OpenGamma Documentation

Sensitivity computation

Marc Henrard
marc@opengamma.com

OpenGamma Documentation n. 23 | Version 1.1 - June 2014

Abstract

In this documentation we describe the way sensitivities to curves and other parameters are computed. In particular we describe rate deltas or Bucketed PV01.

Contents

1 Introduction

In this documentation we describe the way the computation of sensitivities to curves and other parameters is performed in OpenGamma libraries. In particular we describe *rate deltas* (also known as *Bucketed PV01*) to market quotes .

As a one line summary, we can say that Algorithmic Differentiation (AD) is used systematically throughout the OpenGamma's implementation for fast and precise sensitivity computation. The implementation is also done in a flexible way which allows to obtain results that cannot be obtained by a simple "bump and recompute" approach.

The AD implementation can be viewed as an internal cuisine on the way to write the code and irrelevant to the user. This is partly true: the final user wants to see (correct) numbers, how the code that produces them is written is mainly irrelevant. It becomes relevant if the structure of the code has a significant computation time impact or if it gives the user a larger set of results. As this is the case for the implementation we describe, we will spend some space in this documentation describing the impact of the internal organisation on users.

The description of the internal workings of the implementation for developers (not for end users) is available in the documentation Quantitative Research (2014).

2 Bucketed PV01 and similar results

Interest rate deltas roughly describe the first order sensitivity of present values (or other results) to the interest rate curves. In OpenGamma's OG-Analytics, three levels of those sensitivities are available:

Point This is the sensitivity with respect to the zero-coupon continuously compounded rates at each point (each date) for which a discount factor is used and the sensitivity with respect to the forward Ibor rate at each point a forward Ibor rate is used. The information is available for each point, each cash flow, but is of a rather theoretical nature.

One case where the point sensitivity can be important in practice is for analysis of the fixing sensitivity. The sensitivity to each fixing (usually important for the one in the coming days) can potentially be displayed to users.

Internal parameters This is the sensitivity with respect to the internal parameters used to represent the curve. Often the curves are represented by zero-coupon rates at given nodes and interpolated between the nodes. In that case, the sensitivity provided in this case is the sensitivity to the node zero-coupon rates. If the curve is represented by a parametrised curve (like a Nelson-Siegel curve), the sensitivities obtained by this mechanism can be relatively difficult to interpret.

Market quotes In this result, the sensitivity with respect to the market quotes used in the curve calibration is computed. This is the most commonly used result in practice. It allows us to obtain the first order hedge using the instrument used for the curve calibration.

A variant of the *market quote* sensitivity is also available, it is called the *market rate* sensitivity. In that case, the curve calibration is not done with respect to the standard market quote of the instruments but with respect to a rate (or yield) like number. This will impact STIR

[0]First version: 5 June 2014; this version: 1.1 - 20 June 2014.

futures where the price is replaced by the future rate and bonds where the quoted (clean) price is replaced by the conventional yield.

The results above are obtained with a usual AD approach; the different partial derivatives are computed and then composed (multiplied). The *point* sensitivity is the first one computed. For each cash flow the sensitivity to the zero-coupon rate corresponding to a discount factor and to the different forward rates are recorded. Each discount factor and each forward is used but the way the numbers are computed (interpolated curve or other) is irrelevant at this stage.

In the next step, the point sensitivities are "projected" to the curve parameters. The meaning of projection obviously depends on the internal representation of the curve and its internal parameters. The projection takes into account the exact nature of the curve description and in particular of the interpolation mechanism if relevant; it is not a generic cash flow mapping but a specific mechanism dependent of the curve's internal representation. This representation can be useful in practice if all curves have a similar representation (like zero-coupon rate interpolated curves); but in general as the parameters can be of very different nature it would be incoherent to compare those figures. For the particular case where the curves are represented by zero-coupon rates the total sensitivity, called `PV01`, is provided.

In the third step, the internal sensitivities to the parameters are multiplied by the relevant Jacobian matrices to obtain the market quote sensitivities. The Jacobian matrices are computed together with the curve in the calibration process. The market quote sensitivities are thus obtained with a very low computation cost.

The AD approach is also used for the curve calibration and Jacobian matrix computation itself. The curve is calibrated on some quantity (market quote or market rate). The quantity computation and its AD version are implemented in the library. The root finding process solved to calibrate the curves is solved numerically using a recursive algorithm of Newton type. At each step, the Jacobian matrix of the curve parameters to market quote transformation is computed (and inverted). As described below in the Computation Time section, the computation of the Jacobian matrix is also used internally in the root-finding algorithm of the calibration. Computing those internal Jacobians with AD saves some computation time. In total, calibrating the curve and computing the final Jacobian with AD is faster than only calibrating the curves without AD.

That transition matrix is computed using AD on the parameter to market quote function. In the simplest case, the transition matrix (once inverted) is also the Jacobian matrix. When the calibration process is done in several units (i.e. when curves are calibrated consecutively and not simultaneously), the sensitivity to the previous curve's market quotes also needs to be recorded. The theoretical framework behind keeping track of the different sensitivities (using implicit function theorem) is described in the document Henrard (2013) and repeated in (Henrard, 2014, Section 8.3).

Note that the two methods (AD and bump and recompute) can be used in conjunction and are not mutually exclusive. For example the AD Jacobian matrix can be computed for the curve calibration process and then combined with bump and recompute for specific instruments. If AD has not been implemented for a specific instrument, one can compute the internal parameter's sensitivity using bump and recompute (at the parameter level) and multiply those sensitivities by the Jacobian matrix. The AD approach is flexible enough to accept any type of derivative computation (adjoint AD, forward AD, finite difference) for each partial derivative computation.

3 Curve calibration and Jacobian matrix computation

This section highlight how the curve are calibrated and how the Jacobian or transition matrix are computed.

3.1 What to calibrate?

The general goal of the curve calibration is to ensure that the market instruments used in the calibration are "at market level" when priced by the calibrated curves. The "market level" can be seen from several points of view. The standard ways to measure market level are *present value*, *par rate* or *par market quote*. By quote we mean any number that is used in the market to indicate the level of a financial instrument: rate, price, yield, clean price, etc.

The calibration constraint used in the present value approach is that all instruments have a present value zero when valued with the calibrated curves. This is a general approach working in all cases. It has nevertheless a couple of practical drawbacks. The first one is that the hedgers like to see the risk figures, one of the important byproduct of the curve calibration process, in term of sensitivity to a given change of market quote, like one basis point. If the curves are calibrated using present value, one has to compute the market quote sensitivities as an extra information. Even if one uses the present value approach internally, a par rate or equivalent approach would be required from a risk perspective. The second drawback is numerical. The curve calibration process, as described here, is done through a root-finding algorithm. The root-finding algorithm is implemented with an iterative algorithm using the derivative of the target function to the inputs, i.e. the sensitivities of the present values to the market quotes. The ratio in maturity between the shortest and longest instrument can be from 1 to 15,000 (one day to 50 years). The sensitivity matrix, which is inverted in the root-finding process, has values covering several orders of magnitude. Its inversion is numerically less stable. For those two reasons, in practice, we prefer to implement the curve calibration using the par market quote, or a variant of it, described below.

The calibration constraint used in the par rate approach is that the quotes for the instruments as computed from the curves are equal to the actual market quotes. For most instruments, this is not a problem. Nevertheless for some instruments this request is not trivial. Consider a basis swap for which the first period has already been fixed. The market quote is a spread on one leg. The market quote could be computed as the opposite of the sum of the two legs without spread divided by the present value of a basis point of the spread leg. But as the first coupon has already fixed, it is a fixed coupon, not a Ibor coupon with a spread. Implementing in a generic way the "without the spread" concept is relatively complex.

For those reasons we found it is easier to work with figures we called *par market quote spreads*, this is the number that should be added to the market quote to obtain a present value of zero. The idea of present value equal to zero is similar to the one in the first approach but achieved in a different way, through a number that has the dimension of a rate. For the basis spread mentioned above, the par market quote spread is the opposite of the sum of the two legs (including the spread) divided by the present value of a basis point of the spread leg. Whatever the coupons on each leg are – fixed, Ibor or overnight – this quantity can be computed unambiguously. We found this measure easier to work with than the par market quote and numerically more efficient in the solver than the present value. Moreover the sensitivity to the market quote spread is the same as the sensitivity to the market quote. So the matrix used internally by the root-finding scheme is the same than the matrix we want to provide to the user to compute risk figures.

3.2 Calibration

Now that we have clarified the "what to calibrate", we can move to the "how to calibrate". We deal only with exact calibration. We suppose that we have a certain number of instruments in the calibration basket and the same number of parameters in the curves. The parameters are split among the different curves. We suppose also that the multi-dimensional root finding problem defined by the constraints on each instrument and the curve parameter has one non-singular solution.

We index the instruments used in the calibration by k $(1 \leq k \leq n)$. Let $p = (p_l)_{1 \leq l \leq n}$ denote the parameters of the curves and $S = (S_k)_{1 \leq k \leq n}$ the par market quote spread functions for the k-th instrument. The problem to solve is

$$S_k(p) = 0 \quad (1 \leq k \leq n). \tag{1}$$

In general, the problem is non-linear and strongly coupled. The solution has to be obtained globally for all p as one computation block.

In some special cases, the problem can be divided in sub-problems, easier to solve numerically. The most extreme simplification is when the problem can be divided into one dimensional sub-problems. This is the case of the bootstrapping approach, where the interpolation scheme is local and the curves are not entangled. This is non-realistic case and we do not analyse it further.

A more frequent numerical simplification is when some curves are not fully entangled and the problem can be solved inductively; first calibrating some curves and then using the resulting curves as an input to calibrating the next curves. In the sequel we call *curve units* the set of curves calibrated simultaneously in one unique root-finding process. The complete set of all the related curves is called a *block*.

The standard application of this is when the curves are build one by one. More involved examples are when units of curves are first calibrated and then another unit of several curves is calibrated based on the previous units. Let u be the number of units and n_i the total number of parameters in the first i units. The steps in the induction are:

$$\begin{array}{rcll} S_k((p_l)_{0<l\leq n_1}) & = & 0 & (0 < k \leq n_1) \\ S_k((p_l)_{0<l\leq n_1}, (p_l)_{n_1<l\leq n_2}) & = & 0 & (n_1 < k \leq n_2) \\ & \vdots & & \\ S_k((p_l)_{0<l\leq n_{u-1}}, (p_l)_{n_{u-1}<l\leq n}) & = & 0 & (n_{u-1} < k \leq n) \end{array}$$

Each set of equations represents a unit.

Solving the above system of equations is not the only output of the calibration exercise. The output of the procedure should facilitate the computation of the risks measures and provide hedging tools. We now are more explicit by what we mean by that.

The calibrated curve is used to compute the present value of instruments (PV). We are interested also by the sensitivities of the present value to the market quotes q_k used in the curve building procedure:

$$\frac{\partial \, \mathrm{PV}}{\partial q_k}.$$

This is often called the *bucketed PV01*, *bucketed deltas* or *partial PV01* of the instrument.

Using the calibrated curve, it is possible to compute the derivatives of the present value to the curves parameters p. Algorithmic differentiation generally helps for that. The derivative with respect to the market quotes can be obtained through

$$\frac{\partial \,\mathrm{PV}}{\partial q_k} = \sum_{l=1}^{n} \frac{\partial \,\mathrm{PV}}{\partial p_l} \frac{\partial p_l}{\partial q_k}.$$

What is missing to compute the above result in practice is the matrix

$$D_q p = \left(\frac{\partial p_l}{\partial q_k} \right)_{1 \leq l \leq n, 1 \leq k \leq n}.$$

The figures we suggest to use for calibration are the par market quote spreads. They are the additive spreads to the market quotes, this is for q^M the market quotes and $\tilde{q}$ the computed quotes from the curves, $S(p, q^M) = \tilde{q}(p) - q^M$. We rewrite Equation (1) to indicate explicitly the market quote dependency. The calibrated parameters p_0 are such that

$$S(p_0, q^M) = 0.$$

Provided that some regularity conditions are satisfied around the solution (p_0, q^M), one can apply the implicit function theorem. There is a function $p : \mathbb{R}^n \to \mathbb{R}^n ; q \mapsto p(q)$ defined in a neighbourhood of q^M such that

$$S(p(q), q) = 0.$$

Using the implicit function theorem, the derivatives with respect to the quotes q can be obtained by

$$D_q p = -(D_p S)^{-1} D_q S = (D_p S)^{-1}$$

The last equality is obtained using the fact that the derivative of S to the quote q^M is -1.

When the root-finding problem is solved with a Newton-like algorithm, the same type of matrix is computed. The Jacobian matrix is used to find the direction of the next best guess. Computing the matrix for later use does not require developments beyond those already done for solving efficiently the root-finding algorithm.

We call the *Jacobian* or *transition* matrix the matrix $D_p S$ and the inverse Jacobian the matrix $D_q p = (D_p S)^{-1}$. Once the curve calibration process is finished, we store the inverse Jacobian matrix, at least the non-zero blocks of it, for later use.

4 Advantages of Algorithmic Differentiation

The two main advantages of algorithmic differentiation for the implementation of bucketed PV01 and similar risk measures are stability and speed. (Henrard, 2014, Appendix C.3) details some applications of algorithmic differentiation to the case of the multi-curve framework.

4.1 Stability

By definition the limit of the differentiation ratio is equal to the derivative in theory. This is the base of the “bump and recompute approach”. In practice, in a finite precision arithmetic, this is not true. When decreasing the bump size, the machine precision is reached and the results diverge. This stems from the fact that the limit is of the style “0/0”. For standard implementations, the level at which the divergence starts is around 1.0E-10. This is the best case, when the process to compute f does not require heavy numerical procedures. The user is squeezed between taking a small bump to obtain the theoretical convergence and a larger one in order to avoid the numerical instability problem.

An example of instability for a small increment is proposed in Figure 1. This particular example is a swap present value in the multi-curve framework. Any pricing will present a similar picture. The error decreases when the bump size is decreased, up to a certain point where the numerical instability kicks in and the number becomes meaningless. The level where the numerical instability kicks in will depend on the algorithm used in the computation and is almost impossible to guess in practice.

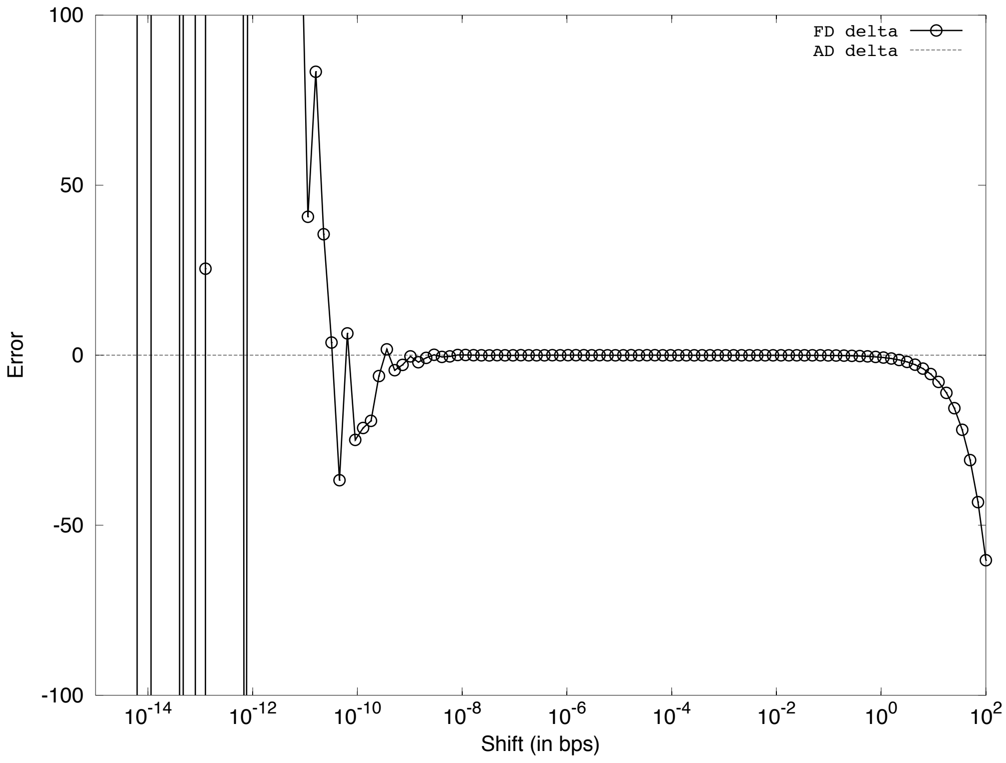

Figure 1: Instability of finite difference computation of derivative for a swap. The horizontal axis represents the bump size in logarithmic scale. The vertical axis is the error. The true value is represented by the horizontal dotted line.

A similar picture for a swaption in the Hull-White model is proposed in Figure 2.

4.2 Computation time

The first example of computation efficiency relates to an indirect application of algorithmic differentiation. In the curve calibration, we use a multi-dimensional root-finding algorithm. The implementation uses the derivative of the function to iterate the point in the root-finding algorithm. The derivative can be computed through finite difference or explicitly by algorithmic differentiation. We compare the finite difference performance to the adjoint algorithmic differentiation

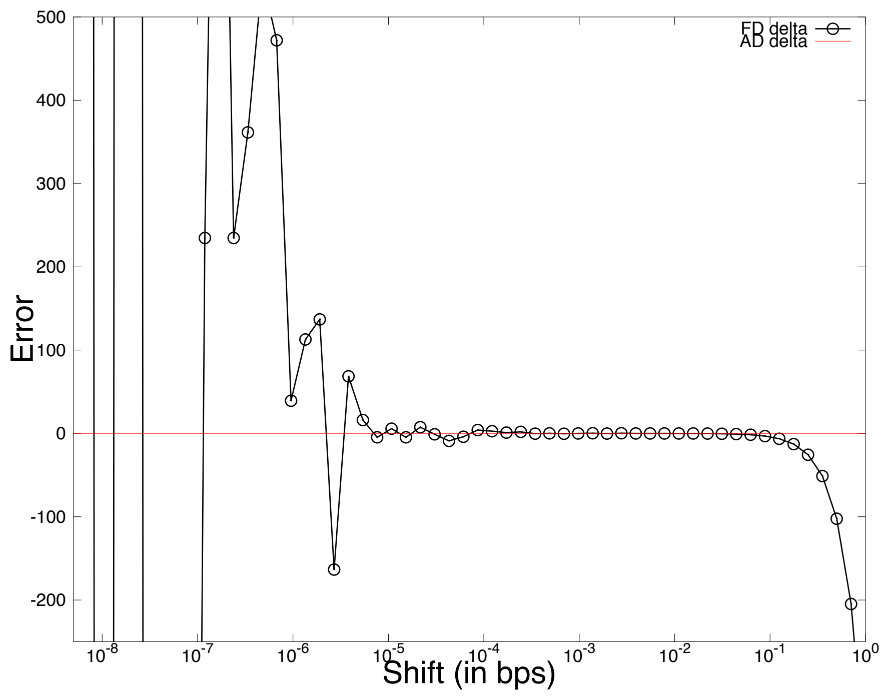

Figure 2: Instability of finite difference computation of derivative for a swaption in Hull-White model. The horizontal axis represents the bump size in logarithmic scale. The vertical axis is the error. The true value is represented by the horizontal line.

performance.

The example refers to the building of three curves in EUR (discounting, forward three months and forward six months). We use OIS, fixed versus three months IRS and fixed versus six months IRS. The choice of instruments is done in such a way that the curves are not entangled and can be built one by one; no simultaneous building is required. With that choice, we can analyse two items: the interest of having a flexible implementation allowing for simultaneous or successive curve building and the impact of algorithmic differentiation. Note that the algorithmic differentiation impact is indirect as the derivative of the objective function is only a small part of the computation.

The example uses three curves with 20 instruments each. The results are presented in Table 1. When three units are used the ratio between the version without algorithmic differentiation and the version with algorithmic differentiation is 2.05; when the three curves are built simultaneously, the ratio is 3.28. Note that the difference of performance comes for the Jacobian matrix computation only; the root-finding algorithm is the same in both case. The difference is that in one case (AD), the Jacobian is explicit while in the other case (FD), the Jacobian is computed by bump and recompute. In the most complex problems, the algorithmic differentiation, even if used indirectly, has a real impact.

The interest of the possibility to split the problem into several units is assessed in the last

	Number of units		
Algorithmic differentiation	1	3	Ratio
No	109.1	41.5	2.63
Yes	33.3	20.2	1.65
Ratio	3.28	2.05	5.40

Time in milliseconds.

Table 1: Time to build curves with and without algorithmic differentiation.

column. The ratio between one unit and three is 1.65 with algorithmic differentiation and 2.63 without. The computation of the derivative being very efficient and only slightly impacted by the number of inputs, the difference is not huge but far from trivial. In the absence of efficient derivative computation, the ratio is significantly larger (2.63). Finally the last ratio is between the number with one unit and no algorithmic differentiation and with three units and algorithmic differentiation. The combination of those two numerical improvements reduces the computation time more than fivefold.

The computation time impact of AD can be seen directly through the computation time of different sensitivities. In Table 2, different computation times are provided. The example is a swap in a simple two-curve environment (one discounting and one forward). There are a total of 30 nodes (15 on each curve). The present value of 10,000 swaps is computed in 225 milliseconds (ms). For comparison, the construction of the swaps (in particular their dates) took 1,000 ms. Computing the point sensitivity (see Section 2 for the definition) takes 950 ms in AD and is impossible in bump and recompute. Computing the sensitivity to internal parameters takes 1.325 s in AD and 6.975 s (for asymmetrical difference, the double for symmetrical difference). The gain is roughly a factor 5. The gain is more important for more complex situations where more curves are involved, in particular for multi-currency curve calibration process.

Approach	Construction	Present Value	PV + Point	PV + Internal par.
AD	1,000	225	950	1,325
Bump & recompute	1,000	225	NA	$31 \times 225 = 6{,}975$

Time in milliseconds; time for 10,000 swaps.

Table 2: Time to present value and to compute sensitivities of swaps (2 curves with 15 points each).

4.3 Comparison with finite difference

In this section we give a few examples of comparison between the AD and bump and recompute approach in terms of computed numbers. To perform the comparison we use OpenGamma's scenario generation tool. We generate scenarios which correspond simply to shift each market data consecutively by one basis point. For each market data, we compute three present values: the reference one, the one with the market rate moved up by one basis point and the one with the market rate moved down by one basis point.

The scenario used for the one-sided bump and recompute is summarised in Listing 1.

Listing 1: Groovy script for point bump and recompute.

```
view {
  name 'Swap Desk USD 1 / pv / ois-irs'
  server 'devsvr-lx-6:8080'
  marketData {
    snapshot 'Quant USD ON-OIS LIBOR3M-FRAIRS Unstructured 2014-03-17'
  }
}
shockGrid {
  ticker = [
      'base scenario',
      'US0003M Index',
      'USFR0CF Curncy',
      ...
      'USSO10 Curncy']
  shock = [1.bp]
}
scenarios {
  valuationTime '2014-03-17 11:50'
  marketData {
    id 'BLOOMBERG_TICKER', ticker
    apply {
      shift Absolute, shock
    }
  }
}
```

This type of scenario implementation of the bump and recompute guarantees that there is a total code independence between the AD version and the bump and recompute version used in this test.

We have created a portfolio of STIR futures, Fed Fund futures, FRA, OIS and Libor IRS, each with 100 millions notional. The maximum difference between the AD numbers and the bump and recompute numbers is displayed in Table 3. We use the symmetrical and one-sided difference. The numbers in the table are the maximum of the absolute difference for all nodes (total of 30 nodes on two curves). The numbers include all the numerical instability: calibration procedure, pricing, finite precision of doubles, etc. As can be seen from the table, using a one basis point shift and symmetrical difference (bump up and bump down), the error is of the order of magnitude of a hundred of a cent for one basis point bump. In practice the difference will be invisible in term of actual numbers displayed.

We display also the details for a 10 year IRS (not ATM and aged by a couple of month, fixed versus USD LIBOR 3M) in Table 4. Other examples, including with large portfolios, can be produce using the above script.

5 Workout example

In this section we create a very simple example of swap pricing using a unique curve with few points to show the way the results are obtained in algorithmic differentiation.

Approach	STIR futures	Fed Fund fut.	FRA	OIS	IRS
Asymmetrical	0.01	0.01	0.05	1.08	0.74
Symmetrical	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$	10^{-4}	10^{-4}
Difference in USD. Finite difference bump of 1 basis point.					

Table 3: Difference between algorithmic differentiation and finite difference bump and recompute for different instrument types.

USD Discounting				USD Libor 3M			
Tenor	AD	FD	FD-AD	Tenor	AD	FD	FD-AD
ON	-0.87	-0.87	0.00	Fixing 3M	1,526.72	1,526.74	0.02
TN	-0.87	-0.87	0.00	FRA 6M	0.01	0.04	0.03
OIS 1M	-5.24	-5.24	0.00	FRA 9M	-0.03	0.02	0.05
OIS 2M	-17.41	-17.41	0.00	IRS 1Y	-58.54	-58.26	0.29
OIS 3M	1.69	1.69	0.00	IRS 2Y	137.30	137.49	0.19
OIS 6M	26.72	26.72	0.00	IRS 3Y	-67.36	-66.94	0.42
OIS 9M	-52.11	-52.11	0.00	IRS 4Y	11.09	11.83	0.74
OIS 1Y	4.40	4.40	0.00	IRS 5Y	-80.75	-80.12	0.64
OIS 2Y	-44.08	-44.09	0.00	IRS 7Y	-9,895.54	-9,895.13	0.42
OIS 3Y	-75.47	-75.48	-0.01	IRS 10Y	-78,371.53	-78,371.37	0.16
OIS 4Y	-103.48	-103.50	-0.02	IRS 12Y	0.00	0.00	0.00
OIS 5Y	-130.65	-130.68	-0.03	IRS 15Y	0.00	0.00	0.00
OIS 6Y	-165.20	-165.24	-0.04	IRS 20Y	0.00	0.00	0.00
OIS 7Y	-100.63	-100.68	-0.05	IRS 25Y	0.00	0.00	0.00
OIS 8Y	-7.91	-7.93	-0.02	IRS 30Y	0.00	0.00	0.00
OIS 9Y	-75.54	-75.59	-0.05				
OIS 10Y	-477.23	-477.36	-0.13				
Difference in USD. Asymmetrical finite difference bump of 1 basis point.							

Table 4: Difference between algorithmic differentiation and finite difference bump and recompute for 10Y IRS.

The results are described in a spreadsheet, called OpenGamma-QuantDoc-23-ad-workout.xlsx, and the corresponding code is available in the OG-Analytics library in the unit test workoutADExample in the class SwapCalculatorTest.

The sheet is divided in three parts: curve description, swap description and results comparison.

The curve is a simplified single curve with only four nodes and rounded zero-coupon rates. The rates are interpolated linearly. Cell `I1` contains the shift used for the bump and recompute computations.

The swap is a USD forward starting swap starting in 6 month and with a tenor of 1 year with standard USD conventions; there are 2 semi-annual fixed coupons and 4 quarterly Libor coupons. The description of the swap has the fixing and payment times and the accrual factors.

For the fixed leg comparison, the bump and recompute numbers are available in rows 24 and 25. the present value is computed in column D. The next four columns (E-H) include the present value for each of the nodes bumped by one basis point. The columns I–L are the delta using the bumped present values. The AD results are available in rows 29 and 30. The point sensitivity is provided in column A, and the projection on the node sensitivities in the columns I to L. The comparison between AD and finite difference (FD) is in row 32. For a 100 million notional and a 1 basis point shift, the total difference is around USD 0.01.

The Libor leg comparison is a little bit longer as we have to compute the forward rate and then discount the forward rate to today. There is a total of three discount factors involved for each cash flow. The finite difference computation are done in lines 36 to 46 with the delta provided in line 46, columns I to L. The AD computations are also more involved as each node is potentially impacted by three rates for each cash flow. The AD results are available in lines 49 to 52. The comparison between AD and FD is in line 54. For a 100 million notional and a 1 basis point shift, the total difference is around USD 0.45.

6 Implementation

The detailed implementation of the different objects in the OG-analytics library is available in the specific technical documentation Quantitative Research (2014).

References

Henrard, M. (2013). Multi-curve Framework with Collateral. Quantitative Research 13, OpenGamma. Available at docs.opengamma.com. 2

Henrard, M. (2014). *Interest Rate Modelling in the Multi-curve Framework*. Applied Quantitative Finance. Palgrave Macmillan. ISBN: 978-1-137-37465-3. 2, 5

Quantitative Research (2014). Multi-curves implementation: Providers, calculators and sensitivities. Technical Documentation 5, OpenGamma. Version 1.0. 1, 11

OpenGamma Analytics Documentation

About OpenGamma

OpenGamma helps financial services firms unify their calculation of analytics across the traditional trading and risk management boundaries.

The company's flagship product, the OpenGamma Platform, is a transparent system for front-office and risk calculations for financial services firms. It combines data management, a declarative calculation engine, and analytics in one comprehensive solution. OpenGamma also develops a modern, independently-written quantitative finance library that can be used either as part of the Platform, or separately in its own right.

Released under the open source Apache License 2.0, the OpenGamma Platform covers a range of asset classes and provides a comprehensive set of analytic measures and numerical techniques.

Find out more about OpenGamma
`www.opengamma.com`

Download the OpenGamma Platform
`developers.opengamma.com/downloads`

Europe
OpenGamma
185 Park Street
London SE1 9BL
United Kingdom

North America
OpenGamma
125 Park Avenue
25th Floor, Suite 2525
New York, NY 10017
United States of America

OpenGamma