructionInput contains the Parameters needed for the Curve Calibration/Estimation. It contains the following:

- Calibration Valuation Parameters
- Calibration Quoting Parameters
- Calibration Market Parameters
- Calibration Pricing Parameters
- Array of Calibration Stretch Representation
- Map of Calibration Quotes
- Map of Calibration Measures
- Double Map of the Date/Index Fixings
- Additional functions provide for retrieval of the above and specific instrument quotes. Derived Classes implement Targeted Curve Calibrators.

ShapePreservingCCIS

ShapePreservingCCIS extends the CurveSpanConstructionInput Instance. Additionally, it exposes the Shape Preserving Linear Curve Calibrator.

BootCurveConstructionInput

BootCurveConstructionInput contains the Parameters needed for the Curve Calibration/Estimation. It contains the following:

- Calibration Valuation Parameters
- Calibration Quoting Parameters
- Array of Calibration Instruments
- Map of Calibration Quotes
- Map of Calibration Measures
- Double Map of the Date/Index Fixings

Curve

Curve extends the Latent State to abstract the functionality required among all financial curve. It exposes the following functionality:

- Set the Epoch and the Identifiers
- Set up/retrieve the Calibration Inputs
- Retrieve the Latent State Metric Measures

CreditCurve

CreditCurve is the stub for the survival curve functionality. It extends the Curve object by exposing the following functions:

- Set of curve and market identifiers
- Recovery to a specific date/tenor, and effective recovery between a date interval
- Hazard Rate to a specific date/tenor, and effective hazard rate between a date interval
- Survival to a specific date/tenor, and effective survival between a date interval
- Set/unset date of specific default
- Generate scenario curves from the base credit curve (flat/parallel/custom)
- Set/unset the Curve Construction Inputs, Latent State, and the Manifest Metrics
- Serialization/De-serialization to and from Byte Arrays

ExplicitBootCurve

In ExplicitBootCurve, the segment boundaries explicitly line up with the instrument maturity boundaries. This feature is exploited in building a boot-strappable curve. Functionality is provides set the Latent State at the Explicit Node, adjust the Latent State at the given Node, or set a common Flat Value across all Nodes.

ExplicitBootCreditCurve

ExplicitBootCreditCurve exposes the functionality associated with the bootstrapped Credit Curve.

FXForwardCurve

FXForwardCurve implements the curve representing the FXForward nodes. It extends the Curve class, and exposes the following functionality:

- Retrieve the spot parameters (FX Spot, Spot Date, and the currency pair)
- Calculate the Zero set of FX Basis/Zero Rate nodes corresponding to each basis node
- Bootstrap basis points/discount curves corresponding to the FXForward node set
- Imply the zero rate to a given date from the FXForward curve

FXBasisCurve

FXBasisCurve implements the curve representing the FXBasis nodes. It extends the Curve class, and exposes the following functionality:

- Retrieve the spot parameters (FX Spot, Spot Date, and the currency pair)
- Indicate if the basis has been bootstrapped
- Calculate the Complete set of FX Forward corresponding to each basis node

Curve Builder: Rates Analytics

Curve Builder Rates Analytics functions are available in the package *org.drip.rates.analytics*. The rates analytics package provides definitions of the discount curve, the forward curve, the zero curve, the discount factor and the forward rate estimators, the turns list, and their construction inputs.

Functionality in this package is implemented over 11 classes – DiscountFactorEstimator, ForwardRateEstimator, Turn, TurnListDiscountFactor, RatesLSMM, SmoothingCCIS, DiscountForwardEstimator, ForwardCurve, DiscountCurve, ExplicitBootDiscountCurve, and ZeroCurve.

DiscountFactorEstimator

DiscountFactorEstimator is the interface that exposes the calculation of the Discount Factor for a specific Sovereign/Jurisdiction Span. It exposes the following functionality:

- Curve Epoch Date
- Discount Factor Target/Effective Variants - to Specified Julian Dates and/or Tenors
- Forward Rate Target/Effective Variants - to Specified Julian Dates and/or Tenors
- Zero Rate Target/Effective Variants - to Specified Julian Dates and/or Tenors
- LIBOR Rate and LIBOR01 Target/Effective Variants - to Specified Julian Dates and/or Tenors
- Curve Implied Arbitrary Measure Estimates

ForwardRateEstimator

ForwardRateEstimator is the interface that exposes the calculation of the Forward Rate for a specific Index. It exposes methods to compute forward rates to a given date/tenor, extract the forward rate index and the Tenor.

Turn

Turn implements rate spread at discrete time spans. It contains the turn amount and the start/end turn spans.

TurnListDiscountFactor

TurnListDiscountFactor implements the discounting based off of the turns list. Its functions add a turn instance to the current set, and concurrently apply the discount factor inside the range to each relevant turn.

RatesLSMM

RatesLSMM contains the Rates specific Latent State MM for the Rates Curve. Current it holds the turn list discount factor.

SmoothingCCIS

SmoothingCCIS enhances the Shape Preserving CCIS for smoothing customizations. It exposes the shape preserving discount curve and the smoothing curve stretch parameters.

DiscountForwardEstimator

DiscountForwardEstimator exposes the "native" forward curve associated with the specified discount curve. It exposes functionality to extract forward rate index/tenor, as well as to compute the forward rate implied off of the discount curve.

ForwardCurve

ForwardCurve is the stub for the forward curve functionality. It extends the Curve object by exposing the following functions:

- The name/epoch of the forward rate instance
- The index/currency/tenor associated with the forward rate instance
- Forward Rate to a specific date/tenor
- Generate scenario-tweaked Latent State from the base forward curve corresponding to mode adjusted (flat/parallel/custom) manifest measure/quantification metric.
- Retrieve array of latent state manifest measure, instrument quantification metric, and the array of calibration components.
- Set/retrieve curve construction input instrument sets.

DiscountCurve

DiscountCurve is the stub for the discount curve functionality. It extends the both the Curve and the DiscountFactorEstimator instances by implementing their functions, and exposing the following:

- Forward Rate to a specific date/tenor, and effective rate between a date interval
- Discount Factor to a specific date/tenor, and effective discount factor between a date interval
- Zero Rate to a specific date/tenor

- Value Jacobian for Forward rate, discount factor, and zero rate
- Cross Jacobian between each of Forward rate, discount factor, and zero rate
- Quote Jacobian to Forward rate, discount factor, and zero rate
- QM (DF/Zero/Forward) to Quote Jacobian
- Latent State Quantification Metric, and the quantification metric transformations
- Implied/embedded ForwardRateEstimator
- Turns - set/unset/adjust

ExplicitBootDiscountCurve

ExplicitBootDiscountCurve exposes the functionality associated with the bootstrapped Discount Curve.

- Generate a curve shifted using targeted basis at specific nodes
- Generate scenario tweaked Latent State from the base forward curve corresponding to mode adjusted (flat/parallel/custom) manifest measure/quantification metric
- Retrieve array of latent state manifest measure, instrument quantification metric, and the array of calibration components
- Set/retrieve curve construction input instrument sets

ZeroCurve

ZeroCurve exposes the node set containing the zero curve node points. In addition to the discount curve functionality that it automatically provides by extension, it provides the functionality to calculate the zero rate.

Regression Suite Library

Regression Suite Library consists of the following 5 packages:

1. Core Regression Library: This contains the full set of Regression Suite's core framework and the set of extensible interfaces.

2. Curve Regression Suite: The Curve Regression Package demonstrates the core curve regression functionality – regression of discount curve, credit curve, FX forward/basis curve, and zero curves.

3. Curve Jacobian Regression Suite: The Product Curve Jacobian Regression package carries out regression across the core suite of products Jacobian to the curve - Cash, EDF, and Fix-float IRS. It also implements the Curve Jacobian Regression Engine.

4. Fixed Point Finder Regression Suite: This contains the suite for regression testing of the non-linear univariate fixed-point finder.

5. Basis Spline Regression Suite: This package contains the random input regression runs on the spline and stretch instances. Runs regress on C1Hermite, local control smoothing, single segment Lagrangians, multi-segment sequences using a variety of spline/stretch basis functions and controls.

Regression Suite: Core

The core functionality of the regression suite library is implemented in the package *org.drip.regression.core*. This contains the full set of Regression Suite's core framework and the set of extensible interfaces.

Functionality in this package is implemented over 7 classes – RegressionEngine, RegressionRunDetail, RegressionRunOutput, RegressorSet, UnitRegressionExecutor, UnitRegressionStat, and UnitRegressor.

RegressionEngine

RegressionEngine provides the control and frame-work functionality for the General Purpose Regression Suite. It invokes the following steps as part of the execution:

- Initialize the regression environment. This step sets up the regression sets, and adds individual regressors to the set.
- Invoke the regressors in each set one by one.
- Collect the results and details of the regression runs.
- Compile the regression statistics.
- Optionally display the regression statistics.

RegressionRunDetail

RegressionRunDetail contains named field level detailed output of the regression activity.

RegressionRunOutput

RegressionRunOutput contains the output of a single regression activity. It holds the following:

- The execution time
- The Success/failure status of the run
- The regression scenario that was executed
- The Completion time for the regression module
- The Regression Run Detail for the regression run

RegressorSet

RegressorSet interface provides the Regression set stubs. It contains a set of regressors and is associated with a unique name. It provides the functionality to set up the contained regressors.

UnitRegressionExecutor

UnitRegressionExecutor implements the UnitRegressor, and splits the regression execution into pre-, execute, and post-regression. It provides default implementations for pre-regression and post-regression. Most typical regressors only need to over-ride the execRegression method.

UnitRegressionStat

UnitRegressionStat creates the statistical details for the Unit Regressor. It holds the following:

- Execution Initialization Delay

- Execution time mean, variance, maximum, and minimum
- The full list of individual execution times

UnitRegressor

UnitRegressor provides the stub functionality for the Individual Regressors. Its derived classes implement the actual regression run. Individual regressors are named.

Curve Regression Suite

The core functionality of the curve regression library is implemented in the package org.drip.regression.curve. The Curve Regression Package demonstrates the core curve regression functionality – regression of discount curve, credit curve, FX forward/basis curve, and zero curves.

Functionality in this package is implemented over 5 classes – DiscountCurveRegressor, ZeroCurveRegressor, CreditCurveRegressor, FXCurveRegressor, and CreditAnalyticsRegressionEngine.

DiscountCurveRegressor

DiscountCurveRegressor implements the regression set analysis for the Discount Curve. DiscountCurveRegressor regresses 11 scenarios:

- #1: Create the discount curve from a set 30 instruments (cash/future/swap).
- #2: Create the discount curve from a flat discount rate.
- #3: Create the discount curve from a set of discount factors.
- #4: Create the discount curve from the implied discount rates.
- #5: Extract the discount curve instruments and quotes.
- #6: Create a parallel shifted discount curve.
- #7: Create a rate shifted discount curve.
- #8: Create a basis rate shifted discount curve.
- #9: Create a node tweaked discount curve.
- #10: Compute the effective discount factor between 2 dates.
- #11: Compute the effective implied rate between 2 dates.

ZeroCurveRegressor

ZeroCurveRegressor implements the regression analysis set for the Zero Curve. The regression tests do the following:

- Build a discount curve, followed by the zero curve.
- Regressor #1: Compute zero curve discount factors.
- Regressor #2: Compute zero curve zero rates.

CreditCurveRegressor

CreditCurveRegressor implements the regression set analysis for the Credit Curve. CreditCurveRegressor regresses 12 scenarios:

- #1: Create an SNAC CDS.
- #2: Create the credit curve from a set of CDS instruments.
- #3: Create the credit curve from a flat hazard rate.
- #4: Create the credit curve from a set of survival probabilities.
- #5: Create the credit curve from an array of hazard rates.
- #6: Extract the credit curve instruments and quotes.
- #7: Create a parallel hazard shifted credit curve.
- #8: Create a parallel quote shifted credit curve.
- #9: Create a node tweaked credit curve.
- #10: Set a specific default date on the credit curve.
- #11: Compute the effective survival probability between 2 dates.
- #12: Compute the effective hazard rate between 2 dates.

FXCurveRegressor

FXCurveRegressor implements the regression analysis set for the FX Curve.

FXCurveRegressor implements 3 regression tests:

- #1: FX Basis and FX Curve Creation: Construct a FX forward Curve from an array of FX forward nodes and the spot.
- #2: Imply the FX Forward given the domestic and foreign discount curves.
- #3a: Compute the domestic and foreign basis given the market FX forward.
- #3b: Build the domestic/foreign basis curve given the corresponding basis nodes.
- #3c: Imply the array of FX forward points/PIPs from the array of basis and domestic/foreign discount curves.

CreditAnalyticsRegressionEngine

CreditAnalyticsRegressionEngine implements the RegressionEngine for the curve regression. It adds the CreditCurveRegressor, DiscountCurveRegressor, FXCurveRegressor, and ZeroCurveRegressor, and launches the regression engine.

Curve Jacobian Regression Suite

The core functionality of the curve Jacobian regression library is implemented in the package org.drip.regression.curveJacobian. The Product Curve Jacobian Regression package carries out regression across the core suite of products Jacobian to the curve– Cash, EDF, and Fix-float IRS. It also implements the Curve Jacobian Regression Engine.

Functionality in this package is implemented over 5 classes – CashJacobianRegressorSet, EDFJacobianRegressorSet, IRSJacobianRegressorSet, DiscountCurveJacobianRegressorSet, and CurveJacobianRegressionEngine.

CashJacobianRegressorSet

CashJacobianRegressorSet implements the regression analysis set for the Cash product related Sensitivity Jacobians. Specifically, it computes the PVDF micro-Jack.

EDFJacobianRegressorSet

EDFJacobianRegressorSet implements the regression analysis set for the EDF product related Sensitivity Jacobians. Specifically, it computes the PVDF micro-Jack.

IRSJacobianRegressorSet

IRSJacobianRegressorSet implements the regression analysis set for the IRS product related Sensitivity Jacobians. Specifically, it computes the PVDF micro-Jack.

DiscountCurveJacobianRegressorSet

DiscountCurveJacobianRegressorSet implements the regression analysis for the full discount curve (built from cash/future/swap) Sensitivity Jacobians. Specifically, it computes the PVDF micro-Jack.

CurveJacobianRegressionEngine

CurveJacobianRegressionEngine implements the RegressionEngine for the curve Jacobian regression. It adds the CashJacobianRegressorSet, the EDFJacobianRegressorSet, the IRSJacobianRegressorSet, and the DiscountCurveJacobianRegressorSet, and launches the regression engine.

Fixed-Point Finder Regression Suite

The core functionality of the non-linear fixed-point finder regression library is implemented in the package org.drip.regression.fixedpointfinder. This contains the suite for regression testing of the non-linear univariate fixed-point finder.

Functionality in this package is implemented over 4 classes – OpenRegressorSet, BracketingRegressorSet, CompoundBracketingRegressorSet, and FixedPointFinderRegressionEngine.

OpenRegressorSet

OpenRegressorSet implements the regression run for the Open (i.e., Newton) Fixed Point Search Method.

BracketingRegressorSet

BracketingRegressorSet implements regression run for the Primitive Bracketing Fixed Point Search Method. It implements the following 4 primitive bracketing schemes: Bisection, False Position, Quadratic, and Inverse Quadratic.

CompoundBracketingRegressorSet

CompoundBracketingRegressorSet implements regression run for the Compound Bracketing Fixed Point Search Method. It implements the following 2 compound bracketing schemes: Brent and Zheng.

FixedPointFinderRegressionEngine

FixedPointFinderRegressionEngine implements the RegressionEngine for the Fixed Point Finder regression. It adds the OpenRegressorSet, the BracketingRegressorSet, and the CompoundBracketingRegressorSet, and launches the regression engine.

Basis Spline Regression Suite

The core functionality of the basis spline regression library is implemented in the package org.drip.regression.spline. This package contains the random input regression runs on the spline and stretch instances. Runs regress on C1Hermite, local control smoothing, single segment Lagrangians, multi-segment sequences using a variety of spline/stretch basis functions and controls.

Functionality in this package is implemented over 6 classes - BasisSplineRegressor, HermiteBasisSplineRegressor, LagrangePolynomialStretchRegressor, LocalControlBasisSplineRegressor, BasisSplineRegressorSet, and BasisSplineRegressionEngine.

BasisSplineRegressor

BasisSplineRegressor implements the custom basis spline regressor for the given basis spline. As part of the regression run, it executes the following:

- Calibrate and compute the left and he right Jacobian.
- Reset right node and re-run calibration.
- Compute an intermediate value Jacobian.

HermiteBasisSplineRegressor

HermiteBasisSplineRegressor implements the BasisSplineRegressor using the Hermite basis spline regressor.

LagrangePolynomialStretchRegressor

LagrangePolynomialStretchRegressor implements the BasisSplineRegressor using the SingleSegmentLagrangePolynomial regressor.

LocalControlBasisSplineRegressor

LocalControlBasisSplineRegressor implements the local control basis spline regressor for the given basis spline. As part of the regression run, it executes the following:

- Calibrate and compute the left and the right Jacobian
- Insert the Local Control Hermite, Cardinal, and Catmull-Rom knots
- Run Regressor for the C1 Local Control C1 Slope Insertion Bessel/Hermite Spline
- Compute an intermediate value Jacobian

BasisSplineRegressorSet

BasisSplineRegressorSet carries out regression testing for the following series of basis splines:

- Polynomial Basis Spline, n = 2 basis functions, and C^1
- Polynomial Basis Spline, n = 3 basis functions, and C^1
- Polynomial Basis Spline, n = 4 basis functions, and C^1
- Polynomial Basis Spline, n = 4 basis functions, and C^2
- Polynomial Basis Spline, n = 5 basis functions, and C^1
- Polynomial Basis Spline, n = 5 basis functions, and C^2
- Polynomial Basis Spline, n = 5 basis functions, and C^3
- Polynomial Basis Spline, n = 6 basis functions, and C^1
- Polynomial Basis Spline, n = 6 basis functions, and C^2

- Polynomial Basis Spline, n = 6 basis functions, and C^3
- Polynomial Basis Spline, n = 6 basis functions, and C^4
- Polynomial Basis Spline, n = 7 basis functions, and C^1
- Polynomial Basis Spline, n = 7 basis functions, and C^2
- Polynomial Basis Spline, n = 7 basis functions, and C^3
- Polynomial Basis Spline, n = 7 basis functions, and C^4
- Polynomial Basis Spline, n = 7 basis functions, and C^5
- Bernstein Polynomial Basis Spline, n = 4 basis functions, and C^2
- Exponential Tension Spline, n = 4 basis functions, Tension = 1., and C^2
- Hyperbolic Tension Spline, n = 4 basis functions, Tension = 1., and C^2
- Kaklis-Pandelis Tension Spline, n = 4 basis functions, KP = 2, and C^2
- C1 Hermite Local Spline, n = 4 basis functions, and C^1
- Hermite Local Spline with Local, Catmull-Rom, and Cardinal Knots, n = 4 basis functions, and C^1

BasisSplineRegressionEngine

BasisSplineRegressionEngine implements the RegressionEngine class for the basis spline functionality.

DRIP MATH

DRIP MATH Library consists of the following 5 packages:

1. Univariate Function Package: The univariate function package implements the individual univariate functions, their convolutions, and reflections.

2. Univariate Calculus Package: The univariate calculus package implements univariate difference based arbitrary order derivative, implements differential control settings, implements several integrand routines, and multivariate Wengert Jacobian.

3. Univariate Distribution: This package implements the univariate distributions – currently normal and its variants.

4. Linear Algebra: This package implements the linear algebra functionality – matrix manipulation, inversion, and transformation, linear system solving, and linearization output representation.

5. DRIP Math Helpers: This package implements a collection of DRIP MATH helper utilities - collections processing, date manipulation, working with strings, real number utilities, and formatting functionality.

6. Univariate Non-linear Fixed Point Finder Solver: This package implements a number of univariate, non-linear fixed-point search routines. Methodology separates execution initialization from variate iteration. A variety of open and closed variate iteration techniques are implemented, along with primitive/complex closed variate iteration techniques.

DRIP MATH: Univariate Function

DRIP MATH Univariate Functions are available in the package *org.drip.quant.function1D*. The univariate function package implements the individual univariate functions, their convolutions, and reflections.

Functionality in this package is implemented over 11 classes - AbstractUnivariate, UnivariateConvolution, UnivariateReflection, Polynomial, BernsteinPolynomial, NaturalLogSeriesElement, ExponentialTension, HyperbolicTension, LinearRationalShapeControl, QuadraticRationalShapeControl, and LinearRationalTensionExponential.

AbstractUnivariate

This abstract class provides the evaluation of the given basis/objective function and its derivatives for a specified variate. Default implementations of the derivatives are for black box, non-analytical functions.

UnivariateConvolution

This class provides the evaluation of the point value and the derivatives of the convolution of 2 univariate functions for the specified variate.

UnivariateReflection

For a given variate x, this class provides the evaluation and derivatives of the reflection at $1-x$.

Polynomial

This class provides the evaluation of the n^{th} order polynomial and its derivatives for a specified variate. The degree n specifies the order of the polynomial.

BernsteinPolynomial

This class provides the evaluation of Bernstein polynomial and its derivatives for a specified variate. The degree exponent specifies the order of the Bernstein polynomial.

NaturalLogSeriesElement

This class provides the evaluation of a single term in the expansion series for the natural log. The exponent parameter specifies which term in the series is being considered.

ExponentialTension

This class provides the evaluation of exponential tension basis function and its derivatives for a specified variate. It can be customized by the choice of exponent, the base, and the tension parameter.

HyperbolicTension

This class provides the evaluation of hyperbolic tension basis function and its derivatives for a specified variate. It can be customized by the choice of the hyperbolic function and the tension parameter.

LinearRationalShapeControl

This class implements the deterministic rational shape control functionality on top of the estimate of the basis splines inside - $[0,...,1)$ - Globally $[x_0,...,x_1)$: $y = \frac{1}{1+\lambda x}$ where is the normalized ordinate mapped as $x = \frac{x - x_{i-1}}{x_i - x_{i-1}}$.

QuadraticRationalShapeControl

This class implements the deterministic rational shape control functionality on top of the estimate of the basis splines inside - $[0,...,1)$ - Globally $[x_0,...,x_1)$: $y = \frac{1}{1+\lambda x(1-x)}$ where is the normalized ordinate mapped as $x = \frac{x - x_{i-1}}{x_i - x_{i-1}}$.

LinearRationalTensionExponential

This class provides the evaluation of the Convolution of the Linear Rational and the Tension Exponential Function and its derivatives for a specified variate.

DRIP MATH: Univariate Calculus

DRIP MATH Univariate Calculus functions are available in the package *org.drip.quant.calculus*. The univariate calculus package implements univariate difference based arbitrary order derivative, implements differential control settings, implements several integrand routines, and multivariate Wengert Jacobian.

Functionality in this package is implemented over 4 classes – DerivativeControl, Differential, Integrator, and WengertJacobian.

DerivativeControl

DerivativeControl provides bumps needed for numerically approximating derivatives. Bumps can be absolute or relative, and they default to a floor.

Differential

Differential holds the incremental differentials for the variate and the objective functions.

WengertJacobian

WengertJacobian contains the Jacobian of the given set of Wengert variables to the set of parameters. It exposes the following functionality:

- Set/Retrieve the Wengert variables
- Accumulate the Partials
- Scale the partial entries

- Merge the Jacobian with another
- Retrieve the WengertJacobian elements
- Display the contents of the WengertJacobian

Integrator

Integrator implements the following routines for integrating the objective functions:

- Linear Quadrature
- Mid-Point Scheme
- Trapezoidal Scheme
- Simpson/Simpson38 Schemes
- Boole Scheme

DRIP MATH: Univariate Distribution

DRIP MATH Univariate Distributions are available in the package *org.drip.quant.distribution*. This package implements the univariate distributions – currently normal and its variants.

Functionality in this package is implemented over 2 classes – Univariate and UnivariateNormal.

Univariate

Univariate implements the base abstract class behind univariate distributions. It exports methods for incremental, cumulative, and inverse cumulative distribution densities.

UnivariateNormal

UnivariateNormal implements the univariate normal distribution. It implements incremental, cumulative, and inverse cumulative distribution densities.

DRIP MATH: Linear Algebra

DRIP MATH Linear Algebra Functions are available in the package *org.drip.quant.linearalgebra*. This package implements the linear algebra functionality – matrix manipulation, inversion, and transformation, linear system solving, and linearization output representation.

Functionality in this package is implemented over 4 classes – LinearizationOutput, MatricComplementTransform, Matrix, and LinearSystemSolver.

LinearizationOutput

LinearizationOutput holds the output of a sequence of linearization operations. It contains the transformed original matrix, the transformed RHS, and the method used for the linearization operation.

MatrixComplementTransform

This class holds the results of Matrix transforms on the source and the complement, e.g., during a Matrix Inversion Operation.

Matrix

Matrix implements Matrix manipulation routines. It exports the following functionality:

- Matrix Inversion using Closed form solutions (for low-dimension matrices), or using Gaussian elimination

- Matrix Product
- Matrix Diagonalization and Diagonal Pivoting
- Matrix Regularization through Row Addition/Row Swap

MatrixComplementTransform

LinearSystemSolver implements the solver for a system of linear equations given by $Ax = B$, where A is the matrix, x the set of variables, and B is the result to be solved for. It exports the following functions:

- Row Regularization and Diagonal Pivoting
- Check for Diagonal Dominance
- Solving the linear system using any one of the following: Gaussian Elimination, Gauss Seidel reduction, or matrix inversion

DRIP MATH: Helper Utilities

DRIP MATH Helper Utilities are available in the package *org.drip.quant.common*. This package implements a collection of DRIP MATH helper utilities - collections processing, date manipulation, working with strings, real number utilities, and formatting functionality.

Functionality in this package is implemented over 5 classes – CollectionUtil, DateUtil, StringUtil, NumberUtil, and FormatUtil.

CollectionUtil

The CollectionUtil class implements generic utility functions used in DRIP modules. Some of the functions it exposes are:

- Map Merging Functionality
- Map Key Functionality - key-value flatteners, key prefixers
- Decompose/transform List/Set/Array Contents
- Multi-Dimensional Map Manipulator Routines
- Construct n-derivatives array from Slope
- Collate Wengerts to a bigger Wengert

DateUtil

DateUtil implements date utility functions those are extraneous to the JulianDate implementation. It exposes the following functionality:

- Retrieve Day, Month, and Year From Java Date

- Switch between multiple date formats (Oracle Date, BBG Date, different string representations etc)

StringUtil

StringUtil implements string utility functions. It exports the following functions:

- Decompose + Transform string arrays into appropriate target type set/array/list, and vice versa
- General-purpose String processor functions, such as GUID generator, splitter, type converter and input checker

NumberUtil

NumberUtil implements number utility functions. It exposes the following functions:

- Verify number/number array validity, and closeness/sign match
- Factorial Permutation/Combination functionality
- Dump multi-dimensional array contents
- Min/Max/Bound the array entries within limits

FormatUtil

FormatUtil implements formatting utility functions. Currently it just exports functions to pad and format.

DRIP MATH: Univariate Non-linear Fixed Point Finder Solver

The core functionality of the DRIP non-linear fixed-point search library is implemented in the package org.drip.math.solver1D. This package implements a number of univariate, non-linear fixed-point search routines. Methodology separates execution initialization from variate iteration. A variety of open and closed variate iteration techniques are implemented, along with primitive/complex closed variate iteration techniques.

Functionality in this package is implemented over 19 classes – BracketingControlParams, BracketingOutput, ConvergenceControlParams, ConvergenceOutput, ExecutionControl, ExecutionControlParams, ExecutionInitializationOutput, ExecutionInitializer, FixedPointFinder, FixedPointFinderBracketing, FixedPointFinderBrent, FixedPointFinderNewton, FixedPoinderOutput, FixedPointFinderZheng, InitializationHeuristics, IteratedBracket, IteratedVariate, VariateIterationSelectionParams, and VariateIteratorPrimitive.

BracketingControlParams

BracketingControlParams implements the control parameters for bracketing solutions. BracketingControlParams provides the following parameters:

- The starting variate from which the search for bracketing begins.
- The initial width for the brackets.
- The factor by which the width expands with each iterative search.
- The number of such iterations.

BracketingOutput

BracketingOutput carries the results of the bracketing initialization. In addition to the fields of ExecutionInitializationOutput, BracketingOutput holds the left/right bracket variates and the corresponding values for the objective function.

ConvergenceControlParams

ConvergenceControlParams holds the fields needed for the controlling the execution of Newton's method. ConvergenceControlParams does that using the following parameters:

- The determinant limit below which the convergence zone is deemed to have reached.
- Starting variate from where the convergence search is kicked off.
- The factor by which the variate expands across each iterative search.
- The number of search iterations.

ConvergenceOutput

ConvergenceOutput extends the ExecutionInitializationOutput by retaining the starting variate that results from the convergence zone search. ConvergenceOutput does not add any new field to ExecutionInitializationOutput.

ExecutionControl

ExecutionControl implements the core fixed-point search execution control and customization functionality. ExecutionControl is used for a) calculating the absolute tolerance, and b) determining whether the OF has reached the goal. ExecutionControl determines the execution termination using its ExecutionControlParams instance.

ExecutionControlParams

ExecutionControlParams holds the parameters needed for controlling the execution of the fixed-point finder. ExecutionControlParams fields control the fixed-point search in one of the following ways:

- Number of iterations after which the search is deemed to have failed.
- Relative Objective Function Tolerance Factor which, when reached by the objective function, will indicate that the fixed point has been reached.
- Variate Convergence Factor, factor applied to the initial variate to determine the absolute convergence.
- Absolute Tolerance fall-back, which is used to determine that the fixed point has been reached when the relative tolerance factor becomes zero.
- Absolute Variate Convergence Fall-back, fall-back used to determine if the variate has converged.

ExecutionInitializationOutput

ExecutionInitializationOutput holds the output of the root initializer calculation. The following are the fields held by ExecutionInitializationOutput:

- Whether the initialization completed successfully.
- The number of iterations, the number of objective function calculations, and the time taken for the initialization.
- The starting variate from the initialization

ExecutionInitializer

ExecutionInitializer implements the initialization execution and customization functionality. ExecutionInitializer performs two types of variate initializations:

- Bracketing initialization: This brackets the fixed point using the bracketing algorithm described in http://www.credit-trader.org. If successful, a pair of variate/OF coordinate nodes that bracket the fixed-point is generated. These brackets are eventually used by routines that iteratively determine the fixed-point. Bracketing initialization is controlled by the parameters in BracketingControlParams.
- Convergence Zone initialization: This generates a variate that lies within the convergence zone for the iterative determination of the fixed point using the Newton's method. Convergence Zone Determination is controlled by the parameters in ConvergenceControlParams.

ExecutionInitializer behavior can be customized/optimized through several of the initialization heuristics techniques implemented in the InitializationHeuristics class.

FixedPointFinder

FixedPointFinder is the base abstract class that is implemented by customized invocations, e.g., Newton's method, or any of the bracketing methodologies. FixedPointFinder invokes the core routine for determining the fixed point from the goal. The ExecutionControl determines the execution termination. The initialization heuristics implements targeted customization of the search.

FixedPointFinder main flow comprises of the following steps:

- Initialize the fixed-point search zone by determining either a) the brackets, or b) the starting variate.
- Compute the absolute OF tolerance that establishes the attainment of the fixed point.
- Launch the variate iterator that iterates the variate.
- Iterate until the desired tolerance has been attained.
- Return the fixed-point output.

Fixed point finders that derive from this provide implementations for the following:

- Variate initialization: They may choose either bracketing initializer, or the convergence initializer - functionality is provided for both in this module.
- Variate Iteration: Variates are iterated using a) any of the standard primitive built-in variate iterators (or custom ones), or b) a variate selector scheme for each iteration.

FixedPointFinderBracketing

FixedPointFinderBracketing customizes the FixedPointFinder for bracketing based fixed-point finder functionality.

FixedPointFinderBracketing applies the following customization:

- Initializes the fixed-point finder by computing the starting brackets.
- Iterating the next search variate using one of the specified variate iterator primitives.

By default, FixedPointFinderBracketing does not do compound iterations of the variate using any schemes - that is done by classes that extend it.

FixedPointFinderBrent

FixedPointFinderBrent customizes FixedPointFinderBracketing by applying the Brent's scheme of compound variate selector.

Brent's scheme, as implemented here, is described in http://www.credit-trader.org. This implementation retains absolute shifts that have happened to the variate for the past 2 iterations as the discriminant that determines the next variate to be generated.

FixedPointFinderBrent uses the following parameters specified in VariateIterationSelectorParams:

- The Variate Primitive that is regarded as the "fast" method.
- The Variate Primitive that is regarded as the "robust" method.
- The relative variate shift that determines when the "robust" method is to be invoked over the "fast".
- The lower bound on the variate shift between iterations that serves as the fall-back to the "robust".

FixedPointFinderNewton

FixedPointFinderNewton customizes the FixedPointFinder for Open (Newton's) fixed-point finder functionality.

FixedPointFinderNewton applies the following customization:

- Initializes the fixed point finder by computing a starting variate in the convergence zone.
- Iterating the next search variate using the Newton's method.

FixedPointFinderOutput

FixedPointFinderOutput holds the result of the fixed-point search.

FixedPointFinderOutput contains the following fields:

- Whether the search completed successfully
- The number of iterations, the number of objective function base/derivative calculations, and the time taken for the search
- The output from initialization

FixedPointFinderZheng

FixedPointFinderZheng implements the fixed-point locator using Zheng's improvement to Brent's method.

FixedPointFinderZheng overrides the iterateCompoundVariate method to achieve the desired simplification in the iterative variate selection.

InitializationHeuristics

InitializationHeuristics implements several heuristics used to kick off the fixed-point bracketing/search process.

The following custom heuristics are implemented as part of the heuristics based kick-off:

- Custom Bracketing Control Parameters: Any of the standard bracketing control parameters can be customized to kick-off the bracketing search.
- Soft Left/Right Bracketing Hints: The left/right-starting bracket edges are used as soft bracketing initialization hints.
- Soft Mid Bracketing Hint: A mid bracketing level is specified to indicate the soft bracketing kick-off.
- Hard Bracketing Floor/Ceiling: A pair of hard floor and ceiling limits is specified as a constraint to the bracketing.
- Hard Search Boundaries: A pair of hard left and right boundaries is specified to kick-off the final fixed-point search.

These heuristics are further interpreted and developed inside the ExecutionInitializer and the ExecutionControl implementations.

IteratedBracket

IteratedBracket holds the left/right bracket variates and the corresponding values for the objective function during each iteration.

IteratedVariate

IteratedVariate holds the variate and the corresponding value for the objective function during each iteration.

VariateIterationSelectionParams

VariateIterationSelectorParams implements the control parameters for the compound variate selector scheme used in Brent's method.

Brent's method uses the following fields in VariateIterationSelectorParams to generate the next variate:

- The Variate Primitive that is regarded as the "fast" method.
- The Variate Primitive that is regarded as the "robust" method.
- The relative variate shift that determines when the "robust" method is to be invoked over the "fast".
- The lower bound on the variate shift between iterations that serves as the fall-back to the "robust".

VariateIteratorPrimitive

VariateIteratorPrimitive implements the various Primitive Variate Iterator routines.

VariateIteratorPrimitive implements the following iteration primitives:

- Bisection
- False Position
- Quadratic
- Inverse Quadratic
- Ridder

It may be readily enhanced to accommodate additional primitives.

Spline Builder

Spline Builder Library consists of the following 8 packages:

1. Spline Parameters: The spline parameters package implements the segment and stretch level construction, design, penalty, and shape control parameters.

2. Spline Basis Function Set: The spline basis function set package implements the basis set, parameters for the different basis functions, parameters for basis set construction, and parameters for B Spline sequence construction.

3. Spline Segment: The spline segment package implements the segment's inelastic state, the segment basis evaluator, the segment flexure penalizer, computes the segment monotonicity behavior, and implements the segment's complete constitutive state.

4. Spline Stretch: The spline stretch package provides single segment and multi segment interfaces, builders, and implementations, along with custom boundary settings.

5. Spline Grid/Span: The spline grid/span package provides the multi-stretch spanning functionality. It specifies the span interface, and provides implementations of the overlapping and the non-overlapping span instances. It also implements the transition splines with custom transition zones.

6. Spline PCHIP: The spline PCHIP package implements most variants of the local piece-wise cubic Hermite interpolating polynomial smoothing functionality. It provides a number of tweaks for smoothing customization, as well as providing enhanced implementations of Akima, Preuss, and Hagan-West smoothing interpolators.

7. Spline B Spline: The spline B Spline package implements the raw and the processed basis B Spline hat functions. It provides the standard implementations for the monic and the multic B Spline Segments. It also exports functionality to generate higher order B Spline Sequences.

8. Tension Spline: The tension spline package implements closed form family of cubic tension splines laid out in the basic framework outlined in Koch and Lyche (1989), Koch and Lyche (1993), and Kvasov (2000).

Spline Builder: Spline Parameters

Spline Builder Spline Parameters functions are available in the package *org.drip.spline.params*. The spline parameters package implements the segment and stretch level construction, design, penalty, and shape control parameters.

Functionality in this package is implemented over 11 classes – ResponseScalingShapeControl, SegmentBasisFlexureConstraint, SegmentResponseValueConstraint, SegmentResponseConstraintSet, SegmentBestFitResponse, StretchBestFitResponse, SegmentFlexurePenaltyControl, SegmentDesignInelasticControl, SegmentCustomBuilderControl, SegmentPredictorResponseDerivative, and SegmentStateCalibration.

ResponseScalingShapeControl

This class implements the segment level basis functions proportional adjustment to achieve the desired shape behavior of the response. In addition to the actual shape controller function, it interprets whether the control is applied on a local or global predicate ordinate basis.

SegmentBasisFlexureConstraint

This class holds the set of fields needed to characterize a single local linear Constraint, expressed linearly as a combination of the local Predictor Ordinates and their corresponding Response Basis Function Realizations. Constraints are expressed as $C_j = \sum_i W_i \beta_i \left(x_j \right)$ where x_j is the predictor ordinate at node j, β_i is the Coefficient for the Response Basis Function i, W_i is the weight applied for the Response Basis Function

i, and C_j is the value of constraint j. SegmentBasisFlexureConstraint may be viewed as the localized basis function transpose of SegmentResponseValueConstraint.

SegmentResponseValueConstraint

This class holds the following set of fields that characterize a single global linear constraint between the predictor and the response variables within a single segment, expressed linearly across the constituent nodes. SegmentBasisFlexureConstraint may be viewed as the localized basis function transpose of SegmentResponseValueConstraint. SegmentResponseValueConstraint exports the following functionality:

- Retrieve the Array of Predictor Ordinates
- Retrieve the Array of Response Weights at each Predictor Ordinate
- Retrieve the Constraint Value
- Convert the Segment Constraint onto Local Predictor Ordinates, the corresponding Response Basis Function, and the Shape Controller Realizations
- Get the Position of the Predictor Knot relative to the Constraints
- Generate a SegmentResponseValueConstraint instance from the given predictor/response pair

SegmentResponseConstraintSet

This class holds the set of SegmentResponseValueConstraint (Base + One/more Sensitivities) for the given Segment. It exposes functions to add/retrieve the base response value constraints as well as additional response value constraint sensitivities.

SegmentBestFitResponse

This class implements basis per-segment Fitness Penalty Parameter Set. Currently it contains the Best Fit Penalty Weight Grid Matrix and the corresponding Segment Local Predictor Ordinate/Response Match Pair.

StretchBestFitResponse

This class implements basis per-Stretch Fitness Penalty Parameter Set. Currently it contains the Best Fit Penalty Weight Grid Matrix and the corresponding Local Predictor Ordinate/Response Match Pair. StretchBestFitResponse exports the following methods:

- Retrieve the Array of the Fitness Weights
- Retrieve the Indexed Fitness Weight Element
- Retrieve the Array of Predictor Ordinates
- Retrieve the Indexed Predictor Ordinate Element
- Retrieve the Array of Responses
- Retrieve the Indexed Response Element
- Retrieve the Number of Fitness Points
- Generate the Segment Local Best Fit Weighted Response contained within the specified Segment
- Construct the StretchBestFitResponse Instance from the given Inputs
- Construct the StretchBestFitResponse Instance from the given Predictor Ordinate/Response Pairs, using Uniform Weightings

SegmentFlexurePenaltyControl

This class implements basis per-segment Flexure Penalty Parameter Set. Currently it contains the Flexure Penalty Derivative Order and the Roughness Coefficient Amplitude. Flexure Penalty Control may be used to implement Segment Curvature Control and/or Segment Length Control.

SegmentDesignInelasticControl

This class implements basis per-segment inelastic parameter set. It exports the following functionality:

- Retrieve the Continuity Order.
- Retrieve the Length Penalty and the Curvature Penalty Parameters.
- Create the C^2 Design Inelastic Parameters.
- Create the Design Inelastic Parameters for the desired C^k Criterion and the Roughness Penalty Order.

SegmentCustomBuilderControl

This class holds the parameters the guide the creation/behavior of the segment. It holds the segment elastic/inelastic parameters and the named basis function set.

SegmentPredictorResponseDerivative

This class contains the segment local parameters used for the segment calibration. It holds the edge Input Response value and its derivatives. It exposes the following functions:

- Retrieve the Response Value as well as the DResponseDPredictorOrdinate Array
- Aggregate the 2 Predictor Ordinate Response Derivatives by applying the Cardinal Tension Weight

SegmentStateCalibration

This class implements basis per-segment Calibration Parameter Set. It exposes the following functionality:

- Retrieve the Array of the Calibration Predictor Ordinates
- Retrieve the Array of the Calibration Response Values
- Retrieve the Array of the Left/Right Edge Derivatives
- Retrieve the Segment Best Fit Response
- Retrieve the Array of Segment Basis Flexure Constraints

Spline Builder: Spline Basis Function Set

Spline Builder Spline Basis Function Set functions are available in the package *org.drip.spline.basis*. The spline basis function set package implements the basis set, parameters for the different basis functions, parameters for basis set construction, and parameters for B Spline sequence construction.

Functionality in this package is implemented over 9 classes –FunctionSet, FunctionSetBuilderParams, SegmentBestFitResponse, ExponentialTensionSetParams, ExponentialRationalSetParams, PolynomialFunctionSetParams, KaklisPandelisSetParams, FunctionSetBuilder, and BSplineSequenceParams.

FunctionSet

This class implements the general-purpose basis spline function set.

FunctionSetBuilderParams

This is an empty stub class whose derived implementations hold the per-segment basis set parameters.

ExponentialMixtureSetParams

ExponentialMixtureSetParams implements per-segment parameters for the exponential mixture basis set - the array of the exponential tension parameters, one per each entity in the mixture.

ExponentialTensionSetParams

ExponentialTensionSetParams implements per-segment parameters for the exponential tension basis set – currently it only contains the tension parameter.

ExponentialRationalSetParams

ExponentialRationalSetParams implements per-segment parameters for the exponential rational basis set – the exponential tension and the rational tension parameters.

PolynomialFunctionSetParams

PolynomialFunctionSetParams implements per-segment basis set parameters for the polynomial basis spline - currently it holds the number of basis functions.

KaklisPandelisSetParams

KaklisPandelisSetParams implements per-segment parameters for the Kalkis-Pandelis basis set – currently it only holds the polynomial tension degree.

FunctionSetBuilder

This class implements the basis set and spline builder for the following types of splines:

- Exponential basis tension splines

- Hyperbolic basis tension splines
- Polynomial basis splines
- Bernstein Polynomial basis splines
- Kaklis-Pandelis basis tension splines

The elastic coefficients for the segment using C^k basis splines inside [0,...,1) - globally $[x_0,...,x_1)$: $y = BasisSplineFunction(C^k, x) \times ShapeController(x)$ where is the normalized ordinate mapped as $x = \frac{x - x_{i-1}}{x_i - x_{i-1}}$. The inverse quadratic/rational spline is a typical shape controller spline used.

BSplineSequenceParams

BSplineSequenceParams implements the parameter set for constructing the B Spline Sequence. It provides functionality to:

- Retrieve the B Spline Order
- Retrieve the Number of Basis Functions
- Retrieve the Processed Basis Derivative Order
- Retrieve the Basis Hat Type
- Retrieve the Shape Control Type
- Retrieve the Tension
- Retrieve the Array of Predictor Ordinates

Spline Builder: Spline Segment

Spline Builder Segment functions are available in the package *org.drip.spline.segment*. The spline segment package implements the segment's inelastic state, the segment basis evaluator, the segment flexure penalizer, computes the segment monotonicity behavior, and implements the segment's complete constitutive state.

Functionality in this package is implemented over 6 classes - InelasticConstitutiveState, BasisEvaluator, SegmentBasisEvaluator, Monotonicity, BestFitFlexurePenalizer, and ConstitutiveState.

InelasticConstitutiveState

This class contains the spline segment in-elastic fields - in this case the start/end ranges. It exports the following functions:

- Retrieve the Segment Left/Right Predictor Ordinate
- Find out if the Predictor Ordinate is inside the segment - inclusive of left/right
- Get the Width of the Predictor Ordinate in this Segment
- Transform the Predictor Ordinate to the Local Segment Predictor Ordinate
- Transform the Local Predictor Ordinate to the Segment Ordinate

BasisEvaluator

This interface implements the Segment's Basis Evaluator Functions. It exports the following functions:

- Retrieve the number of Segment's Basis Functions
- Set the Inelastics that provides the enveloping Context the Basis Evaluation

- Clone/Replicate the current Basis Evaluator Instance
- Compute the Response Value of the indexed Basis Function at the specified Predictor Ordinate
- Compute the Basis Function Value at the specified Predictor Ordinate
- Compute the Response Value at the specified Predictor Ordinate
- Compute the Ordered Derivative of the Response Value off of the indexed Basis Function at the specified Predictor Ordinate
- Compute the Ordered Derivative of the Response Value off of the Basis Function Set at the specified Predictor Ordinate
- Compute the Response Value Derivative at the specified Predictor Ordinate

SegmentBasisEvaluator

This class implements the BasisEvaluator interface for the given Set of the Segment Basis Evaluator Functions.

Monotonicity

This class contains the monotonicity details related to the given segment. It computes whether the segment is monotonic, and if not, whether it contains a maximum, a minimum, or an inflection.

BestFitFlexurePenalizer

This Class implements the Segment's Best Fit, Curvature, and Length Penalizers. It provides the following functionality:

- Compute the Cross-Curvature Penalty for the given Basis Pair

- Compute the Cross-Length Penalty for the given Basis Pair
- Compute the Best Fit Cross-Product Penalty for the given Basis Pair
- Compute the Basis Pair Penalty Coefficient for the Best Fit and the Curvature Penalties
- Compute the Penalty Constraint for the Basis Pair

ConstitutiveState

ConstitutiveState implements the single segment basis calibration and inference functionality. It exports the following functionality:

- Build the ConstitutiveState instance from the Basis Function/Shape Controller Set
- Build the ConstitutiveState instance from the Basis Evaluator Set
- Retrieve the Number of Parameters, Basis Evaluator, Array of the Response Basis Coefficients, and Segment Design Inelastic Control
- Calibrate the Segment State from the Calibration Parameter Set
- Sensitivity Calibrator: Calibrate the Segment Quote Jacobian from the Calibration Parameter Set
- Calibrate the coefficients from the prior Predictor/Response Segment, the Constraint, and fitness Weights
- Calibrate the coefficients from the prior Segment and the Response Value at the Right Predictor Ordinate
- Calibrate the Coefficients from the Edge Response Values and the Left Edge Response Slope
- Calibrate the coefficients from the Left Edge Response Value Constraint, the Left Edge Response Value Slope, and the Right Edge Response Value Constraint
- Retrieve the Segment Curvature, Length, and the Best Fit DPE
- Calculate the Response Value and its Derivative at the given Predictor Ordinate
- Calculate the Ordered Derivative of the Coefficient to the Quote

- Calculate the Jacobian of the Segment's Response Basis Function Coefficients to the Edge Inputs
- Calculate the Jacobian of the Response to the Edge Inputs at the given Predictor Ordinate
- Calculate the Jacobian of the Response to the Basis Coefficients at the given Predictor Ordinate
- Calibrate the segment and calculate the Jacobian of the Segment's Response Basis Function Coefficients to the Edge Parameters
- Calibrate the Coefficients from the Edge Response Values and the Left Edge Response Value Slope and calculate the Jacobian of the Segment's Response Basis Function Coefficients to the Edge Parameters
- Calibrate the coefficients from the prior Segment and the Response Value at the Right Predictor Ordinate and calculate the Jacobian of the Segment's Response Basis Function Coefficients to the Edge Parameters
- Indicate whether the given segment is monotone. If monotone, may optionally indicate the nature of the extrema contained inside maxima/minima/infection)
- Clip the part of the Segment to the Right of the specified Predictor Ordinate. Retain all other constraints the same
- Clip the part of the Segment to the Left of the specified Predictor Ordinate. Retain all other constraints the same
- Display the string representation for diagnostic purposes

Spline Builder: Spline Stretch

Spline Builder Spline Stretch functions are available in the package *org.drip.spline.stretch*. The spline stretch package provides single segment and multi segment interfaces, builders, and implementations, along with custom boundary settings.

Functionality in this package is implemented over 9 classes - BoundarySettings, SingleSegmentSequence, SingleSegmentLagrangePolynomial, MultiSegmentSequence, SegmentSequenceBuilder, CkSegmentSequenceBuilder, CalibratableMultiSegmentSequence, MultiSegmentSequenceBuilder, and MultiSegmentSequenceModifier.

BoundarySettings

This class implements the Boundary Settings that determine the full extent of description of the stretch's State. It exports functions that:

- Specify the type of the boundary condition (NATURAL/FLOATING/IS-A-KNOT)
- Boundary Condition specific additional parameters (e.g., Derivative Orders and Matches)
- Static methods that help construct standard boundary settings

SingleSegmentSequence

SingleSegmentSequence is the interface that exposes functionality that spans multiple segments. Its derived instances hold the ordered segment sequence, the segment control parameters, and, if available, the spanning Jacobian. SingleSegmentSequence exports the following group of functionality:

- Construct adjoining segment sequences in accordance with the segment control parameters
- Calibrate according to a varied set of (i.e., NATURAL/FINANCIAL) boundary conditions
- Estimate both the value, the ordered derivatives, and the Jacobian (quote/coefficient) at the given ordinate
- Compute the monotonicity details - segment/Stretch level monotonicity, co-monotonicity, local monotonicity
- Predictor Ordinate Details - identify the left/right predictor ordinate edges, and whether the given predictor ordinate is a knot

SingleSegmentLagrangePolynomial

SingleSegmentLagrangePolynomial implements the SingleSegmentSequence Stretch interface using the Lagrange Polynomial Estimator. As such it provides a perfect fit that travels through all the predictor/response pairs causing Runge's instability.

MultiSegmentSequence

MultiSegmentSequence is the interface that exposes functionality that spans multiple segments. Its derived instances hold the ordered segment sequence, the segment control parameters, and, if available, the spanning Jacobian. MultiSegmentSequence exports the following group of functionality:

- Retrieve the Segments and their Builder Parameters
- Compute the monotonicity details - segment/Stretch level monotonicity, co-monotonicity, local monotonicity
- Check if the Predictor Ordinate is in the Stretch Range, and return the segment index in that case

- Set up (i.e., calibrate) the individual Segments in the Stretch by specifying one/or more of the node parameters and Target Constraints
- Set up (i.e., calibrate) the individual Segment in the Stretch to the Target Segment Edge Values and Constraints. This is also called the Hermite setup - where the segment boundaries are entirely locally set
- Generate a new Stretch by clipping all the Segments to the Left/Right of the specified Predictor Ordinate. Smoothness Constraints will be maintained.
- Retrieve the Span Curvature/Length, and the Best Fit DPE's
- Retrieve the Merge Stretch Manager
- Display the Segments

SegmentSequenceBuilder

SegmentSequenceBuilder is the interface that contains the stubs required for the construction of the segment stretch. It exposes the following functions:

- Set the Stretch whose Segments are to be calibrated
- Retrieve the Calibration Boundary Condition
- Calibrate the Starting Segment using the Left Slope
- Calibrate the Segment Sequence in the Stretch

CkSegmentSequenceBuilder

CkSegmentSequenceBuilder implements the SegmentSequenceBuilder interface to customize segment sequence construction. Customization is applied at several levels:

- Segment Calibration Boundary Setting/Segment Best Fit Response Settings
- Segment Response Value Constraints for the starting and the subsequent Segments

CalibratableMultiSegmentSequence

CalibratableMultiSegmentSequence implements the MultiSegmentSequence span that spans multiple segments. It holds the ordered segment sequence, segment sequence builder, the segment control parameters, and, if available, the spanning Jacobian. It provides a variety of customization for the segment construction and state representation control.

MultiSegmentSequenceBuilder

MultiSegmentSequenceBuilder exports Stretch creation/calibration methods to generate customized basis splines, with customized segment behavior using the segment control. It exports the following methods of Stretch Creation:

- Create an uncalibrated Stretch instance over the specified Predictor Ordinate Array using the specified Basis Spline Parameters for the Segment
- Create a calibrated Stretch Instance over the specified array of Predictor Ordinates and Response Values using the specified Basis Splines
- Create a calibrated Stretch Instance over the specified Predictor Ordinates, Response Values, and their constraints, using the specified Segment Builder Parameters
- Create a Calibrated Stretch Instance from the Array of Predictor Ordinates and a flat Response Value
- Create a Regression Spline Instance over the specified array of Predictor Ordinate Knot Points and the Set of the Points to be Best Fit

MultiSegmentSequenceModifier

MultiSegmentSequenceModifier exports Stretch modification/alteration methods to generate customized basis splines, with customized segment behavior using the segment control. It exposes the following stretch modification methods:

- Insert the specified Predictor Ordinate Knot into the specified Stretch, using the specified Response Value
- Append a Segment to the Right of the Specified Stretch using the Supplied Constraint
- Insert the Predictor Ordinate Knot into the specified Stretch
- Insert a Cardinal Knot into the specified Stretch at the specified Predictor Ordinate Location
- Insert a Catmull-Rom Knot into the specified Stretch at the specified Predictor Ordinate Location

Spline Builder: Spline Grid/Span

Spline Builder Spline Grid/Span functions are available in the package *org.drip.spline.grid*. The spline grid/span package provides the multi-stretch spanning functionality. It specifies the span interface, and provides implementations of the overlapping and the non-overlapping span instances. It also implements the transition splines with custom transition zones.

Functionality in this package is implemented over 2 classes - Span, and OverlappingStretchSpan.

Span

Span is the interface that exposes the functionality behind the collection of Stretches that may be overlapping or non-overlapping. It exposes the following stubs:

- Retrieve the Left/Right Span Edge
- Indicate if the specified Label is part of the Merge State at the specified Predictor Ordinate
- Compute the Response from the containing Stretches
- Add a Stretch to the Span
- Retrieve the first Stretch that contains the Predictor Ordinate
- Retrieve the Stretch by Name
- Calculate the Response Derivative to the Quote at the specified Ordinate
- Display the Span Edge Coordinates

OverlappingStretchSpan

OverlappingStretchSpan implements the Span interface, and the collection functionality of overlapping Stretches. In addition to providing a custom implementation of all the Span interface stubs, it also converts the Overlapping Stretch Span to a non-overlapping Stretch Span. Overlapping Stretches are clipped from the Left.

Spline Builder: Spline PCHIP

Spline Builder Spline PCHIP functions are available in the package *org.drip.spline.pchip*. The spline PCHIP package implements most variants of the local piece-wise cubic Hermite interpolating polynomial smoothing functionality. It provides a number of tweaks for smoothing customization, as well as providing enhanced implementations of Akima, Preuss, and Hagan-West smoothing interpolators.

Functionality in this package is implemented over 5 classes – AkimaLocalC1Generator, MinimalQuadraticHaganWest, MonotoneConvexHaganWest, LocalMonotoneCkGenerator, and LocalControlStretchBuilder.

AkimaLocalC1Generator

AkimaLocalC1Generator generates the local control C^1 Slope using the Akima (1970) Cubic Algorithm.

MinimalQuadraticHaganWest

This class implements the regime using the Hagan and West (2006) Minimal Quadratic Estimator.

MonotoneConvexHaganWest

This class implements the regime using the Hagan and West (2006) Estimator. It provides the following functionality:

- Static Method to create an instance of MonotoneConvexHaganWest
- Ensure that the estimated regime is monotone an convex
- If need be, enforce positivity and/or apply amelioration
- Apply segment-by-segment range bounds as needed
- Retrieve predictor ordinates/response values

LocalMonotoneCkGenerator

LocalMonotoneCkGenerator generates customized Local Stretch by trading off C^k for local control. This class implements the following variants: Akima, Bessel, Harmonic, Hyman83, Hyman89, Kruger, Monotone Convex, as well as the Van Leer and the Huynh/Le Floch limiters. It also provides the following custom control on the resulting C^1:

- Eliminate the Spurious Extrema in the Input C^1 Entry
- Apply the Monotone Filter in the Input C^1 Entry
- Generate a Vanilla C^1 Array from the specified Array of Predictor Ordinates and the Response Values
- Verify if the given Quintic Polynomial is Monotone using the Hyman89 Algorithm, and generate it if necessary

LocalControlStretchBuilder

LocalControlStretchBuilder exports Stretch creation/calibration methods to generate customized basis splines, with customized segment behavior using the segment control. It provides the following local-control functionality:

- Create a Stretch off of Hermite Splines from the specified the Predictor Ordinates, the Response
- Values, the Custom Slopes, and the Segment Builder Parameters

- Create Hermite/Bessel C1 Cubic Spline Stretch
- Create Hyman (1983) Monotone Preserving Stretch
- Create Hyman (1989) enhancement to the Hyman (1983) Monotone Preserving Stretch
- Create the Harmonic Monotone Preserving Stretch
- Create the Van Leer Limiter Stretch
- Create the Huynh Le Floch Limiter Stretch
- Generate the local control C1 Slope using the Akima Cubic Algorithm
- Generate the local control C1 Slope using the Hagan-West Monotone Convex Algorithm

Spline Builder: Spline B Spline

Spline Builder Spline B Spline functions are available in the package *org.drip.spline.bspline*. The spline B Spline package implements the raw and the processed basis B Spline hat functions. It provides the standard implementations for the monic and the multic B Spline Segments. It also exports functionality to generate higher order B Spline Sequences.

Functionality in this package is implemented over 17 classes - TensionBasisHat, TensionProcessedBasisHat, BasisHatShapeControl, LeftHatShapeControl, RightHatShapeControl, CubicRationalLeftRaw, CubicRationalRightRaw, ExponentialTensionLeftHat, ExponentialTensionRightHat, ExponentialTensionLeftRaw, ExponentialTensionRightRaw, BasisHatPairGenerator, SegmentBasisFunction, SegmentMonicBasisFunction, SegmentMulticBasisFunction, SegmentBasisFunctionSet, and SegmentBasisFunctionGenerator.

TensionBasisHat

TensionBasisHat implements the common basis hat function that forms the basis for all B Splines. It contains the left/right ordinates, the tension, and the normalizer.

TensionProcessedBasisHat

TensionProcessedBasisHat implements the processed hat basis function of the form laid out in the basic framework outlined in Koch and Lyche (1989), Koch and Lyche (1993), and Kvasov (2000).

BasisHatShapeControl

BasisHatShapeControl implements the shape control function for the hat basis set as laid out in the framework outlined in Koch and Lyche (1989), Koch and Lyche (1993), and Kvasov (2000). Currently BasisHatShapeControl implements the following shape control customizers:

- Cubic Polynomial with Rational Linear Shape Controller
- Cubic Polynomial with Rational Quadratic Shape Controller
- Cubic Polynomial with Rational Exponential Shape Controller

LeftHatShapeControl

LeftHatShapeControl implements the BasisHatShapeControl interface for the left hat basis set as laid out in the basic framework outlined in Koch and Lyche (1989), Koch and Lyche (1993), and Kvasov (2000).

RightHatShapeControl

RightHatShapeControl implements the BasisHatShapeControl interface for the right hat basis set as laid out in the basic framework outlined in Koch and Lyche (1989), Koch and Lyche (1993), and Kvasov (2000).

CubicRationalLeftRaw

CubicRationalLeftRaw implements the TensionBasisHat interface in accordance with the raw left cubic rational hat basis function laid out in the basic framework outlined in Koch and Lyche (1989), Koch and Lyche (1993), and Kvasov (2000).

CubicRationalRightRaw

CubicRationalRightRaw implements the TensionBasisHat interface in accordance with the raw right cubic rational hat basis function laid out in the basic framework outlined in Koch and Lyche (1989), Koch and Lyche (1993), and Kvasov (2000).

ExponentialTensionLeftHat

ExponentialTensionLeftHat implements the TensionBasisHat interface in accordance with the left exponential hat basis function laid out in the basic framework outlined in Koch and Lyche (1989), Koch and Lyche (1993), and Kvasov (2000).

ExponentialTensionRightHat

ExponentialTensionRightHat implements the TensionBasisHat interface in accordance with the right exponential hat basis function laid out in the basic framework outlined in Koch and Lyche (1989), Koch and Lyche (1993), and Kvasov (2000).

ExponentialTensionLeftRaw

ExponentialTensionLeftRaw implements the TensionBasisHat interface in accordance with the raw left exponential hat basis function laid out in the basic framework outlined in Koch and Lyche (1989), Koch and Lyche (1993), and Kvasov (2000).

ExponentialTensionRightRaw

ExponentialTensionRightRaw implements the TensionBasisHat interface in accordance with the raw right exponential hat basis function laid out in the basic framework outlined in Koch and Lyche (1989), Koch and Lyche (1993), and Kvasov (2000).

BasisHatPairGenerator

BasisHatPairGenerator implements the generation functionality behind the hat basis function pair. It provides the following functionality:

a. Generate the array of the Hyperbolic Phy and Psy Hat Function Pair
b. Generate the array of the Hyperbolic Phy and Psy Hat Function Pair From their Raw Counterparts
c. Generate the array of the Cubic Rational Phy and Psy Hat Function Pair From their Raw Counterparts
d. Generate the array of the Custom Phy and Psy Hat Function Pair From their Raw Counterparts

SegmentBasisFunction

SegmentBasisFunction is the abstract class over which the local ordered envelope functions for the B Splines are implemented. It exposes the following stubs:

- Retrieve the Order of the B Spline
- Retrieve the Leading Predictor Ordinate

- Retrieve the Following Predictor Ordinate
- Retrieve the Trailing Predictor Ordinate
- Compute the complete Envelope Integrand - this will serve as the Envelope Normalizer
- Evaluate the Cumulative Normalized Integrand up to the given ordinate

SegmentMonicBasisFunction

SegmentMonicBasisFunction implements the local monic B Spline that envelopes the predictor ordinates, and the corresponding set of ordinates/basis functions. SegmentMonicBasisFunction uses the left/right TensionBasisHat instances to achieve its implementation goals.

SegmentMulticBasisFunction

SegmentMulticBasisFunction implements the local multic B Spline that envelopes the predictor ordinates, and the corresponding set of ordinates/basis functions. SegmentMulticBasisFunction uses the left/right SegmentBasisFunction instances to achieve its implementation goals.

SegmentBasisFunctionSet

SegmentBasisFunctionSet class implements per-segment function set for B Splines and tension splines. Derived implementations expose explicit targeted basis functions.

SegmentBasisFunctionGenerator

SegmentBasisFunctionGenerator generates B Spline Functions of different order. It provides the following functionality:

- Create a Tension Monic B Spline Basis Function
- Construct a Sequence of Monic Basis Functions
- Create a sequence of B Splines of the specified order from the given inputs

Spline Builder: Tension Spline

Spline Builder Tension Spline functions are available in the package *org.drip.spline.tension*. The tension spline package implements closed form family of cubic tension splines laid out in the basic framework outlined in Koch and Lyche (1989), Koch and Lyche (1993), and Kvasov (2000).

Functionality in this package is implemented over 4 classes - KLKHyperbolicTensionPhy, KLKHyperbolicTensionPsy, KochLocheKvasovBasis, and KochLycheKvasovFamily.

KLKHyperbolicTensionPhy

KLKHyperbolicTensionPhy implements the custom evaluator, differentiator, and integrator for the KLK Tension Phy Functions outlined in the publications above.

KLKHyperbolicTensionPsy

KLKHyperbolicTensionPsy implements the custom evaluator, differentiator, and integrator for the KLK Tension Psy Functions outlined in the publications above.

KochLycheKvasovBasis

This class exposes functions that implement the monic, quadratic, and the cubic basis B Splines as outlined in the publications above.

KochLycheKvasovFamily

This class implements the basic framework and the family of C^2 Tension Splines outlined above. Functions exposed here implement the Basis Function Set from:

- Hyperbolic Hat Primitive Set
- Cubic Polynomial Numerator and Linear Rational Denominator
- Cubic Polynomial Numerator and Quadratic Rational Denominator
- Cubic Polynomial Numerator and Exponential Denominator

DRIP Samples

DRIP Samples Implementation consists of the following 8 packages:

1. DRIP MATH Samples: This illustrates some of the targeted math functionality exported in DRIP – integrand quadrature/algorithmic differentiation routines, non-linear fixed point searches, and linear algrbra modules.

2. Spline Samples: The spline sample package contains samples that demonstrate the construction and usage of different basis splines and B Spline Sequences.

3. Stretch Samples: The stretch sample package contains samples that demonstrate the construction, modification, and usage of stretches based off of different basis splines. They illustrate the computation of the curvature and the length penalties, and construction of best-fit regression spline samples. Finally they bring it all together in showing how to build latent state from measurements.

4. Bond Samples: The bond sample package contains samples that demonstrate the API to access bond static/closing fields, bond single-field analytics, and RV measures. It also illustrates usage of the bond basket API.

5. Credit Samples: The Credit Sample Package demonstrates the core credit analytics functionality – construction of credit curves, pricing of CDS and CDS basket, and retrieve the built-in pre-constructed CDX baskets and CDS closes.

6. Rates Samples: The Rates Sample Package demonstrates the core rates analytics functionality – construction of rates and forward curves (shape preserving/smoothing/transition spline variants) and pricing of rates, treasury, and rates basket products.

7. Miscellaneous Samples: Miscellaneous Samples demonstrates the set of samples not covered in the other sections – in particular the Day Count, the Calendar, and FXAPI samples.

8. Bloomberg Samples: The Bloomberg Sample Package implements the Bloomberg's calls CDSW, SWPM, and YAS.

DRIP MATH Samples

The DRIP MATH Sample functions are available in the package *org.drip.sample.quant*. This illustrates some of the targeted math functionality exported in DRIP – integrand quadrature/algorithmic differentiation routines, non-linear fixed point searches, and linear algrbra modules.

Functionality in this package is implemented over 3 classes - FixedPointSearch, IntegrandQuadrature, and LinearAlgebra.

FixedPointSearch

FixedPointSearch contains a sample illustration of usage of the Root Finder Library. It demonstrates the fixed-point extraction using the following techniques:

- Newton-Raphson method
- Bisection Method
- False Position
- Quadratic Interpolation
- Inverse Quadratic Interpolation
- Ridder's method
- Brent's method
- Zheng's method

IntegrandQuadrature

IntegrandQuadrature shows samples for the following routines for integrating the objective function:

- Mid-Point Scheme
- Trapezoidal Scheme
- Simpson/Simpson38 schemes
- Boole Scheme

LinearAlgebra

LinearAlgebra implements Samples for Linear Algebra and Matrix Manipulations. It demonstrates the following:

- Compute the inverse of a matrix, and multiply with the original to recover the unit matrix
- Solves system of linear equations using one the exposed techniques

DRIP Samples: Spline

The Spline Sample functions are available in the package *org.drip.sample.spline*. The spline sample package contains samples that demonstrate the construction and usage of different basis splines and B Spline Sequences.

Functionality in this package is implemented over 8 classes - BasisSplineSet, PolynomialBasisSpline, BasisTensionSplineSet, BasisBSplineSet, BasisMonicHatComparison, BasisMonicBSpline, BasisMulticBSpline, and BSplineSequence.

BasisSplineSet

BasisSplineSet implements Samples for the Construction and the usage of various basis spline functions. It demonstrates the following:

- Construction of segment control parameters - polynomial (regular/Bernstein) segment control, exponential/hyperbolic tension segment control, Kaklis-Pandelis tension segment control, and C^1 Hermite
- Control the segment using the rational shape controller, and the appropriate C^k
- Estimate the node value and the node value Jacobian with the segment, as well as at the boundaries
- Calculate the segment monotonicity

PolynomialBasisSpline

PolynomialBasisSpline implements Samples for the Construction and the usage of polynomial (both regular and Hermite) basis spline functions. It demonstrates the following:

- Control the polynomial segment using the rational shape controller, the appropriate C^k, and the basis function
- Demonstrate the variational shape optimization behavior
- Estimate the node value and the node value Jacobian with the segment, as well as at the boundaries
- Calculate the segment monotonicity and the curvature penalty

BasisTensionSplineSet

BasisTensionSplineSet implements Samples for the Construction and the usage of various basis spline functions. It demonstrates the following:

- Construction of Kocke-Lyche-Kvasov tension spline segment control parameters - using hyperbolic, exponential, rational linear, and rational quadratic primitives
- Control the segment using the rational shape controller, and the appropriate C^k
- Estimate the node value and the node value Jacobian with the segment, as well as at the boundaries
- Calculate the segment monotonicity

BasisBSplineSet

BasisBSplineSet implements Samples for the Construction and the usage of various basis B Spline functions.

BasisMonicHatComparison

BasisMonicHatComparison implements the comparison of the basis hat functions used in the construction of the monic basis B Splines. It demonstrates the following:

- Construction of the Linear Cubic Rational Raw Hat Functions
- Construction of the Quadratic Cubic Rational Raw Hat Functions
- Construction of the Corresponding Processed Tension Basis Hat Functions
- Construction of the Wrapping Monic Functions
- Estimation and Comparison of the Ordered Derivatives

BasisMonicBSpline

BasisMonicBSpline implements Samples for the Construction and the usage of various monic basis B Splines. It demonstrates the following:

- Construction of segment B Spline Hat Basis Functions
- Estimation of the derivatives and the basis envelope cumulative integrands
- Estimation of the normalizer and the basis envelope cumulative normalized integrand

BasisMulticBSpline

BasisMulticBSpline implements Samples for the Construction and the usage of various multic basis B Splines. It demonstrates the following:

- Construction of segment higher order B Spline Hat Basis Functions
- Estimation of the derivatives and the basis envelope cumulative integrands
- Estimation of the normalizer and the basis envelope cumulative normalized integrand

BSplineSequence

BSplineSequence implements Samples for the Construction and the usage of various monic basis B Spline Sequences. It demonstrates the following:

- Construction and Usage of segment Monic B Spline Sequence
- Construction and Usage of segment Multic B Spline Sequence

DRIP Samples: Stretch

The Stretch Sample functions are available in the package *org.drip.sample.stretch*. The stretch sample package contains samples that demonstrate the construction, modification, and usage of stretches based off of different basis splines. They illustrate the computation of the curvature and the length penalties, and construction of best fit regression spline samples. Finally they bring it all together in showing how to build latent state from measurements.

Functionality in this package is implemented over 7 classes - StretchEstimation, TensionStretchEstimation, StretchAdjuster, RegressionSplineEstimator, PenalizedCurvatureFit, PenalizedCurvatureLengthFit, and CustomCurveBuilder.

StretchEstimation

StretchEstimation demonstrates the Stretch builder and usage API. It shows the following:

- Construction of segment control parameters - polynomial (regular/Bernstein) segment control, exponential/hyperbolic tension segment control, Kaklis-Pandelis tension segment control
- Perform the following sequence of tests for a given segment control for a predictor/response range
    - Assign the array of Segment Builder Parameters - one per segment
    - Construct the Stretch Instance
    - Estimate, compute the segment-by-segment monotonicity and the Stretch Jacobian
    - Construct a new Stretch instance by inserting a pair of predictor/response knots

    - Estimate, compute the segment-by-segment monotonicity and the Stretch Jacobian
- Demonstrate the construction, the calibration, and the usage of Local Control Segment Spline
- Demonstrate the construction, the calibration, and the usage of Lagrange Polynomial Stretch
- Demonstrate the construction, the calibration, and the usage of C1 Stretch with the desired customization.

TensionStretchEstimation

TensionStretchEstimation demonstrates the Stretch builder and usage API. It shows the following:

- Construction of segment control parameters - polynomial (regular/Bernstein) segment control, exponential/hyperbolic tension segment control, Kaklis-Pandelis tension segment control
- Tension Basis Spline Test using the specified predictor/response set and the array of segment custom builder control parameters
- Complete the full tension stretch estimation sample test

StretchAdjuster

StretchAdjuster demonstrates the Stretch Manipulation and Adjustment API. It shows the following:

- Construct a simple Base Stretch
- Clip a left Portion of the Stretch to construct a left-clipped Stretch
- Clip a right Portion of the Stretch to construct a tight-clipped Stretch

- Compare the values across all the stretches to establish a) the continuity in the base smoothness is, preserved, and b) Continuity across the predictor ordinate for the implied response value is also preserved

RegressionSplineEstimator

RegressionSplineEstimator shows the sample construction and usage of Regression Splines. It demonstrates the construction of the segment's predictor ordinate/response value combination, and eventual calibration.

PenalizedCurvatureFit

PenalizedCurvatureFit demonstrates the setting up and the usage of the curvature and closeness of fit penalizing spline. It illustrates in detail the following steps:

- Set up the X Predictor Ordinate and the Y Response Value Set
- Construct a set of Predictor Ordinates, their Responses, and corresponding Weights to serve as weighted closeness of fit
- Construct a rational shape controller with the desired shape controller tension parameters and Global Scaling
- Construct the segment inelastic parameter that is C2, with 2nd order roughness penalty derivative, and without constraint
- Construct the base, the base + 1 degree segment builder control
- Construct the base, the elevated, and the best fit basis spline stretches
- Compute the segment-by-segment monotonicity for all the three stretches
- Compute the Stretch Jacobian for all the three stretches
- Compute the Base Stretch Curvature Penalty Estimate
- Compute the Elevated Stretch Curvature Penalty Estimate
- Compute the Best Fit Stretch Curvature Penalty Estimate

PenalizedCurvatureLengthFit

PenalizedCurvatureLengthFit demonstrates the setting up and the usage of the curvature, the length, and the closeness of fit penalizing spline. This sample shows the following:

- Set up the X Predictor Ordinate and the Y Response Value Set
- Construct a set of Predictor Ordinates, their Responses, and corresponding Weights to serve as weighted closeness of fit
- Construct a rational shape controller with the desired shape controller tension parameters and Global Scaling
- Construct the Segment Inelastic Parameter that is C2, with First Order Segment Length Penalty Derivative, Second Order Segment Curvature Penalty Derivative, their Amplitudes, and without Constraint
- Construct the base, the base + 1 degree segment builder control
- Construct the base, the elevated, and the best fit basis spline stretches
- Compute the segment-by-segment monotonicity for all the three stretches
- Compute the Stretch Jacobian for all the three stretches
- Compute the Base Stretch Curvature, Length, and the Best Fit DPE
- Compute the Elevated Stretch Curvature, Length, and the Best Fit DPE
- Compute the Best Fit Stretch Curvature, Length, and the Best Fit DPE

CustomCurveBuilder

CustomCurveBuilder contains samples that demo how to build a discount curve from purely the cash flows. It provides for elaborate curve builder control, both at the segment level and at the Stretch level. In particular, it shows the following:

- Construct a discount curve from the discount factors available purely from the cash and the euro-dollar instruments
- Construct a discount curve from the cash flows available from the swap instruments

In addition, the sample demonstrates the following ways of controlling curve construction:

- Control over the type of segment basis spline
- Control over the polynomial basis spline order C^k, and tension parameters
    - Provision of custom shape controllers (in this case rational shape controller)
- Calculation of segment monotonicity and convexity

DRIP Samples: Bond

The Bond Sample functions are available in the package *org.drip.sample.bond*. The bond sample package contains samples that demonstrate the API to access bond static/closing fields, bond single-field analytics, and RV measures. It also illustrates usage of the bond basket API.

Functionality in this package is implemented over 5 classes - BondAnalyticsAPI, BondBasketAPI, BondLiveAndEODAPI, BondRVMeasuresAPI, and BondBasketAPI.

BondAnalyticsAPI

BondAnalyticsAPI contains a demo of the bond analytics API Sample. It generates the value and the RV measures for essentially the same bond (with identical cash flows) constructed in 3 different ways:

- As a fixed rate bond
- As a floater
- As a bond constructed from a set of custom coupon and principal flows

It shows these measures reconcile where they should.

BondBasketAPI

BondBasketAPI contains a demo of the bond basket API Sample. It shows the following:

- Build the IR Curve from the Rates' instruments
- Build the Component Credit Curve from the CDS instruments
- Create the basket market parameters and add the named discount curve and the credit curves to it

- Create the bond basket from the component bonds and their weights
- Construct the Valuation and the Pricing Parameters
- Generate the bond basket measures from the valuation, the pricer, and the market parameters

BondLiveAndEODAPI

BondLiveAndEODAPI contains the comprehensive sample class demonstrating the usage of the EOD and Live Curve Bond API functions.

BondRVMeasuresAPI

BondRVMeasuresAPI is a Simple Bond RV Measures API Sample demonstrating the invocation and usage of Bond RV Measures functionality. It shows the following:

- Create the discount/treasury curve from rates/treasury instruments
- Compute the work-out date given the price
- Compute and display the base RV measures to the work-out date
- Compute and display the bumped RV measures to the work-out date

BondStaticAPI

BondStaticAPI contains a demo of the bond static API Sample. The Sample demonstrates the retrieval of the bond's static fields.

DRIP Samples: Credit

The Credit Sample functions are available in the package *org.drip.sample.credit*. The Credit Sample Package demonstrates the core credit analytics functionality – construction of credit curves, pricing of CDS and CDS basket, and retrieve the built-in pre-constructed CDX baskets and CDS closes.

Functionality in this package is implemented over 4 classes - CreditAnalyticsAPI, CDSLiveAndEODAPI, StandardCDXAPI, and CDSBasketAPI.

CreditAnalyticsAPI

CreditAnalyticsAPI contains a demo of the CDS Analytics API Sample. It illustrates the following:

- Credit Curve Creation: From flat Hazard Rate, and from an array of dates and their corresponding survival probabilities
- Create Credit Curve from CDS instruments, and recover the input measure quotes
- Create an SNAC CDS, price it, and display the coupon/loss cash flow

CDSLiveAndEODAPI

CDSLiveAndEODAPI is a fairly comprehensive sample demonstrating the usage of the EOD and Live CDS Curve API functions. It demonstrates the following:

- Retrieves all the CDS curves available for the given EOD
- Retrieves the calibrated credit curve from the CDS instruments for the given CDS curve name, IR curve name, and EOD. Also shows the 10Y survival probability and hazard rate

- Displays the CDS quotes used to construct the closing credit curve
- Loads all available credit curves for the given curve ID built from CDS instruments between 2 dates and displays the corresponding 5Y quote
- Calculate and display the EOD CDS measures for a spot starting CDS based off of a specific credit curve

StandardCDXAPI

StandardCDXAPI contains a demo of the CDS basket API Sample. It shows the following:

- Construct the CDX.NA.IG 5Y Series 17 index by name and series
- Construct the on-the-run CDX.NA.IG 5Y Series index
- List all the built-in CDX - their names and descriptions
- Construct the on-the run CDX.EM 5Y corresponding to T - 1Y
- Construct the on-the run ITRAXX.ENERGY 5Y corresponding to T - 7Y
- Retrieve the full set of date/index series set for ITRAXX.ENERGY

CDSBasketAPI

CDSBasketAPI contains a demo of the CDS basket API Sample. It shows the following:

- Build the IR Curve from the Rates' instruments
- Build the Component Credit Curve from the CDS instruments
- Create the basket market parameters and add the named discount curve and the credit curves to it
- Create the CDS basket from the component CDS and their weights
- Construct the Valuation and the Pricing Parameters
- Generate the CDS basket measures from the valuation, the pricer, and the market parameters

DRIP Samples: Rates

The Rates Sample functions are available in the package *org.drip.sample.rates*. The Rates Sample Package demonstrates the core rates analytics functionality – construction of rates and forward curves (shape preserving/smoothing/transition spline variants) and pricing of rates, treasury, and rates basket products.

Functionality in this package is implemented over 12 classes - HaganWestForwardInterpolator, ShapeDFZeroLocalSmooth, ShapePreservingDFZeroSmooth, CustomDiscountCurveBuilder, CustomDiscountCurveReconciler, DiscountCurveQuoteSensitivity, TemplatedDiscountCurveBuilder, CustomForwardCurveBuilder, RatesAnalyticsAPI, TreasuryCurveAPI, RatesLiveAndEODAPI, and MultiLegSwapAPI.

HaganWestForwardInterpolator

This sample illustrates using the Hagan and West (2006) Estimator. It provides the following functionality:

- Set up the Predictor ordinates and the response values
- Construct the rational linear shape control with the specified tension
- Create the Segment Inelastic design using the Ck and Curvature Penalty Derivatives
- Build the Array of Segment Custom Builder Control Parameters of the KLK Hyperbolic Tension Basis Type, the tension, the segment inelastic design control, and the shape controller
- Setup the monotone convex stretch using the above settings, and with no linear inference, no spurious extrema, or no monotone filtering applied
- Setup the monotone convex stretch using the above settings, and with linear inference, no spurious extrema, or no monotone filtering applied

- Compute and display the monotone convex output with the linear forward state
- Compute and display the monotone convex output with the harmonic forward state

ShapeDFZeroLocalSmooth

ShapeDFZeroLocalSmooth demonstrates the usage of different local smoothing techniques involved in the discount curve creation. It shows the following:

- Construct the Array of Cash/Swap Instruments and their Quotes from the given set of parameters
- Construct the Cash/Swap Instrument Set Stretch Builder
- Set up the Linear Curve Calibrator using the following parameters:
  - Cubic Exponential Mixture Basis Spline Set
  - $C^k = 2$, Segment Curvature Penalty = 2
  - Quadratic Rational Shape Controller
  - Natural Boundary Setting
- Set up the Akima Local Curve Control parameters as follows:
  - C^1 Akima Monotone Smoothener with spurious extrema elimination and monotone filtering applied
  - Zero Rate Quantification Metric
  - Cubic Polynomial Basis Spline Set
  - $C^k = 2$, Segment Curvature Penalty = 2
  - Quadratic Rational Shape Controller
  - Natural Boundary Setting
- Set up the Harmonic Local Curve Control parameters as follows:
  - C^1 Harmonic Monotone Smoothener with spurious extrema elimination and monotone filtering applied
  - Zero Rate Quantification Metric
  - Cubic Polynomial Basis Spline Set
  - $C^k = 2$, Segment Curvature Penalty = 2

- Quadratic Rational Shape Controller
- Natural Boundary Setting

- Set up the Hyman 1983 Local Curve Control parameters as follows:
  - C^1 Hyman 1983 Monotone Smoothener with spurious extrema elimination and monotone filtering applied
  - Zero Rate Quantification Metric
  - Cubic Polynomial Basis Spline Set
  - $C^k = 2$, Segment Curvature Penalty = 2
  - Quadratic Rational Shape Controller
  - Natural Boundary Setting
- Set up the Hyman 1989 Local Curve Control parameters as follows:
  - C^1 Akima Monotone Smoothener with spurious extrema elimination and monotone filtering applied
  - Zero Rate Quantification Metric
  - Cubic Polynomial Basis Spline Set
  - $C^k = 2$, Segment Curvature Penalty = 2
  - Quadratic Rational Shape Controller
  - Natural Boundary Setting
- Set up the Huynh-Le Floch Delimited Local Curve Control parameters as follows:
  - C^1 Huynh-Le Floch Delimited Monotone Smoothener with spurious extrema elimination and monotone filtering applied
  - Zero Rate Quantification Metric
  - Cubic Polynomial Basis Spline Set
  - $C^k = 2$, Segment Curvature Penalty = 2
  - Quadratic Rational Shape Controller
  - Natural Boundary Setting
- Set up the Kruger Local Curve Control parameters as follows:
  - C^1 Kruger Monotone Smoothener with spurious extrema elimination and monotone filtering applied
  - Zero Rate Quantification Metric

    - Cubic Polynomial Basis Spline Set
    - $C^k = 2$, Segment Curvature Penalty = 2
    - Quadratic Rational Shape Controller
    - Natural Boundary Setting
- Construct the Shape Preserving Discount Curve by applying the linear curve calibrator to the array of Cash and Swap Stretches
- Construct the Akima Locally Smoothened Discount Curve by applying the linear curve calibrator and the Local Curve Control parameters to the array of Cash and Swap Stretches and the shape-preserving discount curve
- Construct the Harmonic Locally Smoothened Discount Curve by applying the linear curve calibrator and the Local Curve Control parameters to the array of Cash and Swap Stretches and the shape preserving discount curve
- Construct the Hyman 1983 Locally Smoothened Discount Curve by applying the linear curve calibrator and the Local Curve Control parameters to the array of Cash and Swap Stretches and the shape preserving discount curve
- Construct the Hyman 1989 Locally Smoothened Discount Curve by applying the linear curve calibrator and the Local Curve Control parameters to the array of Cash and Swap Stretches and the shape preserving discount curve
- Construct the Huynh-Le Floch Delimiter Locally Smoothened Discount Curve by applying the linear curve calibrator and the Local Curve Control parameters to the array of Cash and Swap Stretches and the shape preserving discount curve
- Construct the Kruger Locally Smoothened Discount Curve by applying the linear curve calibrator and the Local Curve Control parameters to the array of Cash and Swap Stretches and the shape preserving discount curve
- Cross-Comparison of the Cash/Swap Calibration Instrument "Rate" metric across the different curve construction methodologies
- Cross-Comparison of the Swap Calibration Instrument "Rate" metric across the different curve construction methodologies for a sequence of bespoke swap instruments

ShapePreservingDFZeroSmooth

ShapePreservingDFZeroSmooth demonstrates the usage of different shape preserving and smoothing techniques involved in the discount curve creation. It shows the following:

- Construct the Array of Cash/Swap Instruments and their Quotes from the given set of parameters
- Construct the Cash/Swap Instrument Set Stretch Builder
- Set up the Linear Curve Calibrator using the following parameters:
  - Cubic Exponential Mixture Basis Spline Set
  - $C^k = 2$, Segment Curvature Penalty = 2
  - Quadratic Rational Shape Controller
  - Natural Boundary Setting
- Set up the Global Curve Control parameters as follows:
  - Zero Rate Quantification Metric
  - Cubic Polynomial Basis Spline Set
  - $C^k = 2$, Segment Curvature Penalty = 2
  - Quadratic Rational Shape Controller
  - Natural Boundary Setting
- Set up the Local Curve Control parameters as follows:
  - C^1 Bessel Monotone Smoothener with no spurious extrema elimination and no monotone filter
  - Zero Rate Quantification Metric
  - Cubic Polynomial Basis Spline Set
  - $C^k = 2$, Segment Curvature Penalty = 2
  - Quadratic Rational Shape Controller
  - Natural Boundary Setting
- Construct the Shape Preserving Discount Curve by applying the linear curve calibrator to the array of Cash and Swap Stretches
- Construct the Globally Smoothened Discount Curve by applying the linear curve calibrator and the Global Curve Control parameters to the array of Cash and Swap Stretches and the shape preserving discount curve

- Construct the Locally Smoothened Discount Curve by applying the linear curve calibrator and the Local Curve Control parameters to the array of Cash and Swap Stretches and the shape preserving discount curve
- Cross-Comparison of the Cash/Swap Calibration Instrument "Rate" metric across the different curve construction methodologies
- Cross-Comparison of the Swap Calibration Instrument "Rate" metric across the different curve construction methodologies for a sequence of bespoke swap instruments

CustomDiscountCurveBuilder

CustomDiscountCurveBuilder discount curve calibration and input instrument calibration quote recovery. It shows the following:

- Construct the Array of Cash/Swap Instruments and their Quotes from the given set of parameters
- Construct the Cash/Swap Instrument Set Stretch Builder
- Set up the Linear Curve Calibrator using the following parameters:
    - Cubic Exponential Mixture Basis Spline Set
    - $C^k = 2$, Segment Curvature Penalty = 2
    - Quadratic Rational Shape Controller
    - Natural Boundary Setting
- Construct the Shape Preserving Discount Curve by applying the linear curve calibrator to the array of Cash and Swap Stretches
- Cross-Comparison of the Cash/Swap Calibration Instrument "Rate" metric across the different curve construction methodologies

CustomDiscountCurveReconciler

CustomDiscountCurveReconciler demonstrates the multi-stretch transition custom discount curve construction, turns application, discount factor extraction, and calibration quote recovery. It shows the following steps:

- Setup the linear curve calibrator
- Setup the cash instruments and their quotes for calibration
- Setup the cash instruments stretch latent state representation - this uses the discount factor quantification metric and the "rate" manifest measure
- Setup the swap instruments and their quotes for calibration
- Setup the swap instruments stretch latent state representation - this uses the discount factor quantification metric and the "rate" manifest measure
- Calibrate over the instrument set to generate a new overlapping latent state span instance
- Retrieve the "cash" stretch from the span
- Retrieve the "swap" stretch from the span
- Create a discount curve instance by converting the overlapping stretch to an exclusive non-overlapping stretch
- Compare the discount factors and their monotonicity emitted from the discount curve, the non-overlapping span, and the "swap" stretch across the range of tenor predictor ordinates
- Cross-Recovery of the Cash Calibration Instrument "Rate" metric across the different curve construction methodologies
- Cross-Recovery of the Swap Calibration Instrument "Rate" metric across the different curve construction methodologies
- Create a turn list instance and add new turn instances
- Update the discount curve with the turn list
- Compare the discount factor implied the discount curve with and without applying the turns adjustment

DiscountCurveQuoteSensitivity

DiscountCurveQuoteSensitivity demonstrates the calculation of the discount curve sensitivity to the calibration instrument quotes. It does the following:

- Construct the Array of Cash/Swap Instruments and their Quotes from the given set of parameters
- Construct the Cash/Swap Instrument Set Stretch Builder.
- Set up the Linear Curve Calibrator using the following parameters:
  - Cubic Exponential Mixture Basis Spline Set
  - $C^k = 2$, Segment Curvature Penalty = 2
  - Quadratic Rational Shape Controller
  - Natural Boundary Setting
- Construct the Shape Preserving Discount Curve by applying the linear curve calibrator to the array of Cash and Swap Stretches
- Cross-Comparison of the Cash/Swap Calibration Instrument "Rate" metric across the different curve construction methodologies
- Display of the Cash Instrument Discount Factor Quote Jacobian Sensitivities
- Display of the Swap Instrument Discount Factor Quote Jacobian Sensitivities

TemplatedDiscountCurveBuilder

TemplatedDiscountCurveBuilder sample demonstrates the usage of the different pre-built Discount Curve Builders. It shows the following:

- Construct the Array of Cash Instruments and their Quotes from the given set of parameters
- Construct the Array of Swap Instruments and their Quotes from the given set of parameters
- Construct the Cubic Tension KLK Hyperbolic Discount Factor Shape Preserver
- Construct the Cubic Tension KLK Hyperbolic Discount Factor Shape Preserver with Zero Rate Smoothening applied
- Construct the Cubic Polynomial Discount Factor Shape Preserver

- Construct the Cubic Polynomial Discount Factor Shape Preserver with Zero Rate Smoothening applied
- Construct the Discount Curve using the Bear Sterns' DENSE Methodology
- Construct the Discount Curve using the Bear Sterns' DUALDENSE Methodology
- Cross-Comparison of the Cash Calibration Instrument "Rate" metric across the different curve construction methodologies
- Cross-Comparison of the Swap Calibration Instrument "Rate" metric across the different curve construction methodologies
- Cross-Comparison of the generated Discount Factor across the different curve construction Methodologies for different node points

CustomForwardCurveBuilder

CustomForwardCurveBuilder contains the sample demonstrating the full functionality behind creating highly customized spline based forward curves.
The first sample illustrates the creation and usage of the xM-6M Tenor Basis Swap:

- Construct the 6M-xM float-float basis swap
- Calculate the corresponding starting forward rate off of the discount curve
- Construct the shape preserving forward curve off of Cubic Polynomial Basis Spline
- Construct the shape preserving forward curve off of Quartic Polynomial Basis Spline
- Construct the shape preserving forward curve off of Hyperbolic Tension Based Basis Spline
- Set the discount curve based component market parameters
- Set the discount curve + cubic polynomial forward curve based component market parameters
- Set the discount curve + quartic polynomial forward curve based component market parameters
- Set the discount curve + hyperbolic tension forward curve based component market parameters

- Compute the following forward curve metrics for each of cubic polynomial forward, quartic polynomial forward, and KLK Hyperbolic tension forward curves:
    - Reference Basis Par Spread
    - Derived Basis Par Spread
- Compare these with a) the forward rate off of the discount curve, b) The LIBOR rate, and c) The Input Basis Swap Quote

The second sample illustrates how to build and test the forward curves across various tenor basis. It shows the following steps:

- Construct the Discount Curve using its instruments and quotes
- Build and run the sampling for the 1M-6M Tenor Basis Swap from its instruments and quotes
- Build and run the sampling for the 3M-6M Tenor Basis Swap from its instruments and quotes
- Build and run the sampling for the 6M-6M Tenor Basis Swap from its instruments and quotes
- Build and run the sampling for the 12M-6M Tenor Basis Swap from its instruments and quotes

RatesAnalyticsAPI

RatesAnalyticsAPI contains a demo of the Rates Analytics API Usage. It shows the following:

- Build a discount curve using: cash instruments only, EDF instruments only, IRS instruments only, or all of them strung together
- Re-calculate the component input measure quotes from the calibrated discount curve object
- Compute the PVDF Wengert Jacobian across all the instruments used in the curve construction

TreasuryCurveAPI

TreasuryCurveAPI contains a demo of construction and usage of the treasury discount curve from government bond inputs. It shows the following:

- Create on-the-run TSY bond set
- Calibrate a discount curve off of the on-the-run yields and calculate the implied zeroes and DF's
- Price an off-the-run TSY

RatesLiveAndEODAPI

RatesLiveAndEODAPI contains the sample API demonstrating the usage of the Rates Live and EOD functions. It does the following:

- Pulls all the closing rates curve names (of any type, incl. TSY) that exist for a given date
- Load the full IR curve created from all the single currency rate quotes (except TSY) for the given currency and date
- Calculate the discount factor to an arbitrary date using the constructed curve
- Retrieve the components and their quotes that went into constructing the curve, and display them
- Load all the rates curves available between the dates for the currency specified, and step through
- Load all the Cash quotes available between the dates for the currency specified, and step through
- Load all the EDF quotes available between the dates for the currency specified, and step through
- Load all the IRS quotes available between the dates for the currency specified, and step through

- Load all the TSY quotes available between the dates for the currency specified, and step through

MultiLegSwapAPI

MultiLegSwapAPI illustrates the creation, invocation, and usage of the MultiLegSwap. It shows how to:

- Create the Discount Curve from the rates instruments
- Set up the valuation and the market parameters
- Create the Rates Basket from the fixed/float streams
- Value the Rates Basket

DRIP Samples: Miscellaneous

The Miscellaneous Sample functions are available in the package *org.drip.sample.misc*. Miscellaneous Samples demonstrates the set of samples not covered in the other sections – in particular the Day Count, the Calendar, and FXAPI samples.

Functionality in this package is implemented over 2 classes - DayCountAndCalendarAPI and FXAPI.

DayCountAndCalendarAPI

DayCountAndCalendarAPI demonstrates Day-count and Calendar API FUnctionality. It does the following:

- Get all the holiday locations in CreditAnalytics, and all the holidays in the year according the calendar set
- Get all the week day/weekend holidays in the year according the calendar set
- Calculate year fraction between 2 dates according to semi-annual, Act/360, and USD calendar
- Adjust the date FORWARD according to the USD calendar
- Roll to the PREVIOUS date according to the USD calendar

FXAPI

FXAPI contains a demo of the FX API Sample. It shows the following:

- Create a currency pair, FX SPot, and FX Forward
- Calculate the FX forward PIP/outright
- Calculate the DC Basis on the domestic and the foreign curves

- Create an FX curve from the spot, and the array of nodes, FX forward, as well as the PIP indicator
- Calculate the array of the domestic/foreign basis
- Calculate the array of bootstrapped domestic/foreign basis
- Re-imply the array of FX Forward from domestic/foreign Basis Curve

DRIP Samples: Bloomberg

The Bloomberg Sample functions are available in the package *org.drip.sample.bloomberg*. The Bloomberg Sample Package implements the Bloomberg's calls CDSW, SWPM, and YAS.

Functionality in this package is implemented over 3 classes - CDSW, SWPM, and YAS.

CDSW

CDSW replicates Bloomberg's CDSW functionality.

SWPM

SWPM replicates Bloomberg's SWPM functionality.

YAS

YAS replicates Bloomberg's YAS functionality.

Installation and Deployment Notes

Installation is really simple just drop of each of the jars (CreditAnalytics, CreditProduct, CurveBuilder, FixedPointFinder, RegressionSuite, and SplineLibrary) - or the common DRIP jar - in the class-path.

Configuration is done off of the configuration files corresponding to each of the libraries. For most typical set-ups, the standard configuration should suffice. Please consult the configuration documentation on each of the libraries to configure each of the modules.

Because there is no other dependency, deployment should also be straightforward. Use the regression output as a guide for module capacity estimation.