Algorithmic Sensitivity Generation in Credit Analytics

Lakshmi Krishnamurthy

v0.34, 8 Mar 2013

Glossary

1. Wengert List: List of all the non over-writable program variables (Wengert (1964)) – can also be seen as a linearization of the computational graph. By construction, it is an intermediate variable.
2. Intermediate Wengert Canonical Variable: These are intermediate financial variables those are fixed from the point-of-view of the output Jacobians and the input parameters that serve as computation graph parsimonious optimizers (Figures 1 and 2).
3. Wengert fan-in and fan-out: Reduction of a set of initial/intermediate Wengert variates onto the subsequent set is called fan-in; the opposite is fan-out.
4. Wengert funneling: Same as Wengert fan-in.
5. Micro-Jacobian: Change in the calibrated instrument measure coefficients to unit change in the quoted instrument measures.
6. Self-Jacobian: Self-Jacobian refers to the Jacobian of the Objective Function at any point in the variate to the Objective Function at the segment nodes, i.e., $\frac{\partial Y(t)}{\partial Y(t_K)}$. Self-Jacobian is a type of micro-Jacobian.
7. Derivative Entity: The entity whose dynamics are determined by the evolution of a stochastic variate, and whose specific facets/measures are observable.
8. Path-wise Derivative Estimator: $\frac{\partial V}{\partial X_i(0)}$, where V is the value of the derivative, and $X_i(0)$ is the starting value for a specific stochastic variate.
9. Non-Parsimonized Parameters: Parameters that map one-to-one with the input instrument set, e.g., typical curve bootstrapping.
10. Parsimonization: Reduction of the parameter space from the input measure space.

1) Overview and Literature Review

This paper details the techniques and methodologies behind the sensitivity generation software employed in Credit Analytics. It draws heavily on key algorithmic differentiation techniques in general, and as applied to finance. It details the variation in approach implemented for the different sensitivity builder for (quasi or total) closed form versus Monte-Carlo pay-off formulations.

As a technology and practice, algorithmic differentiation techniques have been in use for a long time (for history of algorithmic differentiation, see Iri (1991)). Griewank (2000) outlines the mathematical foundations, and an elaborate survey of the methodologies, the processes, the techniques, and the implementation tools are available from Berz (1996) and Bischof, Hovland, and Norris (2005). The de-facto official algorithmic differentiation online reference is at *http://www.autodiff.org/*.

With regards to finance, algorithmic differentiation focus has been primarily on Monte-Carlo methodologies. Although path-wise optimized sensitivity generation had been employed earlier (Glasserman (2004)), Giles and Glasserman (2006) first discussed adjoint methods in path-wise sensitivity generation. Full extension to LMM based stochastic variate evolution and a corresponding exotic (in this case Bermudan) swap option evaluation (Leclerc, Liang, and Schneider (2009)), as well as to correlated defaults and their sensitivities (Capriotti and Giles (2011)). Capriotti (2011) covers automated Greek generation, but with a focus on algorithmic differentiation, and in the context of Monte-Carlo methods. Finally, algorithmic differentiation has also been applied to addressing the issue of calibration along with sensitivity generation (Schlenkirch (2011)).

As indicated above, the purpose of this paper is to describe the sensitivity generation techniques as employed in Credit Analytics Suite, mostly using algorithmic differentiation techniques. It starts with a review of the main algorithmic differentiation concepts, the caveats, and applicability – without getting into the details. It then describes

the general Monte-Carlo based stochastic financial variate evolution and sensitivity formulation as used in Credit Analytics. It then applies it to the calculation of greeks for several products, with specific samples considered. Finally, it also considers in some detail the application of algorithmic differentiation techniques to curve calibration.

2) Algorithmic Differentiation

Algorithmic differentiation is a set of techniques for transforming a program that calculates the numerical values of a function into a program that calculates numerical values for derivatives of that function with about the same accuracy and efficiency as the function values themselves (Bartholomew-Biggs, Brown, Christianson, and Dixon (2000)).

Algorithmic differentiation aims to exploit the fact that calculation of the local derivatives is always symbolic, and thus avoids typical challenges associated a) divided differences, or b) numerical differentials (*Automatic Differentiation - Wikipedia Entry*). Further, apart from the "chain rule" multiplication factor effects, the computation requires the same number of Objective Function Calculations as the original.

Owing to its usage of local symbolic derivatives, the accuracy of Algorithmic Differentiation is always better than numerical differentials. It will automatically scale to arbitrarily small variate infinitesimals, thereby have reduced errors due to bit cancellation etc: Finally, Algorithmic Differentiation does not need additional objective function evaluations for higher order derivative calculations (beyond the chain-rule issues); therefore, those are infinitesimally correct too.

Two prototypical program construction modes can be used in algorithmic differentiation to achieve differentiation code construction – the forward and the reverse mode of algorithmic differentiation.

In the forward mode, the final and the intermediate variables are expressed as a consequence of a computed forward graph, and the symbolic forward derivative graph is then derived. In effect, this is equivalent to computing the gradient of the intermediate variables to the variates or the "independent variables" and transmitting them up the graph.

In the reverse mode, the final and the intermediate variables are expressed as nodes in the computed reverse graph, and the symbolic reverse derivative graph is then derived. Effectively, this computes the gradient of the intermediate variables to the "dependent variables" and transmits them down the graph. Often the reverse mode may need the forward path to store the calculated intermediates needed on the way back.

As can be seen, the run-time of a forward mode algorithmic differentiation is proportional to the number of variates or independent variables, and that for the reverse mode is proportional to the number of dependent variables.

The memory usage is different across the different modes (see for e.g., Ghaffari, Li, Li, and Nie (2007)). In the forward mode, storage is needed for a) each Wengert variable and b) the forward Jacobian for each Wengert. In the reverse mode, storage is needed for a) each Wengert adjoint, b) the reverse Jacobian for each Wengert, and c) forward/reverse dependency graph.

For the purpose of implementing algorithmic differentiation, the constructs of the forward and the reverse mode can be theoretical conveniences. In situations where the dependence of the final Jacobian sensitivity step is the dominating factor, and the adjointing step is not the rate-determining part, then the performance will always be $\Theta(n)$, where n is the number of sensitivities – for e.g., if $y = \sum_{i=1}^{n} x_i$, given that $\frac{\partial y}{\partial x_i}$ is trivial to calculate, the performance will always be $\Theta(n)$. For instance, given a univariate objective function (as in constrained/unconstrained optimization (e.g., maximization/minimization) problems), either forward or reverse Algorithmic differentiation is an equally good choice for sensitivity generation, owing to its performance.

While the sensitivity generation discussed here uses the concepts employed in algorithmic differentiation, Credit Analytics does not use any tools for automated

differential code generation. This is mainly due to performance, but this does introduce additional software development/re-use challenges.

First, no matter how minimally intrusive the design is, building for algorithmic differentiation introduces its own deep-dig perspective. Re-purposing purely for the differentiation perspective forces the visualization of the computation at the granularity of the symbolic functional forms of the objective function.

For instance, usage of the objective function evaluator over-loading requires propagation of the inner most symbolic graph nodes through the graph chain, which causes export of a differential data structure. This results in the alteration/adjustment of the design around objective function invocation - with every Wengert variable, calculation of the set of forward sensitivities and the reverse Jacobians builds a local picture of the Objective Function without having to evaluate it.

Second, source code transformation techniques are very invasive, and require highly locally frozen view fixation, and are therefore harder to implement. Operator overloading enables retention of the domain focus, and is therefore easier to implement. However, naïve operator overloading would simply generate a block-level (or function call level) adjoint. This can explode the required storage, in addition to generating sub-optimal reverse-mode code. Needless to mention, source code transformation techniques can be built to overcome this – in practice, however, many algorithmic differentiation tools may not quite do it.

Finally, without the usage of obfuscating “versatile” templates, auto-generation of very generic forward/reverse accumulation code is impossible. Therefore source level function overloading and automated program instrumentation techniques are very hard. Further, just as in the case of compiled language source code transformation “smart compiler” efforts of the ‘90s, re-use of the algorithmic differentiation paradigms conceptually (rather than “built out-of the-box” through the AD tools) offers clear additional insight onto program construction.

Canonicalization - Program Statements Simplification by Decomposition

Here we discuss techniques used by several algorithmic differentiation tools to achieve line-level program decomposition (called canonicalization). Simply put, canonicalization decomposes the program/statement units into specific analysis bits. In various forms, canonicalization is commonly used in many areas of computer science, e.g., in compiler design/code generation, SKU formulation/synthesis/customization etc.

In general, canonicalization and other related algorithmic differentiation source code generation/transformation techniques go hand in hand with optimizing compiled code emission techniques, program active variable activity analysis – for instance, the canonicalization sequence includes steps (Bischof, Hovland, and Norris (2005)) where to mark out the “Algorithmically Differentiable” code from the others during, for instance, pre-processing etc. For true program transformation effectiveness, however, dynamic run-time analysis is needed in addition to static compile time data flow analysis etc. (For instance, in VM-based architectures, the run-time comprises of both Hot-Spot and GC, so it may make sense to embed algorithmic differentiation execution/selective sensitivity generation as well).

Canonicalization may also be viewed as being equivalent to the Wengert Structuring described above: Given that canonicalization consists of hoisting all the l-value updates separately without side effects, it is effectively the same as Wengert un-rolling and the forward DAG linearization.

Estimation of the program execution run-time costs becomes easy once the code is canonicalized. Given that the worst case is division, going from $c = \frac{a}{b}$ to

$\partial c = \frac{\partial a}{b} - \frac{a}{b^2}\partial b$ results in going from 1 function unit execution cost to 4 algorithmic differentiation execution unit costs. However, often due to presence of high-cost function units (such as log, exp, etc), the worst-case addition to a single post-canonicalized statement is a factor between 4 and 5.

A final caveat needs to be indicated in regards to the limitations with the implementation above. For many of the reasons indicated earlier, automated implementations of canonicalization (like other automated code generation/re-structuring) might result in "invisible inefficiencies", and the had-drafted techniques those are based upon essentially the same principles may be more optimal.

Optimization using Pre-accumulation and Check Pointing

Two other common techniques applied commonly across algorithmic differentiation implements are a) pre-accumulation, and b) Cross-country accumulation or Check-pointing.

Pre-accumulation refers to the process of aggregating (and possibly caching) the sensitivity Jacobian over all the intermediate Wengert's inside a routine/block/module, thereby only exposing $\frac{\partial Output_i}{\partial Input_j}$ for the group unit (not each Wengert inside). Where feasible, pre-accumulation also provides a suitable boundary for parallelization. It may also be looked at as the appropriate edge at which the source code transformation technique and operator overloading technique may "merge".

Cross-country Accumulation (or Check Pointing) is the same as pre-accumulation, but pre- accumulation occurs in a specified (forward/reverse) order, Cross-country accumulation need not – in fact it may be guided by program analysis using any of the optimal Wengert intermediate composition techniques. This typically also requires

snapshotting the program global and other execution context parameters at the checkpoint boundaries. Cross-country accumulation works best when the program state is easily and minimally savable, and quickly recoverable, and has the additional advantage of working well in conjunction with traditional kernel level check pointing schemes for fail-over etc:

Optimal Program Structure Synthesis

As indicated earlier, the forward mode (n inputs) and the reverse mode (m outputs) represent just two possible (extreme) ways of recursing through the chain rule. For n > 1 and m > 1 there is a golden mean that corresponds to synthesis of an optimal program structure, but finding the optimal way is probably an NP-hard problem (Berland (2006)) – optimal Jacobian accumulation is NP-complete (Naumann (2008)).

In Credit Analytics, program structure is optimized by searching for the set of intermediate Wengerts below which the independents fan in, and above which the dependents fan out. This is illustrated in Figures 1 to 3. If there exists an intermediate quantity that is fixed from the point-of-view of the output Jacobians and the input parameters, the performance may be improved (see Figure 1). Further, if the input/output computation leads to sufficient commonality among the Wengert intermediate calculation, that may also reduce computation by promoting reuse, thereby improving efficiency. In general, the condition $\frac{dP_i}{dMI_j} \to \delta_{ij}$ results in a Wengert fan-out – otherwise rippling out causes huge non-diagonal Markov matrices. Similar fan-in/fan-out constraint relations exist at the following boundaries (see Figure 3): a) I -> P, b) P-> W, and c) W -> O.

Extending these observations to computational finance (esp. computational fixed income finance), the payout/product/pricer object serves the function of the intermediate Wengert variate indicated above (Figure 2). From below this variate you have the inputs/parameters rippling up, and from above you have the Jacobians/output measure

adjoints feeding down. The nodes in Figure 2 also correspond to natural reactive Tree up-tick boundaries - every intermediate element in Figure 2 is a reactive tree dependent node from the entity below, so forwarding/adjointing should happen with every real-time uptick.

Thus, algorithmic differentiation for the Wengert variates involves the following:

- Identifying the abstractable financial canonical/reusable common object structures (market parameters, product parameters, pricer parameters, etc.)
- Working out their forward differentials and the reverse adjoints.

The intermediate Wengert variate view presented above is the conceptual parsimonisation of the variate parameters space and the Jacobian measure space.

Algorithmic Differentiation Financial Application Space Customization

Here we consider the specifics of some of the customization that may be needed for specific areas relevant to Credit Analytics.

Math Modules: Forward differentials and auto-adjointing may be needed for any of the math modules where sensitivity needs to be generated, it can go to the extent where, at every block, the base "value", forward differential, and reverse adjoint, are all computed. In fact, for every active double-precision variable v, source code transformation algorithmic differentiation techniques recursively automatically generate the doublet $\left(v, \dot{v} \right)$. Further, this calculation may also be parallelized.

Stochastic Variate Algorithmic Differentiation: Evolution of stochastic variates and their derivative entities may be further optimized by exploiting sparse-ness of the multi-factor co-variance matrix, thereby evolving the variate/derivative matrix that is sparse optimally (as opposed to blind delta bumps that may happen when computing differentials).

Given that variance reduction techniques are commonly used along the forward path of an evolution, these optimizations need to be preserved along the application of the algorithmic differentiation techniques as well. For instance, if a specific forward path a) does not need to be traveled, or b) certain forward Wengert intermediates automatically compute to zero, then these produce zero path derivatives. Further, external pre-computations can be done during the adjoint generation.

When generating sensitivities, special care needs to be exercised during the presence of optimal exercise dates, as they impose restrictions on how the path derivatives maybe computed using algorithmic differentiation. In particular the use of polynomial regression techniques to estimate the optimal exercise times also eases the introduction of the algorithmic differentiation into the code base.

While implementing algorithmic differentiation, the optimal (re) use of tangent multi-mode arc derivatives comes across at many places. While both the arc derivatives and their intermediates may be re-used, the circumstances under which they are effectively re-usable depend (as always) on the speed up and memory used.

In quasi-analytic computation Models, no Monte-Carlo evolution needed at all, but still Wengert intermediate level reformulation may be necessary to enhance the quasi-analytics analysis (e.g., Copula methods). Also, these correspond to the Adjoint-Natural Formulation Mode, i.e., typical quasi-analytic formulation often works out the Wengerts backwards from the final measure (e.g., say from PV), so they are automatically amenable to the adjoint mode of algorithmic differentiation.

Calibration and entity-variate focus: While calibrating a curve/surface to a quote for a calibration instrument measure, the de-convolving of the instrument entity/measure combination is necessary for the extraction of the parameter set (this is what is achieved by the calibration process). Of course, calibration occurs among the elastic and the inelastic dimensions, and the inelastics are parameter set (Krishnamurthy (2012)). The greeks calculated during this process, therefore, need to be accommodative of the

calibration set up – in particular, the Jacobian sensitivities needed are to the curve parameters, or the measure quotes.

3) Sensitivity Generation During Curve Construction

The basic premise of sensitivity generation during curve construction is simple: In addition to the usual advantage that algorithmic differentiation offers on doing Greeks on the same run as pricing, there is no need for multiple bumped curves anymore – but the proper Jacobians need to be calculated. Given that the calibration process calibrates the segment coefficients, further speed up may be achieved when the segment micro-Jacobian is pre-calculated right during the calibration (here, we need to calculate the Jacobian $\frac{\partial C_i}{\partial f_j}$, where C_i is the ith coefficient, and f_j is the jth input).

Depending on the nature of sensitivity sought, typical curve calibration deltas are with respect to one of the following:

- The underlying dynamical stochastic variates (e.g., the forward rates, zero rates, discount factors)
- The calibrated stochastic variate parameters (e.g., the segment spline coefficients)
- The unit change in the quoted instrument measures (e.g., 1 bp change). Here the Jacobians need to ripple upwards from the quoted instrument measures.

To work out the sensitivities, we start by distinguishing between the 5 types of span/segment elastic variates:

- Φ => Span stochastic evolution variate.
- Φ_k => Stochastic evolution variate for segment k.
- ϕ => Implied Span Quoted Instrument Measure.
- ϕ_k => Implied Quoted Instrument Measure for Segment k.
- φ_k => Observed Quoted Instrument Measure for Segment k at precisely a single variate point – typically, the observations are done at the anterior/posterior terminal ends of the segment.

For a given calculated/formulated output measure Ξ, the following are true by definition:

$$\Phi_k(t = t_k) = \Phi(t = t_k) \Rightarrow \left|\frac{\partial \Xi}{\partial \Phi}\right|_{t=t_k} = \left|\frac{\partial \Xi}{\partial \Phi_k}\right|_{t=t_k} \quad \textbf{(3.1)}$$

$$\varphi_k = \phi_k(t = t_k) = \phi(t = t_k) \Rightarrow \frac{\partial \Xi}{\partial \varphi_k} = \left|\frac{\partial \Xi}{\partial \phi}\right|_{t=t_k} = \left|\frac{\partial \Xi}{\partial \phi_k}\right|_{t=t_k} \quad \textbf{(3.2)}$$

We then identify the sensitivities to the elastic variates as:

- Sensitivity to Stochastic Evolution Variate => $\frac{\partial \Xi}{\partial \Phi}$
- Sensitivity to Implied Span Quoted Instrument Measure => $\frac{\partial \Xi}{\partial \phi}$
- Sensitivity to Observed Span Quoted Instrument Measure => $\frac{\partial \Xi}{\partial \varphi_k}$
- $\frac{\partial \Xi}{\partial \varphi_k}$ (Case c) above) is what you need to calculate the hedge ratio

If the segment variate is piece-wise constant, then $\frac{\partial \Xi}{\partial \Phi_k} = \frac{\partial \Xi}{\partial \phi_k} = \frac{\partial \Xi}{\partial \varphi_k}$.

Clearly the above relation is not valid if the segment variate is splined. Recall that segment spline coefficient calibration is simply a problem of matching to a terminal node (which is the quoted instrument measure at the terminal node). Thus, for a formulated output Ξ, at node k, it is obvious that $\frac{\partial \Xi}{\partial \Phi_k} \neq \frac{\partial \Xi}{\partial \phi_k}$. If Ξ refers to the discount factor, it can be shown that, where $t_j < t < t_{j+1}$,

$$D_F(t) = \exp\left\{-\int \Phi(t)dt\right\} = \exp\left\{-\sum_{i=0}^{j}\int_{t_i}^{t_{i+1}} \Phi_i(t)dt - \int_{t_j}^{t} \Phi_j(t)dt\right\} \quad \textbf{(3.3)}$$

Thus,

$$\frac{\partial D_F(t)}{\partial \Phi_k} = -D_F(t) * \begin{Bmatrix} t_{k+1} - t_k \; fork < j \\ t - t_k \; fork = j \\ 0 \; fork > j \end{Bmatrix} \quad \textbf{(3.4)}$$

The sensitivity of the quoted instrument measure, however, depends on the actual details of the quadrature. Thus

$$\frac{\partial D_F(t)}{\partial \phi_k} = -D_F(t) * \begin{Bmatrix} \int_{t_k}^{t_{k+1}} \frac{\partial \Phi}{\partial \varphi_k} \; fork < j \\ \int_{t_k}^{t} \frac{\partial \Phi}{\partial \varphi_k} \; fork = j \\ 0 \; fork > j \end{Bmatrix} \quad \textbf{(3.5)}$$

This is one of the many functional formulations in finance where the calculated product measure (Ξ) has a linear dependence on the stochastic evolution variate, i.e., $\Xi \Rightarrow \Psi\left[\int_{t_a}^{t_b} \Phi(t) dt\right]$. This implies that $\frac{\partial \Xi}{\partial \Phi_k} = \delta_{ik} \frac{\partial \Xi}{\partial \Psi_i}(t_{i+1} - t_i)$, i.e., $\frac{\partial \Xi}{\partial \Phi_k} \alpha \delta_{ik}$ only, and not on the quadrature details.

Curve Jacobian

Every Curve implementation needs to generate the Jacobian for the following measures from its parameterized representation scheme:

- Forward Rate Jacobian
- Discount Factor Jacobian

- Zero Rate Jacobian

Here we define self-Jacobian as $\frac{\partial Y(t)}{\partial Y(t_K)}$. Self-Jacobian computation efficiency is critical, since Jacobian of any function $F(Y)$ is going to be dependent on the self-Jacobian $\frac{\partial Y(t)}{\partial Y(t_K)}$ because of the chain rule.

Given $F(t_A, t_B)$ => Forward rate between times t_A and t_B, and $D_f(t_k)$ => Discount Factor at time t_k, the Forward Rate->DF Jacobian is computed as:

$$\frac{\partial F(t_A, t_B)}{\partial D_f(t_k)} = \frac{1}{t_B - t_A}\left\{\frac{1}{D_f(t_A)}\frac{\partial D_f(t_A)}{\partial D_f(t_k)} - \frac{1}{D_f(t_B)}\frac{\partial D_f(t_B)}{\partial D_f(t_k)}\right\} \quad \textbf{(3.6)}$$

If $Z(t)$ is the zero-rate at time t, Zero Rate->DF Jacobian is given from

$$\frac{\partial Z(t)}{\partial D_f(t_k)} = \frac{1}{t - t_0}\left\{\frac{1}{D_f(t)}\frac{\partial D_f(t)}{\partial D_f(t_k)}\right\} \quad \textbf{(3.7)}$$

Using the Zero Rate to Forward Rate Equivalence, equations (3.6) and (3.7) may be used to construct the Zero Rate Jacobian From the Forward Rate Jacobian.

The corresponding Quote->Zero Rate Jacobian is given from

$$\frac{\partial Q_j(t)}{\partial Z(t_k)} = (t_k - t_0)\left\{D_f(t_k)\frac{\partial Q_j(t)}{\partial D_f(t_k)}\right\} \quad \textbf{(3.8)}$$

Finally the PV->Quote Jacobian is given as:

$$\frac{\partial PV_j(t)}{\partial Q_k} = \sum_{i=1}^{n} \left\{ \frac{\partial PV_j(t)}{\partial D_f(t_i)} \div \frac{\partial Q_j(t)}{\partial D_f(t_i)} \right\} \quad \textbf{(3.9)}$$

Looking at the product level micro-Jacobians, given r_j => Cash Rate Quote for the j[th] Cash instrument, and $D_f(t_j)$ => Discount Factor at time t_j, the cash rate DF micro-Jacobian is given by

$$\frac{\partial r_j}{\partial D_f(t_k)} = -\frac{1}{\partial D_f(t_j)} \frac{1}{t_j - t_{START}} \frac{\partial D_f(t_j)}{\partial D_f(t_k)} \quad \textbf{(3.10)}$$

The Cash Instrument PV-DF micro-Jacobian is given as:

$$\frac{\partial PV_{CASH,j}}{\partial D_f(t_k)} = -\frac{1}{\partial D_f(t_{j,SETTLE})} \frac{\partial D_f(t_j)}{\partial D_f(t_k)} \quad \textbf{(3.11)}$$

There is practically no performance impact on construction of the PV-DF micro-Jacobian in then adjoint mode as opposed for forward mode, due to the triviality of the adjoint.

Given Q_j => Quote for the j[th] EDF with start date of $t_{j,START}$ and maturity of t_j, the Euro-dollar Future DF micro-Jacobian is

$$\frac{\partial Q_j}{\partial D_f(t_k)} = \frac{\partial D_f(t_j)}{\partial D_f(t_k)} \frac{1}{\partial D_f(t_{j,START})} - \frac{D_f(t_j)}{D_f{}^2(t_{j,START})} \frac{\partial D_f(t_{j,START})}{\partial D_f(t_k)} \quad \textbf{(3.12)}$$

The Euro-dollar Future PV-DF micro-Jacobian is given from

$$\frac{\partial PV_{EDF,j}}{\partial D_f(t_k)} = \frac{\partial D_f(t_j)}{\partial D_f(t_k)} \frac{1}{\partial D_f(t_{j,START})} - \frac{D_f(t_j)}{D_f{}^2(t_{j,START})} \frac{\partial D_f(t_{j,START})}{\partial D_f(t_k)} \quad \textbf{(3.13)}$$

As for the Cash instrument, there is practically no performance impact on construction of the PV-DF micro-Jacobian in then adjoint mode as opposed for forward mode, due to the triviality of the adjoint.

Before constructing the Interest Rate Swap DF micro-Jacobian, we define a few terms:

- $Q_j DV01_j = PV_{Floating,j}$
- Q_j => Quote for the j[th] IRS maturing at t_j.
- $DV01_j$ => DV01 of the swap
- $PV_{Floating,j}$ => Floating PV of the swap

$$\frac{\partial\left[Q_j DV01_j\right]}{\partial D_f(t_k)} = \frac{\partial\left[PV_{Floating,j}\right]}{\partial D_f(t_k)} = \frac{\partial Q_j}{\partial D_f(t_k)} DV01_j + Q_j \frac{dDV01_j}{\partial D_f(t_k)} \quad \textbf{(3.14)}$$

$$\frac{dDV01_j}{\partial D_f(t_k)} = \sum_{i=1}^{j} N(t_i)\Delta_i \frac{\partial D_f(t_i)}{\partial D_f(t_k)} \quad \textbf{(3.15)}$$

$$PV_{Floating,j} = \sum_{i=1}^{j} l_i N(t_i)\Delta_i D_f(t_i) \quad \textbf{(3.16)}$$

$$\frac{\partial PV_{Floating,j}}{\partial D_f(t_k)} = \sum_{i=1}^{j} N(t_i)\Delta_i D_f(t_i)\frac{\partial l_i}{\partial D_f(t_k)} + \sum_{i=1}^{j} l_i N(t_i)\Delta_i \frac{\partial D_f(t_i)}{\partial D_f(t_k)} \quad \textbf{(3.17)}$$

The Interest Rate Swap PV-DF micro-Jacobian is given from

$$\frac{\partial PV_{IRS,j}}{\partial D_f(t_k)} = \sum_{i=1}^{j} N(t_i)\Delta(t_{i-1},t_i)\left\{(c_j - l_i)\frac{\partial D_f(t_i)}{\partial D_f(t_k)} - D_f(t_i)\frac{\partial l_i}{\partial D_f(t_k)}\right\} \quad \textbf{(3.18)}$$

Again, there is no performance impact on construction of the PV-DF micro-Jacobian in then adjoint mode as opposed for forward mode, due to the triviality of the adjoint. Either

way the performance is $\Theta(n \times k)$, where n is the number of cash flows, and k is the number of curve factors.

Finally, we examine the Credit Default Swap DF micro-Jacobian.

- $PV_{CDS,j} = PV_{Coupon,j} - PV_{LOSS,j} + PV_{ACCRUED,j}$
- j => j[th] CDS Contract with a maturity t_j
- c_j => Coupon of the j[th] CDS
- $PV_{CDS,j}$ => PV of the full CDS contract
- $PV_{Coupon,j}$ => PV of the Coupon leg of the CDS Contract
- $PV_{ACCRUED,j}$ => PV of the Accrual paid on default

$$PV_{Coupon,j} = c_j \sum_{i=1}^{j} N(t_i)\Delta_i S_P(t_i) D_f(t_i) \quad \textbf{(3.19)}$$

$$\frac{\partial PV_{Coupon,j}}{\partial D_f(t_k)} = c_j \sum_{i=1}^{j} N(t_i)\Delta_i S_P(t_i) \frac{\partial D_f(t_i)}{\partial D_f(t_k)} + \frac{\partial c_j}{\partial D_f(t_k)} \sum_{i=1}^{j} N(t_i)\Delta_i S_P(t_i) D_f(t_i) \quad \textbf{(3.20)}$$

$$PV_{LOSS,j} = \int_0^{t_j} N(t)[1-R(t)] D_f(t) dS_P(t) \quad \textbf{(3.21)}$$

$$\frac{\partial PV_{LOSS,j}}{\partial D_f(t_k)} = \int_0^{t_j} N(t)[1-R(t)] \frac{\partial D_f(t)}{\partial D_f(t_k)} dS_P(t) \quad \textbf{(3.22)}$$

$$PV_{ACCRUED,j} = c_j \sum_{i=1}^{j} \int_{t_{i-1}}^{t_i} N(t)\Delta(t, t_{i-1}) D_f(t) dS_P(t) \quad \textbf{(3.23)}$$

$$\frac{\partial PV_{ACCRUED,j}}{\partial D_f(t_k)} = \frac{\partial c_j}{\partial D_f(t_k)} \sum_{i=1}^{j} \int_{t_{i-1}}^{t_i} N(t)\Delta(t,t_{i-1}) D_f(t) dS_P(t) + c_j \sum_{i=1}^{j} \int_{t_{i-1}}^{t_i} N(t)\Delta(t,t_{i-1}) \frac{\partial D_f(t)}{\partial D_f(t_k)} dS_P(t)$$

(3.24)

The Credit Default Swap DF micro-Jacobian is then given as

$$\frac{\partial PV_{CDS,j}}{\partial D_f(t_K)} = c_j \sum_{i=1}^{j} \left\{ \left[N(t_i)\Delta(t_{i-1},t_i) S(t_i) \frac{\partial D_f(t_i)}{\partial D_f(t_k)} \right] + \int_{t_{i-1}}^{t_i} N(t) \left[c_j \Delta(t_{i-1},t) - \{1 - R(t)\} \right] dP(t) \right\}$$

(3.25)

Again, there is no performance impact on construction of the PV-DF micro-Jacobian in then adjoint mode as opposed for forward mode, due to the triviality of the adjoint. Either way the performance is $\Theta(n \times k)$, where n is the number of cash flows, and k is the number of curve factors.

4) Stochastic Entity Evolution

We start with the formulation of the sensitivities for the evolution of the stochastic entities. The simplest evolution dynamics of the stochastic variables $L_i(t)$ will be ones with constant forward volatilities. Once the dynamics is formulated according to

$$\Delta L_i(t) = \mu_i(L_i, t)\Delta t + \sum_j \sigma_{ij}(L_i, t)\Delta W_j \qquad \textbf{(4.1)}$$

where $\mu_i(L_i, t)$ is the component drift, and $\sigma_{ij}(L_i, t)$ is the component co-variance to the factor $W_j(L_i, t)$, subsequent evolution can be determined.

The discretized, Eulerized version of the above is

$$\Delta x_j(t) = h\mu_j\left(\vec{x}, t\right) + \sqrt{h}\sum_l \sigma_{jl}\left(\vec{x}, t\right)\Delta Z_l \qquad \textbf{(4.2)}$$

where h is the time-step, and Z is the Weiner random variable. In the case of forward rates, e.g., the drifts can be established by a no-arbitrage condition binding the forward rate drifts to their variances.

Once the stochastic variate dynamics is established, the dynamics of the observed derivative entity can be progressively determined.

The evolution sequence can be determined for the individual pay-off measures as well. These measures may further be dependent on the path-wise differentials of the derivative entity, so those may also need to be evolved using algorithmic differentiation.

Several techniques have been considered in the literature for enhancing the computational efficiency of the derivative entity:

- Using the adjoint algorithmic differentiation methods
- Using optimal combination of forward and adjoint algorithmic differentiation methods
- Further optimizations using sparse-ness of the multi-factor co-variance matrix, thereby evolving the variate/derivative matrix that is sparse optimally (as opposed to delta bumps that may happen when computing differentials).
- Quasi-analytic computation models and algorithmic differentiation techniques do not Monte-Carlo evolution needed at all, but still Wengert intermediate level reformulation necessary to enhance the quasi-analytics analysis (e.g., Copula methods).

A note on the Derivative Entity Measure Sensitivity: Sensitivity calculations reduce to calculating the quoted measure of the set of the input calibrated derivative entity input to the quoted measure of the output derivative entity (they are maintained in the Jacobian). In practice, however, the sensitivity Jacobian may be computed as sensitivity of the calibrated parameter to the output derivative entity Output map.

Before detailing the calculations of the sensitivities, it makes sense to differentiate between the different types of variables that serve as the “dependent” variables to which we compute the sensitivities.

First are the stochastic state variates. These are base stochastic entities that characterize the actual system statics/dynamics. The sensitivities to the state variates are typically sensitivities to the “current” (or starting) realization of these variates – e.g., delta, gamma.

The next group is the dynamic parameters. These are the model parameters that govern the evolution/equilibrium behavior of the state variates, and thereby the system dynamics. Examples would be sensitivities to volatility, correlation, etc:

The final group is that of the segment/span coefficients. These coefficients serve act as the interpolated "PROXY" for the segments at the unobserved points in the segment. Sensitivities may also be sought to these coefficients.

Additional considerations need to be accounted for when treating the stochastic variate evolution constrained by splines. The forward rates (or indeed any term instrument measures) need to evolve such that

- They are continuous at the boundaries
- The first (and possibly the second) derivatives are continuous at the boundaries
- The boundary conditions (either financial or tensional) are retained intact

For instance, the evolution dynamics of the forward rates (or indeed any term instrument measures) can still be via LMM, but splines may still be applicable to the intermediate nodes, as the segment spline coefficients adjust to the forward rate nodes.

Splines may also be used for any term instrument measure determinant (e.g., the volatility surface maybe also be interpolatively constructed using splines), so as to preserve the continuity/smoothness, as opposed to piece-wise discreteness.

We now treat the formulation of the Evolution of Stochastic Variate Self-Jacobian in detail. We start with base evolution equation for a stochastic variate

$$\Delta x_j(t) = \mu_j(x_1 \ldots x_n, t)\Delta t + \sum_{l=1}^{m} \sigma_{jl}(x_1 \ldots x_n, t)\Delta W_l(t) \quad \textbf{(4.3)}$$

We define the Self-Jacobian (J_{ij}) Delta as

$$J_{ij} = \frac{\partial x_i(t)}{\partial x_j(0)} \quad \textbf{(4.4)}$$

We now formulate the evolution of the sensitivity. Let:

- i => Index over the number of underliers (1 to n)
- l => Index over the number of independent stochastic factors (1 to m)

Then extending equation (4.3) we get

$$\frac{\partial \Delta x_j(t)}{\partial x_K(0)} = \Delta t\left[\sum_{i=1}^{n}\frac{\partial \mu_j(x_1...x_n,t)}{\partial x_i(t)}\frac{\partial x_i(t)}{\partial x_K(0)}\right] + \sum_{l=1}^{m}\Delta W_l(t)\left[\sum_{i=1}^{n}\frac{\partial \sigma_{jl}(x_1...x_n,t)}{\partial x_i(t)}\frac{\partial x_i(t)}{\partial x_K(0)}\right] \quad \textbf{(4.5)}$$

Considering the Euler-discretized version of the above, we get

$$\frac{\partial \Delta x_j(t)}{\partial x_K(0)} = h\left[\sum_{i=1}^{n}\frac{\partial \mu_j(x_1...x_n,t)}{\partial x_i(t)}\frac{\partial x_i(t)}{\partial x_K(0)}\right] + \sqrt{h}\sum_{l=1}^{m}\mathrm{Z}_l(t)\left[\sum_{i=1}^{n}\frac{\partial \sigma_{jl}(x_1...x_n,t)}{\partial x_i(t)}\frac{\partial x_i(t)}{\partial x_K(0)}\right] \quad \textbf{(4.6)}$$

Re-write Equation (4.6) to get

$$\frac{\partial x_j(t+h)}{\partial x_k(0)} = \sum_{i=1}^{n}\left[\delta_{ji} + h\frac{\partial \mu_j(t)}{\partial x_i(t)} + \sqrt{h}\sum_{l=1}^{m}\left\{\mathrm{Z}_l(t+h)\frac{\partial \sigma_{jl}(t)}{\partial x_i(t)}\right\}\right]\frac{\partial x_i(t)}{\partial x_k(0)} = \sum_{i=1}^{n}D_{ji}(k,t)\frac{\partial x_i(t)}{\partial x_k(0)}$$

(4.7)

where

$$D_{ji}(k,t) = \delta_{ji} + h\frac{\partial \mu_j(t)}{\partial x_i(t)} + \sqrt{h}\sum_{l=1}^{m}\left\{\mathrm{Z}_l(t+h)\frac{\partial \sigma_{jl}(t)}{\partial x_i(t)}\right\} \quad \textbf{(4.8)}$$

Equation (4.8) can be re-cast as

$$\left[\frac{\partial x(t+h)}{\partial x_k(0)}\right] = [D(k,t)]\left[\frac{\partial x(t)}{\partial x_k(0)}\right] \quad \textbf{(4.9)}$$

where $\left[\frac{\partial x(t+h)}{\partial x_k(0)}\right]$ and $\left[\frac{\partial x(t)}{\partial x_k(0)}\right]$ are column matrices, and $[D(k,t)]$ is an n x n square matrix.

The time evolution of Equation (4.9) can be iterated as

$$\left[\frac{\partial x(t+h)}{\partial x_k(0)}\right] = [D(k,t)][D(k,t-h)]....[D(k,0)]\left[\frac{\partial x(0)}{\partial x_k(0)}\right] \quad \textbf{(4.10)}$$

While this is still forward algorithmic differentiation mode and is $\Theta(n)$, this may be optimized using specific path-wise techniques shown in Glasserman and Zhao (1999). However, further significant optimization can be achieved by adjointing techniques [Griewank (2000), Giles and Pierce (2000)]. To achieve this, transpose Equation (4.10) to get the following adjoint form:

$$\left[\frac{\partial x(t+h)}{\partial x_k(0)}\right]^{\mathrm{T}} = \left[\frac{\partial x(0)}{\partial x_k(0)}\right]^{\mathrm{T}} [D(k,0)]^{\mathrm{T}}[D(k,h)]^{\mathrm{T}} ....[D(k,t-h)]^{\mathrm{T}}[D(k,t)]^{\mathrm{T}} \quad \textbf{(4.11)}$$

Equation (4.11) reduces to vector/matrix as opposed to matrix/matrix in the non-transposed version Equation (4.10), and would thus be $\Theta(n^2)$, as opposed to $\Theta(n^3)$.

To gain insight from the components of $D_{ji}(k,t)$ in Equation (4.8), decompose $D_{ji}(k,t)$ as

$$D_{ji}(k,t) = D_{ji,PRIOR}(k,t) + D_{ji,DRIFT}(k,t) + D_{ji,VOLATILITY}(k,t) \quad \textbf{(4.12)}$$

This separates out the different contributions to $D_{ji}(k,t)$.

a) The term $D_{ji,PRIOR}(k,t) = \delta_{ji}$ is the contribution due to the previous D, i.e., $D_{ji}(k,t-h)$.

b) The term $D_{ji,\ DRIFT}(k,t) = h\dfrac{\partial \mu_j(t)}{\partial x_i(t)}$ is the contribution from the derivative of the drift term.

c) The term $D_{ji,VOLATILITY}(k,t) = \sqrt{h}\sum_{l=1}^{m}\left\{ Z_l(t+h)\dfrac{\partial \sigma_{jl}(t)}{\partial x_i(t)} \right\}$ is the contribution from the volatility derivative.

Finally, we compute the self-Jacobian Gamma Γ_{ij}. We define it as

$$\Gamma_{ij} = \frac{\partial^2 x_C(t)}{\partial x_A(0)\partial x_B(0)} \quad \textbf{(4.13)}$$

It is fairly straightforward (albeit tedious) to show that

$$\frac{\partial^2 x_C(t+h)}{\partial x_A(0)\partial x_B(0)} = \sum_{i=1}^{n}\sum_{j=1}^{n}\left[S_{ij}\left(C, \vec{x}(t), t \right)\frac{\partial x_i(t)}{\partial x_A(0)}\frac{\partial x_j(t)}{\partial x_B(0)} \right] + \sum_{i=1}^{n}\left[M_i\left(C, \vec{x}(t), t \right)\frac{\partial^2 x_i(t)}{\partial x_A(0)\partial x_B(0)} \right]$$

(4.14)

where

$$S_{ij}\left(C, \vec{x}(t), t \right) = h\frac{\partial^2 \mu_C\left(\vec{x}(t), t \right)}{\partial x_i(t)\partial x_j(t)} + \sqrt{h}Z_C(t+h)\frac{\partial^2 \sigma_C\left(\vec{x}(t), t \right)}{\partial x_i(t)\partial x_j(t)} \quad \textbf{(4.15)}$$

and

$$M_i\left(C, \vec{x}(t), t \right) = h\frac{\partial \mu_C\left(\vec{x}(t), t \right)}{\partial x_i(t)} + \sqrt{h}Z_C(t+h)\frac{\partial \sigma_C\left(\vec{x}(t), t \right)}{\partial x_i(t)} \quad \textbf{(4.16)}$$

Thus far correlation among the stochastic variables ha s not been treated explicitly – we now remove that. We start with by formulating the evolution process for continuously evolving stochastic variates. Given $\vec{X}$ => the vector of financial variables that need to be mapped to the corresponding Weiner variates $\vec{Z}$. For instance, among forward rates that use the LMM dynamics, the evolution sequence may start with $\vec{X}(0) = \{X_1(0), X_2(0), ..., X_n(0)\}$, after which the LMM evolutionary techniques generate $\vec{Z}$ and update $\vec{X}(t)$. Extensions to model correlations among non-LMM–type asset movements is straightforward too, but additional transformation is required. For instance, if the default process can be correspondingly transformed to an asset indicator variable, that may be correlated with the other asset variables too.

Extending equation (4.3) for a set of correlated variates, the correlated stochastic evolution equation can be written as:

$$X_j(t+h) = X_j(t) + h\mu_j\left(\vec{X}, t\right) + \sqrt{h}\sigma_j\left(\vec{X}, t\right)\sum_{l=1}^{m}\rho_{jl}\left(\vec{X}, t\right)\mathrm{Z}_l(t+h) \quad \textbf{(4.17)}$$

Here $\sigma_j\left(\vec{X}, t\right)$ is the variance, and $\rho_{jl}\left(\vec{X}, t\right)$ is the correlation matrix – the variance is factored out of the covariance matrix to produce the correlation grid. $\mathrm{Z}_l(t+h)$ is produced by the usual i.i.d. $\aleph(0,1)$. Extending the sensitivity formulation analogous to Equation (4.9) above, the corresponding delta is:

$$\left[\frac{\partial \vec{X}(t+h)}{\partial X_k(0)}\right] = [D]\left[\frac{\partial \vec{X}(t)}{\partial X_k(0)}\right] \quad \textbf{(4.18)}$$

The entry in matrix D is given as:

$$D_{ij} = \delta_{ij} + h\frac{\partial \mu_j\left(\vec{X},t\right)}{\partial X_i(t)} + \sqrt{h}\sum_{l=1}^{m} Z_l(t+h)\left\{\rho_{jl}\left(\vec{X},t\right)\frac{\partial \sigma\left(\vec{X},t\right)}{\partial X_i(t)} + \sigma\left(\vec{X},t\right)\frac{\partial \rho_{jl}\left(\vec{X},t\right)}{\partial X_i(t)}\right\}$$

(4.19)

The corresponding parameter sensitivity evolution dynamics are specified by:

$$\left[\frac{\partial \vec{X}(t+h)}{\partial \alpha}\right] = [D]\left[\frac{\partial \vec{X}(t)}{\partial \alpha}\right] \quad \textbf{(4.20)}$$

Equation (4.20) may be simplified in cases where α is an explicit function ONLY of the state evolution variables as:

$$\frac{\partial X_j(t+h)}{\partial \alpha} = \frac{\partial X_j(t)}{\partial \alpha} + h\frac{\partial \mu_j\left(\vec{X},t\right)}{\partial \alpha} + \sqrt{h}\sum_{l=1}^{m} Z_l(t+h)\left[\frac{\partial \sigma_j\left(\vec{X},t\right)}{\partial \alpha}\rho_{jl}\left(\vec{X},t\right) + \sigma_j\left(\vec{X},t\right)\frac{\partial \rho_{jl}\left(\vec{X},t\right)}{\partial \alpha}\right]$$

(4.21)

An alteration to the above algorithm is needed when needed to generate correlated default times efficiently. Unlike the continuous variables above, if we are to consider the correlations between default times ONLY, it is much more efficient to draw correlated default times – again this correlation is different from that of continuous asset value times that results in default.

The algorithm for the generation of Correlated Default Times is as follows:

- Generate the vector $\vec{Z}_{INDEPENDENT}$.
- Factorize the correlation matrix ρ_{jk} to create the Cholesky diagonal matrices C and C^{T}.

- Use the Cholesky transformation to create $\vec{Z}_{CORRELATED}$ from $\vec{Z}_{INDEPENDENT}$ using $\vec{Z}_{CORRELATED} = C\vec{Z}_{INDEPENDENT}$.
- For each entity $\tilde{Z}_i$ in $\vec{Z}_{CORRELATED}$:
  a) Evaluate the cumulative normal $y_i = \int_{x=-\infty}^{x=\tilde{Z}_i} \aleph(0,1)dx$, where $\aleph(0,1)$ is a Normal distribution with unit mean and zero variance.
  b) $\tau_{i,DEFAULT} = S_i^{-1}(y_i)$ where S_i is the survival probability for the entity i.
  c) In general, $X_i = M_i^{-1}(y_i)$.

LMM Forward Rate Evolution

We now treat the special case of evolution of the stochastic forward rate using the LIBOR Market Model (LMM) dynamics. LMM formulation is particularly important, as it is one of the most popularly used formulation, and is essential to evaluate the impact the no-arbitrage constrained drift has on the evolution and the impact on the greeks. Of course, the results applicable to the lognormal nature of the forward rate $\vec{L}(t)$ are important in its own right.

The no-arbitrage constraint specification of LMM is first specified in conjunction with the base forward rate evolution dynamics:

$$L_j(t+h) = L_j(t) + h\mu_j\left(\vec{L}(t), t\right) + \sqrt{h}Z_j(t+h)\sigma_j\left(\vec{L}(t), t\right) \quad \textbf{(4.22)}$$

$$\mu_j\left(\vec{L}(t), t\right) = b_j L_j(t) \sum_{p=\eta(t)}^{j} \frac{b_p \Delta(t_{p-1}, t_p) L_p(t)}{1 + \Delta(t_{p-1}, t_p) L_p(t)} \quad \textbf{(4.23)}$$

$$\sigma_j\left(\vec{L}(t),t\right) = b_j L_j(t) \quad \textbf{(4.24)}$$

$\eta(t)$ is the maturity of the first instrument that matures after t [Brace, Gatarek, and Musiela (1997), Jamshidian (1997)].

At this stage, the distinction between the forward rate volatility and at-the-money swap option volatility needs to be made. LMM uses forward rate volatilities, so there needs to be a conversion step that involves converting the market observed at-the-money swap option volatility onto LMM forward rate volatility [Brigo and Mercurio (2001)].

With the above as base, the self-Jacobian of the forward rate using LMM is easy to formulate. As shown in Denson and Joshi (2009a) and Denson and Joshi (2009b),

$$\frac{\partial L_j(t+h)}{\partial L_k(0)} = \frac{\partial L_j(t)}{\partial L_k(0)} + \sum_{i=1}^{n}\left\{h\frac{\partial \mu_j\left(\vec{L}(t),t\right)}{\partial L_i(t)} + \sqrt{h}Z_j(t+h)\frac{\partial \sigma_j\left(\vec{L}(t),t\right)}{\partial L_i(t)}\right\}\frac{\partial L_i(t)}{\partial L_k(0)} \quad \textbf{(4.25)}$$

$$\frac{\partial \sigma_j\left(\vec{L}(t),t\right)}{\partial L_i(t)} = \delta_{ji} b_i \quad \textbf{(4.26)}$$

$$\frac{\partial \mu_j\left(\vec{L}(t),t\right)}{\partial L_i(t)}(\eta(t) \le i) = b_j \delta_{ji} \sum_{p=\eta(t)}^{j} \frac{b_p \Delta(t_{p-1},t_p) L_p(t)}{1+\Delta(t_{p-1},t_p)L_p(t)} + \frac{b_j L_j(t)}{(1+\Delta(t_{i-1},t_i)L_i(t))^2} \quad \textbf{(4.27)}$$

$$\frac{\partial \mu_j\left(\vec{L}(t),t\right)}{\partial L_i(t)}(\eta(t) > i) = b_j \delta_{ji} \sum_{p=\eta(t)}^{j} \frac{b_p \Delta(t_{p-1},t_p) L_p(t)}{1+\Delta(t_{p-1},t_p)L_p(t)} \quad \textbf{(4.28)}$$

We extend Equations (4.9) and (4.18) to evaluate the forward-rate evolution matrix. As expected,

$$\left[\frac{\partial \vec{L}(t)}{\partial x_k(0)}\right]^{\mathrm{T}} = \left[\frac{\partial \vec{L}(0)}{\partial x_k(0)}\right]^{\mathrm{T}} [D(k,0)]^{\mathrm{T}}[D(k,h)]^{\mathrm{T}} \ldots.[D(k,t-h)]^{\mathrm{T}}[D(k,t)]^{\mathrm{T}} \quad \textbf{(4.29)}$$

$$D_{ij}\left(\vec{L}(t),t\right)(\eta(t) \le i) = \delta_{ij}\left[1 + hb_j \sum_{p=\eta(t)}^{j} \frac{b_p \Delta(t_{p-1},t_p)L_p(t)}{1+\Delta(t_{p-1},t_p)L_p(t)} + \sqrt{h}b_j Z_j(t+h)\right] + \frac{hb_j L_j(t)}{(1+\Delta(t_{i-1},t_i)L_i(t))^2} \quad \textbf{(4.30)}$$

$$D_{ij}\left(\vec{L}(t),t\right)(\eta(t) > i) = \delta_{ij}\left[1 + hb_j \sum_{p=\eta(t)}^{j} \frac{b_p \Delta(t_{p-1},t_p)L_p(t)}{1+\Delta(t_{p-1},t_p)L_p(t)} + \sqrt{h}b_j Z_j(t+h)\right] \quad \textbf{(4.31)}$$

Finally, the variate Jacobian Parameter Sensitivity is also straightforward to evaluate.

$$\frac{\partial L_j(t+h)}{\partial \alpha} = \frac{\partial L_j(t)}{\partial \alpha} + h\frac{\partial \mu_j\left(\vec{L}(t),t\right)}{\partial \alpha} + \sqrt{h}Z_j(t+h)\frac{\partial \sigma_j\left(\vec{L}(t),t\right)}{\partial \alpha} + \sum_{i=1}^{n} D_{ij}\left(\vec{L}(t),t\right)\frac{\partial L_i(t)}{\partial \alpha} \quad \textbf{(4.32)}$$

$D_{ij}\left(\vec{L}(t),t\right)$ is available from above for the two scenarios treated in detail earlier. Re-casting $\frac{\partial L_j(t+h)}{\partial \alpha}$ as

$$\frac{\partial L_j(t+h)}{\partial \alpha} = B_j\left(\vec{L}(t),t\right) + \sum_{i=1}^{n} D_{ij}\left(\vec{L}(t),t\right)\frac{\partial L_i(t)}{\partial \alpha} \quad \textbf{(4.33)}$$

$$B_j\left(\vec{L}(t),t\right) = L_j(t)\left\{\sqrt{h}Z_j(t+h)\frac{\partial b_j}{\partial \alpha} + h\sum_{p=\eta(t)}^{j}\frac{\Delta(t_{p-1},t_p)L_p(t)}{1+\Delta(t_{p-1},t_p)L_p(t)}\left[b_p\frac{\partial b_j}{\partial \alpha} + b_j\frac{\partial b_p}{\partial \alpha}\right]\right\} \quad \textbf{(4.34)}$$

5) Formulation of Sensitivities for Pay-off Functions

Having treated the evolution of the stochastic variate in detail, we now attend to the formulation of the pay-off function for the stochastic evolution.

Algorithms to estimate Monte-Carlo Path-wise Derivatives are now well established. However, path-wise derivatives are typically forward derivatives, not adjoint [Giles and Glasserman (2006)]. Therefore computation time is proportional to the number of inputs. Further, it is not easy to accommodate these in complex payouts [Capriotti (2011)].

The Payoff Expectation is given as

$$V = E_Q\left[P\left(\vec{X} \right) \right] \quad \textbf{(5.1)}$$

[e.g., in Harrison and Kreps (1979)], where $\vec{X}$ is the vector of financial variables. The corresponding path Payoff Expectation can be given as [Kallenberg (1997)]

$$V = \frac{1}{N_{MC}} \sum_{i_{MC}=1}^{N_{MC}} P\left(\vec{X}\left[i_{MC} \right] \right) \quad \textbf{(5.2)}$$

and the corresponding variance from

$$Variance = \frac{{N_{MC}}^2 \sum_{i_{MC}=1}^{N_{MC}} \left[\left\{ P\left(\vec{X}\left[i_{MC} \right] \right) \right\}^2 \right] - \left\{ \sum_{i_{MC}=1}^{N_{MC}} P\left(\vec{X}\left[i_{MC} \right] \right) \right\}^2}{{N_{MC}}^2} \quad \textbf{(5.3)}$$

Path Greek

The Estimate of Path Sensitivity is unbiased [Kunita (1990), Protter (1990), Broadie and Glasserman (1996), Glasserman (2004)] if

$$\left\langle \frac{\partial Y(x)}{\partial x(0)} \right\rangle = \frac{\partial}{\partial x(0)} \langle \partial Y(x) \rangle \quad \textbf{(5.4)}$$

Here $x(0)$ is the starting point for the variate.

The path Monte-Carlo Greek is defined at the change in Y with respect to the starting value of x, i.e., $x(0)$.

$$\frac{\partial Y(x(t))}{\partial x(0)} = \frac{\partial Y(x(t))}{\partial x(t)} \frac{\partial x(t)}{\partial x(0)} \quad \textbf{(5.5)}$$

If x is a multi-component vector $\vec{X}$, then

$$\frac{\partial Y\left[\vec{X}(t)\right]}{\partial x_j(0)} = \sum_{i=1}^{n} \frac{\partial Y\left[\vec{X}(t)\right]}{\partial x_j(t)} \frac{\partial x_j(t)}{\partial x_j(0)} \quad \textbf{(5.6)}$$

The Pay-off Function Delta is defined simply as

$$\frac{\partial V(t)}{\partial x_k(0)} = \sum_{j=1}^{n} \frac{\partial V(t)}{\partial x_j(t)} \frac{\partial x_j(t)}{\partial x_k(0)} \quad \textbf{(5.7)}$$

The earlier formulation for $\left[\frac{\partial x(t)}{\partial x_k(0)}\right]$ may be used to establish the path delta. In particular, using Equations (4.9) and (4.18),

$$\left[\frac{\partial V(t)}{\partial x_k(0)}\right]^{\mathrm{T}} = \left[\frac{\partial V(0)}{\partial x_k(0)}\right]^{\mathrm{T}} [D(k,0)]^{\mathrm{T}}[D(k,h)]^{\mathrm{T}} \ldots.[D(k,t-h)]^{\mathrm{T}}[D(k,t)]^{\mathrm{T}} \quad \textbf{(5.8)}$$

Thus, all the speed up advantages associated with the adjoint formulation above follows.

In addition to the base Greeks, the Variance in the Greeks may also be computed as follows:

1. Cluster all the Path-wise Greeks calculated for a given input (either $x_k(0)$ or a parameter θ).
2. Within that cluster estimate the corresponding Greek.
3. Usual population sampling variance techniques applied to compute the variance in the Greek.

The Path Parameter (α) Sensitivity may be evaluated as:

$$\frac{dV(t)}{d\alpha} = \frac{\partial V(t)}{\partial \alpha} + \sum_{j=1}^{n} \frac{\partial V(t)}{\partial x_j(t)} \frac{\partial x_j(t)}{\partial \alpha} \quad \textbf{(5.9)}$$

The earlier formulation for $\left[\frac{\partial x(t)}{\partial \alpha}\right]$ may be used to establish the path parameter sensitivity.

The pay-off Greek may be explicitly formulated as:

$$\frac{\partial x_j(t+h)}{\partial \alpha} = h\frac{\partial \mu_j(t)}{\partial \alpha} + \sqrt{h}\sum_{l=1}^{m} Z_l(t+h)\frac{\partial \sigma_{jl}(t)}{\partial \alpha} + \sum_{i=1}^{n}\left\{\delta_{ji} + h\frac{\partial \mu_j(t)}{\partial x_i(t)} + \sqrt{h}\sum_{l=1}^{m} Z_l(t+h)\frac{\partial \sigma_{jl}(t)}{\partial x_i(t)}\right\}\frac{\partial x_i(t)}{\partial \alpha}$$

(5.10)

Additional partial derivative terms arise owing to the explicit dependence of μ, σ on α. (Otherwise $B_j(t,\alpha) = 0$). The pay-off parameter sensitivity formulation proceeds precisely along the same lines as delta formulation.

$$\frac{\partial x_j(t+h)}{\partial \alpha} = B_j(t,\alpha) + \sum_{i=1}^{n} D_{ji}(t,\alpha)\frac{\partial x_i(t)}{\partial \alpha} \quad \textbf{(5.11)}$$

While $D_{ji}(t,\alpha)$ is exactly the same as earlier, $B_j(t,\alpha)$ is given from

$$B_j(t,\alpha) = h\frac{\partial \mu_j(t)}{\partial \alpha} + \sqrt{h}\sum_{l=1}^{m} Z_l(t+h)\frac{\partial \sigma_{jl}(t)}{\partial \alpha} \quad \textbf{(5.12)}$$

$$\left[\frac{\partial \vec{x}(t+h)}{\partial \alpha}\right] = [B(\alpha,t)] + [D(\alpha,t)]\left[\frac{\partial \vec{x}(t)}{\partial \alpha}\right] \quad \textbf{(5.13)}$$

Here $\left[\frac{\partial \vec{x}(t+h)}{\partial \alpha}\right]$, $\left[\frac{\partial \vec{x}(t)}{\partial \alpha}\right]$, and $[B(\alpha,t)]$ are n x 1 column matrices, and $[D(k,t)]$ is an n x n square matrix. Generalizing over all the j's, we get

$$\left[\frac{\partial \vec{x}(t+h)}{\partial \alpha}\right] = \sum_{e=0}^{s}\left[\left\{\prod_{f=1}^{e}[D(t-fh)]\right\}[B(t-eh)]\right] + \left\{\prod_{e=1}^{s}[D(t-eh)]\right\}\left[\frac{\partial \vec{x}(t)}{\partial \alpha}\right] \quad \textbf{(5.14)}$$

Transposing the above we get

$$\left[\frac{\partial \vec{x}(t+h)}{\partial \alpha}\right]^{\mathrm{T}} = \sum_{e=s}^{1}[B(\alpha,t-eh)]^{\mathrm{T}}\left[\left\{\prod_{f=e}^{1}[D(\alpha,t-fh)]^{\mathrm{T}}\right\}\right] + \left[\frac{\partial \vec{x}(t)}{\partial \alpha}\right]^{\mathrm{T}}\left\{\prod_{e=s}^{1}[D(\alpha,t-eh)]^{\mathrm{T}}\right\}$$

(5.15)

Given that $[B(\alpha,t)]^{\mathrm{T}}$ and $\left[\frac{\partial \vec{x}(t)}{\partial \alpha}\right]^{\mathrm{T}}$ are row matrices, and that they are the preceding terms in the series, all the adjoint advantages indicated earlier continue to be valid. Further the previous formulations for $[D(\alpha,t)]$ can be re-used at the same Eulerian time step.

The adjointing step above causes additional storage demands. Since $[B(\alpha,t)]$ and $[D(\alpha,t)]$ still need to be retained in memory during the forward evolutionary sweep for $\left[\frac{\partial \vec{x}(t)}{\partial \alpha}\right]$, this represents a corresponding increase on the storage requirements.

The next step is to formulate the dependence of the payoff sensitivity to the correlation matrix. Irrespective of where the stochastic process is diffusive or not, the sensitivity of the pay-off to the correlation matrix entry ρ_{jk} ($\frac{\partial V}{\partial \rho_{jk}}$) is given as

$$\frac{\partial V}{\partial \rho_{jk}} = \sum_{i=1}^{n}\frac{\partial V}{\partial \tilde{\mathrm{Z}}_i}\frac{\partial \tilde{\mathrm{Z}}_i}{\partial \rho_{jk}} \quad \textbf{(5.16)}$$

Recall that if

$$y(z) = \int_{x=-\infty}^{x=z}\Phi(x)dx \Rightarrow \frac{\partial y}{\partial z} = \Phi(z) \quad \textbf{(5.17)}$$

From this, and using $X_i = M_i^{-1}(y_i)$, you get

$$\frac{\partial X_i}{\partial \tilde{Z}_i} = \Phi\left(\tilde{Z}_i\right)\frac{\partial X_i}{\partial \Phi(X_i)} \quad \textbf{(5.18)}$$

With the above, we are ready to estimate the differential of the Cholesky Factorization Matrix. Using $\frac{\partial C_{lm}}{\partial \rho_{jk}}$ as given in Smith (1995), we get

$$\frac{\partial \tilde{Z}_i}{\partial \rho_{jk}} = \sum_{l=1}^{n}\sum_{m=1}^{n}\frac{\partial \tilde{Z}_i}{\partial C_{lm}}\frac{\partial C_{lm}}{\partial \rho_{jk}} \quad \textbf{(5.19)}$$

Therefore, using $\frac{\partial \tilde{Z}_i}{\partial \rho_{jk}}$ is given from above, we get

$$\frac{\partial V}{\partial \rho_{jk}} = \sum_{i=1}^{n}\frac{\partial V}{\partial \tilde{Z}_i}\frac{\partial \tilde{Z}_i}{\partial \rho_{jk}} = \sum_{i=1}^{n}\frac{\partial V}{\partial X_i}\frac{\partial X_i}{\partial \tilde{Z}_i}\frac{\partial \tilde{Z}_i}{\partial \rho_{jk}} = \sum_{i=1}^{n}\frac{\partial V}{\partial X_i}\Phi\left(\tilde{Z}_i\right)\frac{\partial X_i}{\partial \Phi(X_i)}\frac{\partial \tilde{Z}_i}{\partial \rho_{jk}} \quad \textbf{(5.20)}$$

It is important to note the distinction between adjoint mode sensitivities vs. reverse mode algorithmic differentiation. Typically adjoint refers **ONLY** to the intermediate/dynamical matrices [Giles (2007), Giles (2009)], whereas **REVERSE** refers to calculation of only the relevant outputs and their sensitivities [Griewank (2000)]. Adjointing deals with the evolved variate space parameters left to right, therefore technically it is reverse in the time sense – and achieves optimization by minimizing the matrix<->matrix computations. In the sense of adjoint algorithmic differentiation, however, the term reverse and adjoint are used synonymously, i.e., adjoint/reverse refer to a scan backwards from right to left inside the SAME step, for e.g., a time step. Finally, formalized pure "forward" and pure "reverse" is often theoretical constructs. Just like hand-rolled code can beat generic optimizers, hand-rolled algorithmic differentiation code will be better –

even for Monte-Carlo sensitivity runs. However, development productivity gains to be attained by using automated AD tools are well documented.

Capriotti and Giles (2011) detail several techniques for the systematic design paradigm for using Algorithmic Differentiation for Path-wise Monte-Carlo Derivatives.

As indicated earlier, the cost associated with the forward and the reverse algorithmic is different.

- Forward Algorithmic Differentiation Cost => $\frac{Cost[B+F]}{Cost[B]} = [2,2.5]$
- Reverse Algorithmic Differentiation Cost => $\frac{Cost[B+F+R]}{Cost[B]} = [4,5]$
- B => Base; F => Forward; R => Reverse.

Algorithmic differentiation can be effectively used in conjunction with other related methods to improve performance. For instance algorithmic differentiation is natural performance fit in these situations (Kaebe, Maruhn, and Sachs (2009), Schlenkirch (2011)). Some approaches in this regard end up utilizing intermediate value theorem to facilitate the formulation (Christianson (1998), Giles and Pierce (2000)).

6) Bermudan Swap Option Sensitivities

In this section, we consider the formulation of the sensitivity evaluation for the Bermudan swap option sensitivities.

The details of an H x M Bermudan Swap Option are as follows:

- Define the M swap exercise/pay date tenor grids $T_0 < T_1 < ... < T_M$.
- Option exercise dates T_r start from date T_H onwards, i.e., $T_r \in \{T_H, T_{H+1}, ..., T_{M-1}\}$.
- The cash flow stream after the exercise is the payment stream $\vec{X} = \{X_r, X_{r+1}, ..., X_M\}$.

Given a fixed rate R, the ith cash flow constituting an exercised Bermudan swap is

$$X_i = N(T_i)\Delta(t_{i-1}, t_i)[L_i - R] \quad \textbf{(6.1)}$$

The Bermudan Swap PV is

$$PV_{Berm}(T_r) = \mathrm{E}\left[\sum_{i=r}^{M} D_f(t_i) X_i\right] \quad \textbf{(6.2)}$$

The following algorithm illustrates the Monte-Carlo methodology for valuing an H x M Bermudan Swap Valuation:

- Simulate a single path sequence of $\vec{L}$.
- For this path, evaluate $PV(X_i)$ for each $\vec{X} = \{X_r, X_{r+1}, ..., X_M\}$.
- For this path, generate a vector of $PV_{Berm}(T_p)$ corresponding to each possible exercise date $T_p \in \{T_H, T_{H+1}, ..., T_{M-1}\}$.
- Find T_r that maximizes $PV_{Berm}(T_p)$.
- Record $\{T_r, PV_{Berm}(T_r)\}$.

The analysis of Section 4 can be used in the estimation of the greeks.

Under certain conditions of regularity (Lipschitz continuity), the H x M Exercised Bermudan Swap Option Delta/Parameter Sensitivity can become an expectation of path derivative [Piterbarg (2004), Capriotti and Giles (2011)]

$$\frac{\partial PV_{Berm}(T_r)}{\partial L_K(0)} = \frac{\partial\left\{\mathrm{E}\left[\sum_{i=r}^{M} D_f(t_i)X_i\right]\right\}}{\partial L_K(0)} = \mathrm{E}\left[\sum_{i=r}^{M}\frac{\partial\{D_f(t_i)X_i\}}{\partial L_K(0)}\right] \quad \textbf{(6.3)}$$

$$\frac{\partial PV_{Berm}(T_r)}{\partial \alpha} = \frac{\partial\left\{\mathrm{E}\left[\sum_{i=r}^{M} D_f(t_i)X_i\right]\right\}}{\partial \alpha} = \mathrm{E}\left[\sum_{i=r}^{M}\frac{\partial\{D_f(t_i)X_i\}}{\partial \alpha}\right] \quad \textbf{(6.4)}$$

Leclerc, Liang, and Schneider (2009) formulate the individual Cash-flow PV and Greeks for the Bermudan swap options. First, the ith cash flow constituting an exercised Bermudan swap is

$$PV_j = D_f(t_j)\Delta(t_{j-1},t_j)[L_j - R] \quad \textbf{(6.5)}$$

The discount factor corresponding to the ith cash flow is

$$D_f(t_j) = \prod_{p=1}^{j}\frac{1}{1+\Delta(t_{p-1},t_p)L_p} \Rightarrow PV_j = \left\{\prod_{p=1}^{j}\frac{1}{1+\Delta(t_{p-1},t_p)L_p}\right\}\Delta(t_{j-1},t_j)[L_j - R] \quad \textbf{(6.6)}$$

Using the relation established earlier $\frac{\partial PV_j(t)}{\partial L_K(0)} = \sum_{i=1}^{n} \frac{\partial PV_j(t)}{\partial L_i(t)} \frac{\partial L_i(t)}{\partial L_K(0)}$ the delta is evaluated (the path derivatives estimator $\frac{\partial L_i(t)}{\partial L_K(0)}$ is given by the LMM formulation presented earlier).

Finally, the Cash-flow PV Delta ($\frac{\partial PV_j(t)}{\partial L_i(t)}$) is calculated as

$$\frac{\partial PV_j(t)}{\partial L_i(t)} = \frac{\partial}{\partial L_i(t)} \left[\left\{ \prod_{p=1}^{j} \frac{1}{1+\Delta(t_{p-1},t_p)L_p} \right\} \Delta(t_{j-1},t_j)[L_j - R] \right] \quad \textbf{(6.7)}$$

$$\frac{\partial PV_j(t)}{\partial L_i(t)}[j \geq i] = \left[\delta_{ij} - \frac{\Delta(t_{i-1},t_i)[L_j - R]}{1+\Delta(t_{i-1},t_i)L_i(t)} \right] \Delta(t_{j-1},t_j) D_f(t_j) \quad \textbf{(6.8)}$$

$$\frac{\partial PV_j(t)}{\partial L_i(t)}[j < i] = 0 \quad \textbf{(6.9)}$$

Credit Analytics uses the LSM Methodology to evaluate the optimal exercise dates.

Since the simple model of maximizing $PV_{Berm}(T_r)$ across T_r gets too cumbersome if the exercise dates are numerous – LSM based optimal exercise determination laid out in [Longstaff and Schwartz (2001)] can be used – regress T_r against $PV_{Berm}(T_r)$. Regression is then applied to Curve-Fit to extract the optimal exercise date.

LSM is highly effective in situations where the call schedules are continuous or fine-grained. Sampling is reduced to a few evenly spaced-out grid points – such that the full sample scoping is eliminated.

Any appropriate inter-nodal interpolating/splining technique to determine $PV_{Berm}(T_r)$ as a function of T_r is valid – e.g., constant $PV_{Berm}(T_r)$ over T_r, linear/quadratic/polynomial $PV_{Berm}(T_r)$ over T_r, or even exponential/hyperbolic tension spline-based $PV_{Berm}(T_r)$ over T_r.

7) NTD Basket Sensitivities

In this section, we consider the sensitivities associated with an nth-to-default basket. We start with describing the details of an NTD Product, followed by the formulation of its greeks.

Details of an NTD Product:

1. $p = 1 \to n$ => Number of Components
2. $j,k = 1 \to n$ => Row, column index of the correlation matrix for each of the n components
3. $l,m = 1 \to n$ => Factorized Cholesky diagonal matrix for the n components
4. r => rth component in the current draw of ordered default times; it corresponds to the current nth-to-default.
5. N => The "N" in NTD ($\tau_N \equiv \tau_r$).

The NTD contract contains 3 components, each of which is valued separately during a given Monte-Carlo run.

$$V_{NTD} = V_{Loss} + V_{\Pr emium} + V_{Accrued} \quad \textbf{(7.1)}$$

The PV of the loss leg is given as

$$V_{Loss} = [1 - R_r(\tau_r)]D_f(\tau_r)N(\tau_r) \quad \textbf{(7.2)}$$

The PV of the premium and the accrual legs are

$$V_{\Pr emium} = c\sum_{i=1}^{n} N(t_i)D_f(t_i)\Delta(t_{i-1},t_i) \perp (t_i \le \tau_r) \quad \textbf{(7.3)}$$

$$V_{Accrual} = c\sum_{i=1}^{n} N(\tau_r) D_f(\tau_r) \Delta(t_{i-1}, \tau_r) \perp (t_{i-1} \le \tau_r) \perp (t_i \ge \tau_r) \quad \textbf{(7.4)}$$

Here $\perp(t \le \tau)$ be the default indicator that is 1 if $t \le \tau$, and 0 otherwise.

To make the computation convenient [Capriotti and Giles (2010), Capriotti and Giles (2011), Giles (2009), Chen and Glasserman (2008)] $\perp(t \le \tau)$ is regularized and smeared out using an appropriate proxy, i.e., $\perp(t \le \tau) \cong \mathrm{H}(t \le \tau)$. One choice for $\mathrm{H}(t \le \tau)$ is the Heaviside function – it has a bias, but it can be designed to be much tighter than the Monte-Carlo accuracy.

Using the formulations of Sections 4) and 5), the NTD Sensitivity can be computed as follows:

$$\frac{\partial V_{NTD}}{\partial \rho_{jk}} = \sum_{p=1}^{n} \frac{\partial V_{NTD}}{\partial \tau_p} \frac{\partial \tau_p}{\partial \rho_{jk}} \quad \textbf{(7.5)}$$

The NTD path derivatives estimator is given as

$$\frac{\partial V_{NTD}}{\partial \tau_p} = \frac{\partial V_{Loss}}{\partial \tau_p} + \frac{\partial V_{\mathrm{Pr}emium}}{\partial \tau_p} + \frac{\partial V_{Accrued}}{\partial \tau_p} \quad \textbf{(7.6)}$$

The NTD loss path derivatives estimator is given as

$$\frac{\partial V_{Loss}}{\partial \tau_p} = \delta_{rp} \frac{\partial \{[1 - R_r(\tau_r)] D_f(\tau_r) N(\tau_r)\}}{\partial \tau_p} \quad \textbf{(7.7)}$$

The NTD premium path derivatives estimator is given as

$$\frac{\partial V_{\text{Pr}\,emium}}{\partial \tau_p} = c\delta_{rp}\sum_{i=1}^{n} N(t_i)D_f(t_i)\Delta(t_{i-1},t_i)\frac{\partial \text{H}(t_i \le \tau_r)}{\partial \tau_p} \quad \textbf{(7.8)}$$

$$V_{Accrual} = c\delta_{rp}\sum_{i=1}^{n} \frac{\partial\{N(\tau_r)D_f(\tau_r)\Delta(t_{i-1},\tau_r)\text{H}(t_{i-1} \le \tau_r)\text{H}(t_i \ge \tau_r)\}}{\partial \tau_p} \quad \textbf{(7.9)}$$

8) Basket Options

In this section we consider the sensitivity generation for Basket Options. The PV for an Option Basket may be determined from the Black Scholes relation as:

$$V = D_f(T)\sum_{i=1}^{n}\left[W_i X_i(T) - S\right]^+ \quad \textbf{(8.1)}$$

As before, the formulations in Sections 4) and 5) may be used to evaluate the sensitivities.

Recall from earlier that $\frac{\partial V(t)}{\partial x_k(0)} = \sum_{j=1}^{n}\frac{\partial V(t)}{\partial x_j(t)}\frac{\partial x_j(t)}{\partial x_k(0)}$. Here:

- $t = T$
- $V(t) = V_{BO}(T)$

The Basket Options path derivatives estimator is then given as

$$\frac{\partial V_{BO}\left(\vec{X}(T),T\right)}{\partial X_i(T)} = \frac{\partial D_f(T)}{\partial X_i(T)}\left[\sum_{p=1}^{n} W_p X_p(T) - S\right]^+ + W_i D_f(T) * Black\_Scholes\_Delta\left(Strike_p, T\right) \quad \textbf{(8.2)}$$

The strike is given as

$$Strike_p = \frac{S - \sum_{p\neq i,\, p=1}^{n} W_p X_p(T)}{W_i} \quad \textbf{(8.3)}$$

References

- Bartholomew-Biggs, M., S. Brown, B. Christianson, and L. Dixon (2000): Automatic Differentiation of Algorithms. *Journal of Computational and Applied Mathematics* **124** (2000): 171-190.
- Berland, H (2006): *Automatic Differentiation*.
- Berz, M., et al. (1996): Computational Differentiation: Techniques, Applications and Tools, *Society for Industrial and Applied Mathematics*, Philadelphia, PA.
- Bischof, C, P Hovland, and B Norris (2005): *On the Implementation of Automatic Differentiation Tools*.
- Brace, A., D. Gatarek, and M. Musiela (1997): The Market Model of Interest-Rate Dynamics. *Mathematical Finance* **7**: 127-155.
- Brigo, D., and F. Mercurio (2001): *Interest-Rate Models: Theory and Practice*, **Springer-Verlag**.
- Broadie, M., and M. Glasserman (1996): Estimating Security Derivative Prices Using Simulation. *Management Science* **42**: 269-285.
- Capriotti, L. (2011): Fast Greeks by Algorithmic Differentiation. *Journal of Computational Finance* **14(3)**: 3-35.
- Capriotti, L., and M. Giles (2010): Fast Correlation Greeks by Adjoint Algorithmic Differentiation. *Risk* (2010): 79-83.
- Capriotti, L., and M. Giles (2011): Algorithmic Differentiation: Adjoint Greeks Made Easy.
- Chen, Z., and P. Glasserman (2008): Sensitivity Estimates for Portfolio Credit Derivatives using Monte-Carlo. *Finance and Stochastics* **12 (4)**: 507-540.
- Christianson, B. (1998). Reverse Accumulation and Implicit Functions. *Optimization Methods and Software* **9 (4):** 307-322.
- Denson, N., and M. Joshi (2009a): Fast and Accurate Greeks for the LIBOR Market Model, *Journal of Computational Finance* **14 (4):** 115-140.
- Denson, N., and M. Joshi (2009b): Flaming Logs, *Wilmott Journal* **1:** 5-6.

- Ghaffari, H, J Li, Y Li, and Z Nie (2007): *Automatic Differentiation*.
- Giles, M. (2007): Monte Carlo Evaluation of Sensitivities in Computational Finance. *Proceedings of the Eighth HERCMA Conference*.
- Giles, M. (2009): Vibrato Monte-Carlo Sensitivities, *Monte-Carlo and Quasi Monte-Carlo Methods 2008,* P. L'Ecuyer, and Owen, A., editors, **Springer**, New York.
- Giles, M., and P. Glasserman (2006): Smoking Adjoints: fast Monte-Carlo Greeks. *Risk* (2006): 92-96.
- Giles, M., and N. Pierce (2000): An introduction to the adjoint approach to design. *Flow, Turbulence, and Control* **65**: 393-415.
- Glasserman, P. (2004): *Monte-Carlo Methods in Financial Engineering*, **Springer-Verlag**, New York.
- Glasserman, P., and X. Zhao (1999): Fast Greeks by Simulation in Foreard Libor Models. *Journal of Computational Finance* **3**: 5-39.
- Griewank, A. (2000): *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation,* **Society for Industrial and Applied Mathematics**, Philadelphia.
- Harrison, J., and D. Kreps (1979): Martingales and Arbitrage in multi-period Securities Markets. *Journal of Economic Theory* **20 (3)**: 381-408.
- Iri, M (1991): History of Automatic Differentiation and Rounding Error Estimation, in: A. Griewank, G. Corliss (Eds.), Automatic Differentiation of Algorithms, *Society for Industrial and Applied Mathematics*, Philadelphia, PA, 3-16.
- Jamshidian, F. (1997): LIBOR and Swap Market Models and Measures. *Finance and Stochastics* **1**: 293-330.
- Kaebe, C., J. Maruhn, and E. Sachs (2009). Adjoint based Monte-Carlo Calibration of Financial Market Models. *Finance and Stochastics* **13 (3):** 351-379.
- Krishnamurthy, L. (2012): *A Sample Calibration Framework.*
- Kallenberg, O. (1997): *Foundations of Modern Probability Theory,* **Springer**, New York.
- Kunita, H. (1990): *Stochastic Flows and Stochastic Differential Equations,* **Cambridge University Press**.

- Leclerc, M., Q. Liang, and I. Schneider (2009): Fast Monte-Carlo Bermudan Greeks. *Risk* (2009): 84-88.
- Longstaff, F., and E. Schwartz (2001): Valuing American Options by Simulation: A Simple Least-Squares Approach. *Review of Financial Studies* **14**: 113-147.
- Naumann, U (2008): Optimal Jacobian accumulation is NP-complete. *Mathematical Programming* **112** (2): 427–441.
- Piterbarg, V. (2004): Computing deltas of callable LIBOR exotics in Forward LIBOR Models. *Journal of Computational Finance* **7(3)**: 107-144.
- Protter, P. (1990): *Stochastic Integration and Differential Equations*, **Springer-Verlag**, Berlin.
- Schlenkirch, S. (2011). Efficient Calibration of the Hull-White Model. *Optimal Control Applications and Methods* **33 (3):** 352-362.
- Smith, S. (1995): Differentiation of the Cholesky Algorithm. *Journal of Computational and Graphical Statistics* **4 (2)**: 134-147.
- Wengert, R (1964): A Simple Automatic Derivative Evaluation Program. *Communications of the ACM* **7**: 463–464.

Figure 1: Optimal Intermediate Wengert Variable

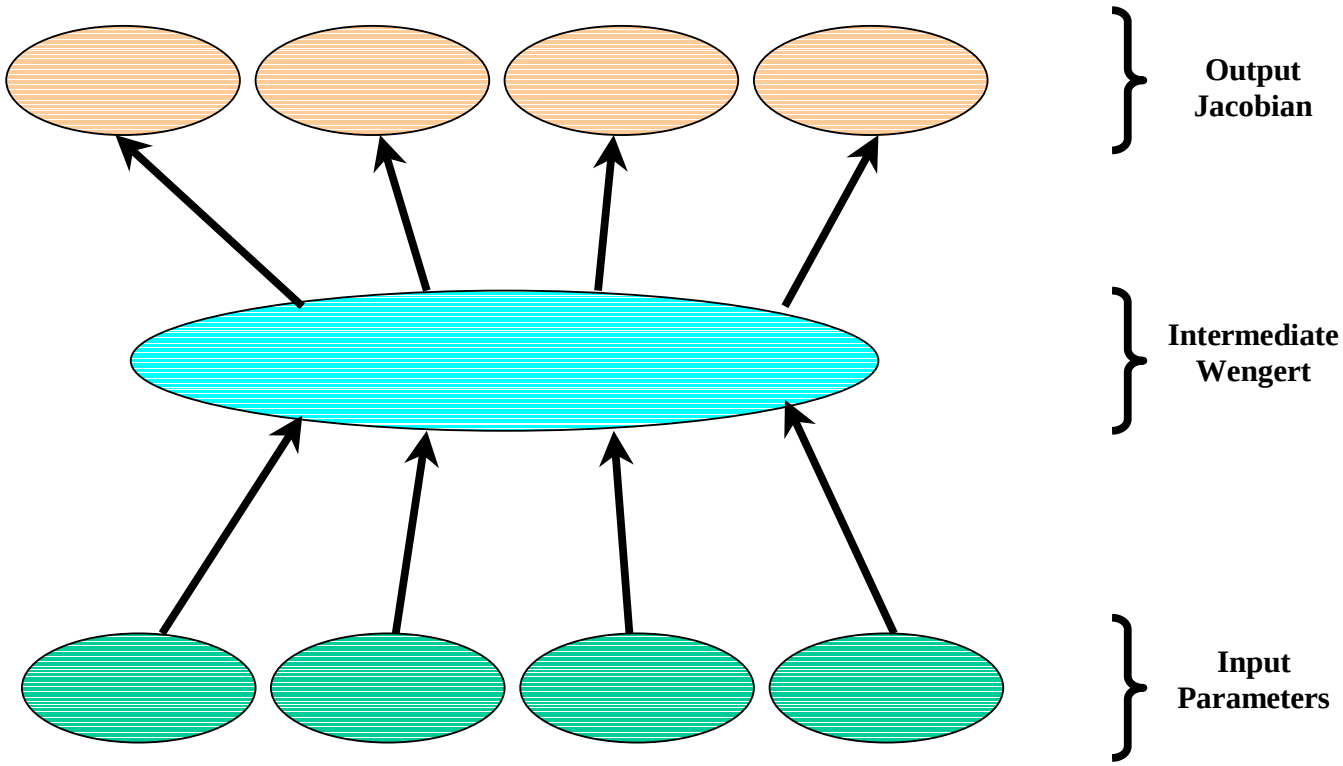

Figure 2: Computation Financial Object Scheme

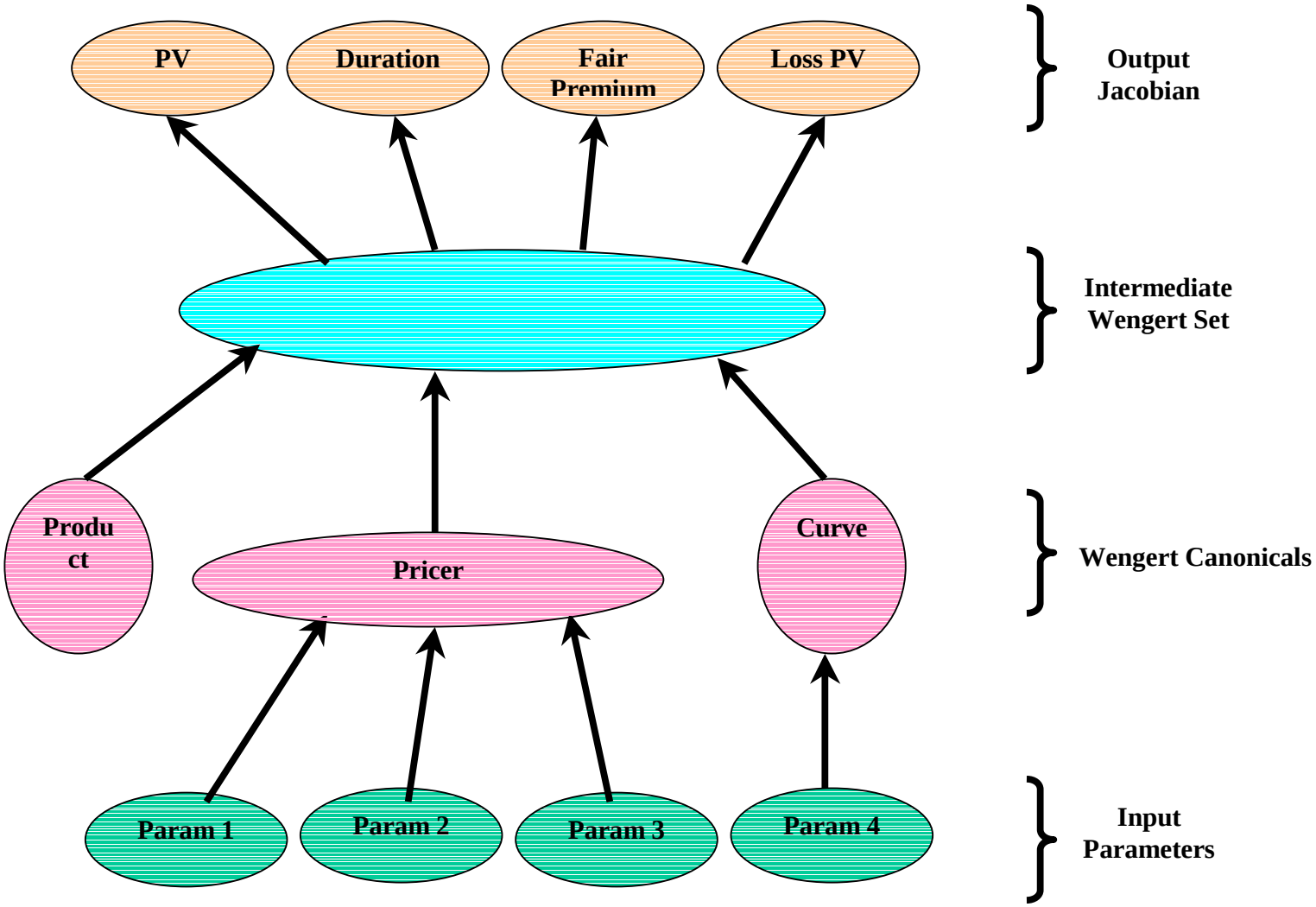

Figure 3: Wengert Fan-in and fan-out