Stochastic Analyzer in DRIP

Lakshmi Krishnamurthy

v0.41, 20 January 2014

Introduction

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.

Overview

1. AD History: Iri (1991)
2. Mathematical Foundations: Griewank (2000)
3. Survey: Berz (1996)
4. Implementation Tools, Methodologies, Processes, and Techniques (Bischof, Hovland, and Norris (2005))
5. AD Resource: *http://www.autodiff.org/*

Algorithmic Differentiation in Finance

1. Focus has been primarily on Monte-Carlo methodologies.
2. 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.
3. 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)).
4. Capriotti (2011) covers automated Greek generation, but with a focus on automatic differentiation, and in the context of Monte-Carlo methods.
5. Finally, algorithmic differentiation has also been applied to addressing the issue of calibration along with sensitivity generation (Schlenkirch (2011)).

Document Layout

1. Introduction

Algorithmic Differentiation - Basics

Motivation and Advantages

1. Definition: Automatic 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)).
2. Symbolic Derivatives: Calculate the local symbolic derivatives rather than the a) divided differences, or b) numerical differentials (*Automatic Differentiation - Wikipedia Entry*).
3. Calculation Speed: Same number of Objective Function Calculation as the original; however, potential "chain rule" multiplication factor effects.
4. Accuracy vs. Performance: Due to the usage of symbolics, accuracy of Automatic Differentiation always better than numerical differentials; however, due to the chain-rule issue, may not be always faster.
5. Scalability at infinitesimal variates: Since Automatic Differentiation is always symbolic and therefore infinitesimal, it will automatically scale to arbitrarily small variate infinitesimals – reduced errors due to bit cancellation etc:
6. Higher-order derivatives: Automatic Differentiation does not need additional objective function evaluations for higher order derivative calculations (beyond the chain-rule issues); therefore, those are infinitesimally correct too.

Program Sequence Construction Modes

1. Forward Automatic Differentiation: Express the final and the intermediate variables as a consequence of a computed forward graph, and derive the symbolic forward derivative graph.
    - Effectively computes the gradient of the intermediate variables to the variates or the "independent variables" and transmits them up the graph.
2. Reverse Automatic Differentiation: Express the intermediate variables and the input variates as nodes in the computed reverse graph, and derive the symbolic reverse derivative graph.
    - Often may still need the forward path to store the calculated intermediates needed on the way back.
    - Effectively computes the gradient of the intermediate variables to the "dependent variables" and transmits them down the graph.
3. Speed:
    - Forward Mode => Speed proportional to n, the number of "independent" variables
    - Reverse Mode => Speed proportional to m, the number of "dependent" variables
4. Memory Usage (Ghaffari, Li, Li, and Nie (2007)):
    - Forward Mode => a) Each Wengert variable, b) Forward Jacobian for each Wengert, c) Forward Dependency Graph
    - Reverse Mode => a) Each Wengert Adjoint, b) Reverse Jacobian for each Wengert, c) Forward/Backward Dependency graph
5. When the difference is minimal: When 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 Automatic Differentiation is an equally good choice for sensitivity generation, owing to its performance.

Canonicalization - Program Statements Simplification by Decomposition

1. Program Line-level decomposition: Canonicalization decomposes the program/statement units into specific analysis bits.
   - Canonicalization is commonly used in many areas of computer science, e.g., in compiler design/code generation, SKU formulation/synthesis/customization etc.
2. Canonicalization Implementation: In general, canonicalization and other related Automatic Differentiation Source Code generation/transformation techniques should go hand in hand with optimizing compiled code emission techniques, program active variable activity analysis.
   - Canonicalization sequence should include steps (Bischof, Hovland, and Norris (2005)) where you would be able to mark the mathematical "Automatically Differentiable" code segments to separate from the others during, for instance, pre-processing etc.
   - For true program transformation effectiveness, Hot-Spot type dynamic run-time analysis is needed in addition to static compile time data flow analysis etc.
   - In VM oriented languages like Java, the run-time GC already works, so would it might make a candidate for embedding AD execution/selective sensitivity generation in.
3. Equivalence with Wengert Structuring: 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 DAG linearization.
4. Limitations with the implementation: For many of the reasons above, 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.

5. Post canonicalized Enhancement Cost: Given that the worst case operation 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 automatic differentiation execution unit costs. Typically due to "weird" functions, the worst-case addition to a single post-canonicalized statement is a factor of 5, not 4.
6. Divided Differences based Differentiation Fall back:

$$\frac{\Delta^n y}{\Delta x^n} = \sum_{i=0}^{n} \frac{(-1)^i C_i^n y[x + \delta(n - 2i)]}{(2\delta)^n}.$$

Challenges of Automating the Differentiation

1. Deep-dig perspective: Re-purposed Automatic Differentiation perspective forces the visualization of the computation at the granularity of the symbolic functional forms of the objective function.
    a. Objective Function evaluator over-loading => This requires propagation of the inner most symbolic graph nodes through the graph chain => causes additional cognitive SKU export!!
    b. Objective Function Neighborhood Behavior => 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.
2. Block-level View Fixation: Source code transformation techniques are very invasive, and require highly locally frozen view fixation, and are therefore less cognitive. Operator overloading techniques enable retention of the domain focus, and are therefore more cognitive.
    a. 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 may not quite do it.

3. Complied language Automatic Differentiation implementation: 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.
    a. Further, compiled language source code transformation appears to be a vestige of “smart compiler” efforts of the ‘90s – classic instance of where a simple idea is “intellectually transmitted” than “built out-of the-box”.
4. Symbolic Differentiation Challenges with certain Unit Functional Forms: When you consider functions such as $y = \frac{1}{f(x)}$, and you seek $\frac{dy}{dx}$ symbolically, the higher order symbolic differentiations become much more challenging - $\frac{dy}{dx} = -\frac{1}{f^2(x)}\frac{df(x)}{dx}$, $\frac{d^2y}{dx^2} = \frac{2}{f^3(x)}\frac{df(x)}{dx} - \frac{1}{f^2(x)}\frac{d^2f(x)}{d^2x}$ and so on for higher orders. Thus symbolically handling these series this way gets out of control fast!

Wengert Representation and Optimal Program Structure Synthesis

1. Combination of Forward/Reverse Modes: Forward (n inputs) and reverse (m outputs) mode represent just two possible (extreme) ways of recursing through the chain rule. For n > 1 and m > 1 there is a golden mean, but finding the optimal way is probably an NP-hard problem (Berland (2006)) – optimal Jacobian accumulation is NP-complete (Naumann (2008)).
2. Wengert Intermediate Fan-in and possibly fan-out: See Figures 1 to 3 for illustrate this.
    - Wengert Intermediate Performance Enhancement => 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).
    - Reusable Intermediate Performance Improvement => 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.

- Wengert Funneling Criterion => For non-optimizing, non-parsimonized Wengert funnels, $\frac{dP_i}{dMI_j} \to \delta_{ij}$ for the Wengert fan to be a funneling fan-in – otherwise rippling out causes huge non-diagonal Markov matrices. This is true for a) I -> P, b) P-> W, and c) W -> O.

3. Standardized Computational Finance Structures: In computational finance (esp. computational fixed income finance), the payout/product/pricer object serves the function of the intermediate Wengert variate indicated above. From below this variate you have the inputs/parameters rippling up, and from above you have the Jacobians/output measure adjoints feeding down (Figure 2).
   - Reactive Tree Upticks => 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.
   - Automatic Differentiation for the Wengert Canonicals => This involves the following:
     a. Identifying the abstractable financial canonical/reusable common object structures (market parameters, product parameters, pricer parameters, etc.)
     b. Working out their forward differentials and the reverse adjoints.
   - One Financial Automatic Differentiation view => The Intermediate Wengert Canonical View is the conceptual parsimonisation of the variate parameters space and the Jacobian measure space.

Optimization using Pre-accumulation and Check Pointing

1. Pre-accumulation: Aggregation (and possibly caching) of 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).

   a. Pre-accumulation also provides a suitable boundary for parallelization.
   b. It may also be looked at as the appropriate edge at which the source code transformation technique and operator overloading technique may "merge".

2. Cross-country Accumulation: Same as pre-accumulation, but pre- accumulation occurs in a specified (forward/reverse), Cross-country accumulation need not – in fact it may be guided by program analysis using Optimal Wengert intermediate composition techniques.
   a. This is also referred to as check pointing.
   b. This typically also requires snapshotting the program global and other execution context parameters at the checkpoint boundaries.
   c. Works best when the program state is easily and minimally savable, and quickly recoverable.
   d. Will also work well in conjunction with traditional kernel level check pointing schemes for fail-over etc:

Algorithmic Differentiation Financial Application Space Customization

1. Math Modules:

- Forward differentials and auto-adjointing of math modules => May be needed for most of them.
- Every block, compute the base "value", forward differential, and reverse adjoint.
- In fact, for every active double-precision variable v, source code transformation automatic differentiation techniques recursively automatically generate the doublet $\left(v, \dot{v}\right)$. Further, this calculation may also be parallelized.
    - This particular calculation may also be propagated at the function call level, so that the assignment outputs are automatically generated for the doublet/multiple.

    - Computational structures => Design restrictions may also be imposed by the computability of the AD of a math module, i.e., would the financial **MarketParamsContainer** be broken down into further parameter units?

2. <u>Stochastic Variate Automatic Differentiation</u>: 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).
    - Variance Reduction along the forward path => 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.
    - Delta effects on the Optimal Exercise Dates => This imposes restrictions on how the path derivatives maybe computed using automatic differentiation. This may also be used in conjunction with regression analysis for estimating optimal exercise times. That certainly enables adjoint automatic differentiation techniques to be used.
    - Tangent multi-mode arc derivatives =>
        a. Identifying the circumstances under which they are re-usable
        b. Arc derivatives extraction intermediates may also be re-used
        c. Depends (as always) on the speed up and memory used.
3. <u>Quasi-analytic Computation Models</u>: No Monte-Carlo evolution needed at all, but still Wengert intermediate level reformulation necessary to enhance the quasi-analytics analysis (e.g., Copula methods).
    - Adjoint-Natural Formulation Mode => Typical formulation works out the Wengerts backwards from the final measure (e.g., say from PV), so they are automatically amenable to the adjoint mode of automatic differentiation.
4. <u>Latent State Calibration from Observed Manifest Measures:</u>
    - Formulation of the de-convolution the latent state from the observed manifest measure is necessary for the extraction of the latent state parameter set (this is accomplished by the calibration process).

- Of course, latent state calibration occurs among the elastic and the inelastic dimensions, and the inelastics are parameter set!
- Latent state calibration/parameterization etc: inherently involve parsimonization – this is where the models come in.

Sensitivity Generation During Curve Construction

Introduction

1. Advantages: In addition to the usual advantage that Automatic Differentiation provides on doing accurate Greeks on the same run as pricing, there is no need for multiple bumped curves anymore – but the proper Jacobians need to be calculated.
    - Further speed up => The segment micro-Jacobian needs to be pre-calculated right during the calibration - we need to calculate the Jacobian $\frac{\partial C_i}{\partial q_j}$, where C_i is the i^th^ coefficient, and q_j is the j^th^ input.
2. Curve Calibration Deltas: Typical deltas are with respect to the
    - dynamical latent state stochastic variates (e.g., the forward rates)
    - calibrated parameters (e.g., the segment spline coefficients)
    - unit change in the quoted instrument measures (e.g., 1 bp change) - here the Jacobians need to ripple upwards from the quoted instrument manifest measures.
3. Span/Segment Elastic Variates: Consider the situation where the latent state in itself (not its transformation) is explicitly measured. There are 5 different kinds of latent state proxies to consider:
    - Φ => Span stochastic latent state evolution variate.
    - Φ_k => Stochastic latent state evolution variate for segment k.
    - ϕ => Implied Span Quoted Instrument Manifest Measure.
    - ϕ_k => Implied Quoted Instrument Manifest Measure for Segment k.
    - φ_k => Observed Quoted Instrument Manifest Measure for Segment k at precisely a single variate point – typically, the observations are done at the anterior/posterior terminal ends of the segment.
4. Span/Segment variate relations: For a given calculated/formulated output manifest measure Ξ, the following are true by definition:

- $\Phi_k(t = t_k) = \Phi(t = t_k) \Rightarrow \left|\frac{\partial \Xi}{\partial \Phi}\right|_{t=t_k} = \left|\frac{\partial \Xi}{\partial \Phi_k}\right|_{t=t_k}$
- $\varphi_k = \phi_k(t = t_k) = \phi(t = t_k) \Rightarrow \frac{\partial \Xi}{\partial f_k} = \left|\frac{\partial \Xi}{\partial \phi}\right|_{t=t_k} = \left|\frac{\partial \Xi}{\partial \phi_k}\right|_{t=t_k}$

5. <u>Sensitivities to the elastic variates</u>:
   - 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
6. <u>Piece-wise constant segment variate</u>: In this case, $\frac{\partial \Xi}{\partial \Phi_k} = \frac{\partial \Xi}{\partial \phi_k} = \frac{\partial \Xi}{\partial \varphi_k}$.
7. <u>Splined segment variate</u>: 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 $\frac{\partial \Xi}{\partial \Phi_k} \neq \frac{\partial \Xi}{\partial \phi_k}$.
   - Stochastic Evolution Variate Derivative => For the case where Ξ refers to the discount factor, it can be shown that

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

     Thus, $\frac{\partial D_F(t)}{\partial \Phi_k} = -D_F(t) * \begin{Bmatrix} t_{k+1} - t_k \; for k < j \\ t - t_k \; for k = j \\ 0 \; for k > j \end{Bmatrix}$

- Quoted Instrument Manifest Measure Derivative => This depends on the actual details of the quadrature. Thus, $\frac{\partial D_F(t)}{\partial \phi_k} = -D_F(t) * \left\{ \begin{array}{l} \int_{t_k}^{t_{k+1}} \frac{\partial \Phi}{\partial \varphi_k} fork < j \\ \int_{t_k}^{t} \frac{\partial \Phi}{\partial \varphi_k} fork = j \\ 0\, fork > j \end{array} \right\}$

8. Linear Dependence of Integrand Quadrature: For many functional formulations in finance, 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

1. Representation Jacobian: Every Curve implementation needs to generate the Jacobian of the following latent state metric using its corresponding latent state quantification metric:
   - Forward Rate Jacobian to Quote Manifest Measure
   - Discount Factor Jacobian to Quote Manifest Measure
   - Zero Rate Jacobian to Quote Manifest Measure
2. Calibration Jacobian vs. Monte-Carlo Automatic Differentiation Delta: Both of these are actually path-wise, the difference being that:
   - Jacobian generated during calibration is part of inference, therefore iterative.
   - Jacobian of Monte-Carlo Automatic Differentiation is typically path-wise and non-iterative, therefore it is technically part of prediction.

3. Importance of the representation Self-Jacobian: Representation 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.

4. Forward Rate->DF Jacobian:
   - $F(t_A,t_B)=\frac{1}{t_B-t_A}\ln\left(\frac{\partial D_f(t_A)}{\partial D_f(t_B)}\right)$.
   - $\frac{\partial F(t_A,t_B)}{\partial D_f(t_k)}=\frac{1}{t_B-t_A}\left\{\frac{1}{D_f(t_A)}\frac{\partial D_f(t_A)}{\partial D_f(t_k)}-\frac{1}{D_f(t_B)}\frac{\partial D_f(t_B)}{\partial D_f(t_k)}\right\}$.
   - $F(t_A,t_B)$ => Forward rate between times t_A and t_B.
   - $D_f(t_k)$ => Discount Factor at time t_k

5. Zero Rate to Forward Rate Equivalence: This equivalence may be used to construct the Zero Rate Jacobian From the Forward Rate Jacobian. Thus the above equation may be used to extract the Zero Rate micro-Jacobian.

6. Zero Rate->DF Jacobian:
   - $\frac{\partial Z(t)}{\partial D_f(t_k)}=\frac{1}{t-t_0}\left\{\frac{1}{D_f(t)}\frac{\partial D_f(t)}{\partial D_f(t_k)}\right\}$
   - $Z(t)$ => Zero rate at time t

7. Quote->Zero Rate Jacobian:
   - $\frac{\partial Q_j(t)}{\partial Z(t_k)}=(t_k-t_0)\left\{D_f(t_k)\frac{\partial Q_j(t)}{\partial D_f(t_k)}\right\}$
   - $Z(t)$ => Zero rate at time t

8. PV->Quote Jacobian:
   - $\frac{\partial PV_j(t)}{\partial Q_k}=\sum_{i=1}^{n}\left\{\frac{\partial PV_j(t)}{\partial D_f(t_i)}\div\frac{\partial Q_j(t)}{\partial D_f(t_i)}\right\}$

9. Cash Rate DF micro-Jacobian:
   - $\frac{\partial r_j}{\partial D_f(t_k)}=-\frac{1}{\partial D_f(t_j)}\frac{1}{t_j-t_{START}}\frac{\partial D_f(t_j)}{\partial D_f(t_k)}$

- r_j => Cash Rate Quote for the jth Cash instrument.
- $D_f(t_j)$ => Discount Factor at time t_j

10. <u>Cash Instrument PV-DF micro-Jacobian</u>:

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

11. <u>Euro-dollar Future DF micro-Jacobian</u>:

- $$\frac{\partial Q_j}{\partial D_f(t_k)} = \frac{\partial D_f(t_j)}{\partial D_f(t_k)}\frac{1}{\partial D_f(t_{j,START})} - \frac{D_f(t_j)}{D_f^{\,2}(t_{j,START})}\frac{\partial D_f(t_{j,START})}{\partial D_f(t_k)}$$
- Q_j => Quote for the jth EDF with start date of $t_{j,START}$ and maturity of t_j.

12. <u>Euro-dollar Future PV-DF micro-Jacobian</u>:

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

13. <u>Interest Rate Swap DF micro-Jacobian</u>:

- $Q_j DV01_j = PV_{Floating,j}$
- Q_j => Quote for the jth IRS maturing at t_j.
- $DV01_j$ => DV01 of the swap
- $PV_{Floating,j}$ => Floating PV of the swap
- $$\frac{\partial\lfloor Q_j DV01_j\rfloor}{\partial D_f(t_k)} = \frac{\partial\lfloor PV_{Floating,j}\rfloor}{\partial D_f(t_k)}$$
- $$\frac{\partial\lfloor Q_j DV01_j\rfloor}{\partial D_f(t_k)} = \frac{\partial Q_j}{\partial D_f(t_k)}DV01_j + Q_j\frac{dDV01_j}{\partial D_f(t_k)}$$

- $\frac{dDV01_j}{\partial D_f(t_k)} = \sum_{i=1}^{j} N(t_i)\Delta_i \frac{\partial D_f(t_i)}{\partial D_f(t_k)}$
- $PV_{Floating,j} = \sum_{i=1}^{j} l_i N(t_i)\Delta_i D_f(t_i)$
- $\frac{\partial PV_{Floating,j}}{\partial D_f(t_k)} = \sum_{i=1}^{j} N(t_i)\Delta_i D_f(t_i)\frac{\partial l_i}{\partial D_f(t_k)} + \sum_{i=1}^{j} l_i N(t_i)\Delta_i \frac{\partial D_f(t_i)}{\partial D_f(t_k)}$

14. <u>Interest Rate Swap PV-DF micro-Jacobian</u>:
    - $\frac{\partial PV_{IRS,j}}{\partial D_f(t_k)} = \sum_{i=1}^{j} N(t_i)\Delta(t_{i-1},t_i)\left\{(c_j - l_i)\frac{\partial D_f(t_i)}{\partial D_f(t_k)} - D_f(t_i)\frac{\partial l_i}{\partial D_f(t_k)}\right\}$
    - 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.
15. <u>Credit Default Swap DF micro-Jacobian</u>:
    - $PV_{CDS,j} = PV_{Coupon,j} - PV_{LOSS,j} + PV_{ACCRUED,j}$
    - 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
    - $PV_{Coupon,j} = c_j \sum_{i=1}^{j} N(t_i)\Delta_i S_P(t_i) D_f(t_i)$
    - $\frac{\partial PV_{Coupon,j}}{\partial D_f(t_k)} = c_j \sum_{i=1}^{j} N(t_i)\Delta_i S_P(t_i)\frac{\partial D_f(t_i)}{\partial D_f(t_k)} + \frac{\partial c_j}{\partial D_f(t_k)} \sum_{i=1}^{j} N(t_i)\Delta_i S_P(t_i) D_f(t_i)$
    - $PV_{LOSS,j} = \int_0^{t_j} N(t)[1 - R(t)] D_f(t)\, dS_P(t)$
    - $\frac{\partial PV_{LOSS,j}}{\partial D_f(t_k)} = \int_0^{t_j} N(t)[1 - R(t)] \frac{\partial D_f(t)}{\partial D_f(t_k)} dS_P(t)$

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

16. <u>Credit Default Swap DF micro-Jacobian</u>:

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

- 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.

Stochastic Entity Evolution

Stochastic Entity Evolution – Sensitivity Formulation

1. Evolution Dynamics: Simplest evolution of stochastic variables $L_i(t)$ will be ones with constant forward volatilities. Once the dynamics is formulated according to $\Delta L_i(t) = \mu_i(L_i, t)\Delta t + \sum_j \sigma_{ij}(L_i, t)\Delta W_j$ where $\mu_i(L_i, t)$ is the component drift, and σ_{ij} is the component co-variance to the factor $W_j(L_i, t)$, subsequent evolution can be determined.
   - The Eulerized version of the above is: $\Delta x_j(t) = h\mu_j\left(\vec{x}, t\right) + \sqrt{h}\sum_l \sigma_{jl}\left(\vec{x}, t\right)\Delta Z_l$, 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.
2. Evolution of the derivative entity: Once the stochastic variate dynamics is established, the dynamics of the observed derivative entity can be progressively determined.
3. Derivative Entity Measure path-wise evolution: Evolution sequence can be determined for the individual pay-off measures as well. These measures may further be dependent on the differentials of the derivative entity, so those may also need to be evolved using automatic differentiation.
4. Derivative Entity Computational efficiency enhancement:
   - Using the adjoint automatic differentiation methods
   - Using optimal combination of forward and adjoint automatic 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 blind delta bumps that may happen when computing differentials).
   - Quasi-analytic computation models and automatic differentiation techniques => No Monte-Carlo evolution needed at all, but still Wengert intermediate level

reformulation necessary to enhance the quasi-analytics analysis (e.g., Copula methods).

5. Derivative Entity Measure Calculation: [Instrument, Manifest Measure] input to [Instrument, Manifest Measure] output is equivalently maintained in the Jacobian. Alternately, the computation may also hold [Latent State calibrated parameter] to [Instrument, Manifest Measure] Output map.

Sensitivities to Stochastic State Variates and Dynamical Parameters

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

2. Dynamic Parameters**:** 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:

3. Segment/Span Coefficients: These are the additional coefficients serve act as the interpolated “PROXY” for the segment latent state at the unobserved points in the segment.
   - Sensitivities may also be sought to the coefficients.

Stochastic Variate Evolution Constrained by Splines

1. 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
2. For e.g., 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.

3. 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.

Formulation of the Evolution of Stochastic Variate Self-Jacobian

1. Evolution Formulation: $\Delta x_j(t) = \mu_j(x_1 ... x_n, t)\Delta t + \sum_{l=1}^{m} \sigma_{jl}(x_1 ... x_n, t)\Delta W_l(t)$
2. Definition of Self-Jacobian Delta: $J_{ij} = \frac{\partial x_i(t)}{\partial x_j(0)}$
3. Evolution Sensitivity Formulation:
   a. i => Index over the number of underliers (1 to n)
   b. l => Index over the number of independent stochastic factors (1 to m)
   c. Then $\frac{\partial \Delta x_j(t)}{\partial x_K(0)} = \Delta t \left[\sum_{i=1}^{n} \frac{\partial \mu_j(x_1 ... x_n, t)}{\partial x_i(t)} \frac{\partial x_i(t)}{\partial x_K(0)} \right] + \sum_{l=1}^{m} \Delta W_l(t) \left[\sum_{i=1}^{n} \frac{\partial \sigma_{jl}(x_1 ... x_n, t)}{\partial x_i(t)} \frac{\partial x_i(t)}{\partial x_K(0)} \right]$
   d. Eulerized version of the above is:
   $$\frac{\partial \Delta x_j(t)}{\partial x_K(0)} = h \left[\sum_{i=1}^{n} \frac{\partial \mu_j(x_1 ... x_n, t)}{\partial x_i(t)} \frac{\partial x_i(t)}{\partial x_K(0)} \right] + \sqrt{h} \sum_{l=1}^{m} \mathrm{Z}_l(t) \left[\sum_{i=1}^{n} \frac{\partial \sigma_{jl}(x_1 ... x_n, t)}{\partial x_i(t)} \frac{\partial x_i(t)}{\partial x_K(0)} \right]$$
   e. Re-write #1:
   $$\frac{\partial x_j(t+h)}{\partial x_k(0)} = \sum_{i=1}^{n} \left[\delta_{ji} + h \frac{\partial \mu_j(t)}{\partial x_i(t)} + \sqrt{h} \sum_{l=1}^{m} \left\{ \mathrm{Z}_l(t+h) \frac{\partial \sigma_{jl}(t)}{\partial x_i(t)} \right\} \right] \frac{\partial x_i(t)}{\partial x_k(0)} = \sum_{i=1}^{n} D_{ji}(k,t) \frac{\partial x_i(t)}{\partial x_k(0)}$$
   where $D_{ji}(k,t) = \delta_{ji} + h \frac{\partial \mu_j(t)}{\partial x_i(t)} + \sqrt{h} \sum_{l=1}^{m} \left\{ \mathrm{Z}_l(t+h) \frac{\partial \sigma_{jl}(t)}{\partial x_i(t)} \right\}$
   f. Re-write #2: $\left[\frac{\partial x(t+h)}{\partial x_k(0)} \right] = [D(k,t)] \left[\frac{\partial x(t)}{\partial x_k(0)} \right]$ 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.

g. Re-write #3: $\left[\frac{\partial x(t+h)}{\partial x_k(0)}\right] = [D(k,t)][D(k,t-h)]\ldots[D(k,0)]\left[\frac{\partial x(0)}{\partial x_k(0)}\right]$.

- This is still forward automatic differentiation mode and is $\Theta(n)$, but you can optimize this using specific techniques shown in Glasserman and Zhao (1999).
- Another significant optimization can be achieved by adjointing techniques [Griewank (2000), Giles and Pierce (2000)].
- To achieve further significant optimization, transpose this, to get the following adjoint form:

  $\left[\frac{\partial x(t+h)}{\partial x_k(0)}\right]^{\mathrm{T}} = \left[\frac{\partial x(0)}{\partial x_k(0)}\right]^{\mathrm{T}} [D(k,0)]^{\mathrm{T}}[D(k,h)]^{\mathrm{T}} \ldots.[D(k,t-h)]^{\mathrm{T}}[D(k,t)]^{\mathrm{T}}$, which actually reduces to vector/matrix as opposed to matrix/matrix in the non-transposed version – this would be $\Theta(n^2)$, as opposed to $\Theta(n^3)$.

h. The matrix nature of $[D(k,t)]$ simply arises from the chain rule summation over i. Similar chain rules may be set for the different cash flow Jacobians, etc.

i. Re-casting $D_{ji}(k,t)$ from above as $D_{ji}(k,t) = D_{ji,PRIOR}(k,t) + D_{ji,DRIFT}(k,t) + D_{ji,VOLATILITY}(k,t)$, we can separate 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 $D_{ji,\,DRIFT}(k,t) = h\frac{\partial \mu_j(t)}{\partial x_i(t)}$ is the contribution from the derivative of the drift term. c) The term $D_{ji,VOLATILITY}(k,t) = \sqrt{h}\sum_{l=1}^{m}\left\{Z_l(t+h)\frac{\partial \sigma_{jl}(t)}{\partial x_i(t)}\right\}$ is the contribution from the volatility derivative.

4. <u>Definition of Self-Jacobian Gamma</u>:

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

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

Correlated Stochastic Variables Evolution

1. <u>Continuous Evolution of LMM-type Quantities</u>: Given $\vec{X}$ => the vector of financial variables that need to be mapped to the corresponding Weiner variates $\vec{Z}$. In LMM, for e.g., start with $\vec{X}(0) = \{X_1(0), X_2(0), ..., X_n(0)\}$, then the LMM evolutionary techniques generate $\vec{Z}$ and update $\vec{X}(t)$.
   - Any continuous entity can be chosen to model correlations, not just LMM –type asset movements. If the default process can be correspondingly transformed to an asset indicator variable, that may be correlated with the other asset variables too.
   - For a set of correlated variates, the stochastic evolution equation is:

     $$X_j(t+h) = X_j(t) + h\mu_j\left(\vec{X},t\right) + \sqrt{h}\sigma_j\left(\vec{X},t\right)\sum_{l=1}^{m}\rho_{jl}\left(\vec{X},t\right)Z_l(t+h).$$
   - 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. $Z_l(t+h)$ is in the usual i.i.d. $\aleph(0,1)$.

- The corresponding delta is: $\left[\dfrac{\partial \vec{X}(t+h)}{\partial X_k(0)}\right] = [D]\left[\dfrac{\partial \vec{X}(t)}{\partial X_k(0)}\right]$
- The entry in matrix D is given as:

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

- The corresponding parameter sensitivity is: $\left[\dfrac{\partial \vec{X}(t+h)}{\partial \alpha}\right] = [D]\left[\dfrac{\partial \vec{X}(t)}{\partial \alpha}\right]$
    - This may be simplified in cases where α is an explicit function ONLY of the state evolution variables as:

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

2. <u>Correlated Default Times</u>: 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.
3. <u>Generation of Correlated Default Times</u>:

- 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 $\tilde{\mathrm{Z}}_i$ in $\vec{Z}_{CORRELATED}$:

i. 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.

ii. $\tau_{i,DEFAULT} = S_i^{-1}(y_i)$ where S_i is the survival probability for the entity i.

iii. In general, $X_i = M_i^{-1}(y_i)$.

LMM Forward Rate Evolution

1. Importance of the LMM Formulation: 2 reasons why it is important:
   - LMM is one of the most popularly used formulation, and it is essential to evaluate the impact the no-arbitrage constrained drift has on the evolution and the impact on the greeks.
   - The lognormal nature of the forward rate $\vec{L}(t)$ is important in its own right.
2. No-arbitrage constraint specification:

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

   $\mu_j\left(\vec{L}(t),t\right) = b_j L_j(t) \sum_{p=\eta(t)}^{j} \frac{b_p \Delta(t_{p-1},t_p) L_p(t)}{1+\Delta(t_{p-1},t_p)L_p(t)}$, $\sigma_j\left(\vec{L}(t),t\right) = b_j L_j(t)$, and $\eta(t)$ is the maturity of the first instrument that matures after t [Brace, Gatarek, and Musiela (1997), Jamshidian (1997)].
3. Forward Rate Volatility vs. at-the-swap Swap Option Volatility: 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)].

4. <u>Self-Jacobian of the extended LMM Formulation</u>: As shown in Denson and Joshi (2009a) and Denson and Joshi (2009b),

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

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

5. <u>Forward-Rate Evolution Matrix</u>: As expected,

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

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

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

6. <u>Variate Jacobian Parameter Sensitivity</u>:

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

where $D_{ij}\left(\vec{L}(t),t\right)$ is available from above for the two scenarios.

Rewrite #1: $\dfrac{\partial L_j(t+h)}{\partial \alpha} = B_j\left(\vec{L}(t), t\right) + \sum_{i=1}^{n} D_{ij}\left(\vec{L}(t), t\right)\dfrac{\partial L_i(t)}{\partial \alpha}$, where

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

Formulation of Sensitivities for Pay-off Functions

Formulation of Pay-off Function Stochastic Evolution

1. Monte-Carlo Path-wise Derivatives: Path-wise derivatives are typically forward derivatives, not adjoint [Giles and Glasserman (2006)]. Therefore computation time is proportional to the number of inputs. Further, not easy to accommodate these in complex payouts [Capriotti (2011)].
2. Payoff Expectation Formulation: $V = E_Q\left[P\left(\vec{X}\right)\right]$ [Harrison and Kreps (1979)],

   where $\vec{X}$ is the vector of financial variables.

   - Path Payoff Expectation [Kallenberg (1997)] => $V = \frac{1}{N_{MC}}\sum_{i_{MC}=1}^{N_{MC}} P\left(\vec{X}[i_{MC}]\right)$ and

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

Path Greek

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

   $\left\langle \frac{\partial Y(x)}{\partial x(0)} \right\rangle = \frac{\partial}{\partial x(0)} \langle \partial Y(x) \rangle$ where $x(0)$ is the starting point for the variate.

2. Monte-Carlo Greek Definition: Greek is defined at the change in Y with respect to the starting value of x, i.e., $x(0)$. $\frac{\partial Y(x(t))}{\partial x(0)} = \frac{\partial Y(x(t))}{\partial x(t)}\frac{\partial x(t)}{\partial x(0)}$. If x is a multi-component vector $\vec{X}$, then $\frac{\partial Y\left[\vec{X}(t)\right]}{\partial x_j(0)} = \sum_{i=1}^{n}\frac{\partial Y\left[\vec{X}(t)\right]}{\partial x_j(t)}\frac{\partial x_j(t)}{\partial x_j(0)}$.
3. Pay-off Function Delta: $\frac{\partial V(t)}{\partial x_k(0)} = \sum_{j=1}^{n}\frac{\partial V(t)}{\partial x_j(t)}\frac{\partial x_j(t)}{\partial x_k(0)}$. Now use the earlier formulation for $\left[\frac{\partial x(t)}{\partial x_k(0)}\right]$ to establish the path delta. In particular, using above, $\left[\frac{\partial V(t)}{\partial x_k(0)}\right]^{\mathrm{T}} = \left[\frac{\partial V(0)}{\partial x_k(0)}\right]^{\mathrm{T}}[D(k,0)]^{\mathrm{T}}[D(k,h)]^{\mathrm{T}}....[D(k,t-h)]^{\mathrm{T}}[D(k,t)]^{\mathrm{T}}$, so all the speed up advantages associated with the adjoint formulation above follows.
4. Variance in the Greeks in addition to the base Greeks:
    - Cluster all the Path-wise Greeks calculated for a given input (either $x_k(0)$ or a parameter θ).
    - Within that cluster estimate the corresponding Greek.
    - Usual population sampling variance techniques applied to compute the variance in the Greek.
5. Path Parameter (α) Sensitivity: $\frac{dV(t)}{d\alpha} = \frac{\partial V(t)}{\partial \alpha} + \sum_{j=1}^{n}\frac{\partial V(t)}{\partial x_j(t)}\frac{\partial x_j(t)}{\partial \alpha}$. Now use the earlier formulation for $\left[\frac{\partial x(t)}{\partial \alpha}\right]$ to establish the path parameter sensitivity.
6. Explicit Pay-off Greek Formulation:

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

- Notice that it has additional terms since the explicit dependence of μ,σ on α is, in general, non-zero: otherwise, $B_j(t,\alpha) = 0$, and the pay-off parameter sensitivity formulation proceeds precisely along the same lines as delta formulation.

a. <u>Rewrite #1</u>: $\frac{\partial x_j(t+h)}{\partial\alpha} = B_j(t,\alpha) + \sum_{i=1}^{n} D_{ji}(t,\alpha)\frac{\partial x_i(t)}{\partial\alpha}$ where $D_{ji}(t,\alpha)$ is exactly the same as earlier, and $B_j(t,\alpha) = h\frac{\partial\mu_j(t)}{\partial\alpha} + \sqrt{h}\sum_{l=1}^{m} Z_l(t+h)\frac{\partial\sigma_{jl}(t)}{\partial\alpha}$.

b. <u>Rewrite #2</u>: $\left[\frac{\partial\vec{x}(t+h)}{\partial\alpha}\right] = [B(\alpha,t)] + [D(\alpha,t)]\left[\frac{\partial\vec{x}(t)}{\partial\alpha}\right]$ where $\left[\frac{\partial\vec{x}(t+h)}{\partial\alpha}\right]$, $\left[\frac{\partial\vec{x}(t)}{\partial\alpha}\right]$, and $[B(\alpha,t)]$ are n x 1 column matrices, and $[D(k,t)]$ is an n x n square matrix.

c. <u>Rewrite #3</u>: Generalizing over all the j's, we get

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

d. <u>Rewrite #4</u>: Transposing the above we get

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

.

e. <u>Implications of rewrite #4</u>: Given that $[B(\alpha,t)]^{\mathrm{T}}$ and $\left[\frac{\partial\vec{x}(t)}{\partial\alpha}\right]^{\mathrm{T}}$ are now 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.

f. <u>Adjoint Storage Demands</u>: Remember that $[B(\alpha,t)]$ and $[D(\alpha,t)]$ still need to be retained in memory during the forward evolutionary sweep for $\left[\frac{\partial\vec{x}(t)}{\partial\alpha}\right]$, so this represents a corresponding increase on the storage requirements.

Payoff Sensitivity to the Correlation Matrix

1. Payoff Sensitivity Formulation: Irrespective of where the stochastic process is diffusive or not, $\frac{\partial V}{\partial \rho_{jk}} = \sum_{i=1}^{n} \frac{\partial V}{\partial \tilde{Z}_i} \frac{\partial \tilde{Z}_i}{\partial \rho_{jk}}$, where ρ_{jk} is the correlation matrix.
2. Financial Variable to Correlated Random Partial:
   - Remember the general theorem that if $y(z) = \int_{x=-\infty}^{x=z} \Phi(x)dx$, then $\frac{\partial y}{\partial z} = \Phi(z)$.
   - From this, and using $X_i = M_i^{-1}(y_i)$, you can derive $\frac{\partial X_i}{\partial \tilde{Z}_i} = \Phi\left(\tilde{Z}_i\right)\frac{\partial X_i}{\partial \Phi(X_i)}$.
3. Differential of the Cholesky Factorization Matrix:
   - $\frac{\partial \tilde{Z}_i}{\partial \rho_{jk}} = \sum_{l=1}^{n}\sum_{m=1}^{n} \frac{\partial \tilde{Z}_i}{\partial C_{lm}} \frac{\partial C_{lm}}{\partial \rho_{jk}}$ where $\frac{\partial C_{lm}}{\partial \rho_{jk}}$ is given in Smith (1995).
   - Therefore $\frac{\partial V}{\partial \rho_{jk}} = \sum_{i=1}^{n} \frac{\partial V}{\partial \tilde{Z}_i} \frac{\partial \tilde{Z}_i}{\partial \rho_{jk}} = \sum_{i=1}^{n} \frac{\partial V}{\partial X_i} \frac{\partial X_i}{\partial \tilde{Z}_i} \frac{\partial \tilde{Z}_i}{\partial \rho_{jk}} = \sum_{i=1}^{n} \frac{\partial V}{\partial X_i} \Phi\left(\tilde{Z}_i\right) \frac{\partial X_i}{\partial \Phi(X_i)} \frac{\partial \tilde{Z}_i}{\partial \rho_{jk}}$

     where $\frac{\partial \tilde{Z}_i}{\partial \rho_{jk}}$ is given from above.

Algorithmic Differentiation in Payoff Sensitivities Calculation

1. Monte-Carlo Path-wise Derivatives: Path-wise derivatives are typically forward derivatives, not adjoint (Giles and Glasserman (2006)). Therefore computation time is proportional to the number of inputs.
2. Forward Monte-Carlo evolution variates: The full set forward evolution variates is still needed for extracting the fields/parameters required for the delta estimation of the adjoint path.
3. Corresponding storage requirements: All the variates set (the transition matrices etc.) still need to be maintained, so this represents an increase in the storage needed.
4. Adjointing vs. Reverse Mode: 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 latent state space parameters left to right, therefore technically it is still forward in the time sense – and achieves optimization by minimizing the matrix<->matrix computations.
- In the non-matrix sense (as in adjoint automatic differentiation), the term reverse and adjoint are analogous, 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.

5. <u>Systematic Design Paradigm for using Automatic Differentiation for Path-wise Monte-Carlo Derivatives</u>: Capriotti and Giles (2011) detail several techniques for this.
6. <u>Cost</u>:
   - Forward Automatic Differentiation Cost => $\frac{Cost[B+F]}{Cost[B]} = [2,2.5]$
   - Reverse Automatic Differentiation Cost => $\frac{Cost[B+F+R]}{Cost[B]} = [4,5]$
   - B => Base; F => Forward; R => Reverse.
7. <u>Calibration along with Automatic Sensitivities Generation</u>: Automatic Differentiation is natural performance fit in these situations (Kaebe, Maruhn, and Sachs (2009), Schlenkirch (2011)). Many approaches in this regard end up utilizing intermediate value theorem to facilitate the formulation (Christianson (1998), Giles and Pierce (2000)).

Bermudan Swap Option Sensitivities

Base Formulation

1. Option Valuation under Monte-Carlo: Unlike typical closed forms (such Black-Scholes, Black etc.), volatility does not explicitly show up in the PV generation part for options. Instead, it features intrinsically, through the evolution dynamics, and from the valuation of the underlying that needs to be valued under a specific exercise scenario.
2. H x M Bermudan Swap Option Details:
   - 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 $\vec{X} = \{X_r, X_{r+1}, ..., X_M\}$.
3. H x M Exercised Bermudan Swap Valuation: $X_i = N(T_i)\Delta(t_{i-1}, t_i)[L_i - R]$, where R is the fixed rate. The Bermudan Swap PV is $PV_{Berm}(T_r) = \mathrm{E}\left[\sum_{i=r}^{M} D_f(t_i) X_i\right]$, where $\mathrm{E}[...]$ is the expectation operator.
4. H x M Bermudan Swap Valuation SKU:
   - 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 $\{T_r, PV_{Berm}(T_r)\}$.

Greek Estimation

1. H x M Exercised Bermudan Swap Option Delta/Parameter Sensitivity [Piterbarg (2004), Capriotti and Giles (2011)]:

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

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

2. Individual Cash-flow PV and Greeks [Leclerc, Liang, and Schneider (2009)]:

- $PV_j = D_f(t_j)\Delta(t_{j-1},t_j)[L_j - R]$
- $D_f(t_j) = \prod_{p=1}^{j}\frac{1}{1+\Delta(t_{p-1},t_p)L_p} \Rightarrow PV_j = \left\{\prod_{p=1}^{j}\frac{1}{1+\Delta(t_{p-1},t_p)L_p}\right\}\Delta(t_{j-1},t_j)[L_j - R]$
- Remember that $\frac{\partial PV_j(t)}{\partial L_K(0)} = \sum_{i=1}^{n}\frac{\partial PV_j(t)}{\partial L_i(t)}\frac{\partial L_i(t)}{\partial L_K(0)}$ where $\frac{\partial L_i(t)}{\partial L_K(0)}$ is given by the LMM formulation presented earlier.

3. Cash-flow PV Delta:

- $\frac{\partial PV_j(t)}{\partial L_i(t)} = \frac{\partial}{\partial L_i(t)}\left[\left\{\prod_{p=1}^{j}\frac{1}{1+\Delta(t_{p-1},t_p)L_p}\right\}\Delta(t_{j-1},t_j)[L_j - R]\right]$
- $\frac{\partial PV_j(t)}{\partial L_i(t)}[j \ge i] = \left[\delta_{ij} - \frac{\Delta(t_{i-1},t_i)[L_j - R]}{1+\Delta(t_{i-1},t_i)L_i(t)}\right]\Delta(t_{j-1},t_j)D_f(t_j)$
- $\frac{\partial PV_j(t)}{\partial L_i(t)}[j < i] = 0$

LSM Methodology

1. Curve-Fitting to Extract Optimal Exercise: 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 $PV_{Berm}(T_r)$.
2. Continuous or Fine-grained Call Schedules: LSM is highly effective in these situations. Sampling is reduced to a few evenly spaced-out grid points – such that the full sample scoping is eliminated.
3. Interpolation between Sampled Nodes: 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.

NTD Basket Sensitivities

NTD Product Formulation

1. Running Index Details:
    - $p = 1 \to n$ => Number of Components
    - $j,k = 1 \to n$ => Row, column index of the correlation matrix for each of the n components
    - $l,m = 1 \to n$ => Factorized Cholesky diagonal matrix for the n components
    - r => rth component in the current draw of ordered default times; it corresponds to the current nth-to-default.
    - N => The "N" in NTD ($\tau_N \equiv \tau_r$).
2. Base NTD Pricing:
    - $V_{NTD} = V_{Loss} + V_{\Pr emium} + V_{Accrued}$
    - $V_{Loss} = [1 - R_r(\tau_r)]D_f(\tau_r)N(\tau_r)$
    - $V_{\Pr emium} = c\sum_{i=1}^{n} N(t_i)D_f(t_i)\Delta(t_{i-1}, t_i) \perp (t_i \le \tau_r)$
    - $V_{Accrual} = c\sum_{i=1}^{n} N(\tau_r)D_f(\tau_r)\Delta(t_{i-1}, \tau_r) \perp (t_{i-1} \le \tau_r) \perp (t_i \ge \tau_r)$
    - $\perp (t \le \tau)$ => Default Indicator that is 1 if $t \le \tau$, and 0 otherwise.
        1. 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., $\perp (t \le \tau) \cong \mathrm{H}(t \le \tau)$.
        2. $\mathrm{H}(t \le \tau)$ can be the Heaviside function.
        3. The proxy $\mathrm{H}(t \le \tau)$ has a bias, but it can be designed to be much tighter than the Monte-Carlo accuracy.
3. NTD Sensitivity:

- $$\frac{\partial V_{NTD}}{\partial \rho_{jk}} = \sum_{p=1}^{n} \frac{\partial V_{NTD}}{\partial \tau_p} \frac{\partial \tau_p}{\partial \rho_{jk}}$$
- $$\frac{\partial V_{NTD}}{\partial \tau_p} = \frac{\partial V_{Loss}}{\partial \tau_p} + \frac{\partial V_{\Pr emium}}{\partial \tau_p} + \frac{\partial V_{Accrued}}{\partial \tau_p}$$
- $$\frac{\partial V_{Loss}}{\partial \tau_p} = \delta_{rp} \frac{\partial \{[1 - R_r(\tau_r)] D_f(\tau_r) N(\tau_r)\}}{\partial \tau_p}$$
- $$\frac{\partial V_{\Pr emium}}{\partial \tau_p} = c\delta_{rp} \sum_{i=1}^{n} N(t_i) D_f(t_i) \Delta(t_{i-1}, t_i) \frac{\partial \mathrm{H}(t_i \le \tau_r)}{\partial \tau_p}$$
- $$V_{Accrual} = c\delta_{rp} \sum_{i=1}^{n} \frac{\partial \{N(\tau_r) D_f(\tau_r) \Delta(t_{i-1}, \tau_r) \mathrm{H}(t_{i-1} \le \tau_r) \mathrm{H}(t_i \ge \tau_r)\}}{\partial \tau_p}$$

Basket Options

1. <u>Base Pricing Formulation</u>: $V = D_f(T)\sum_{i=1}^{n}\left[W_i X_i(T) - S\right]^+$

2. <u>Basket Options Delta</u>:

   - Remember 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:

     i. $t = T$

     ii. $V(t) = V_{BO}(T)$

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

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

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.
- 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

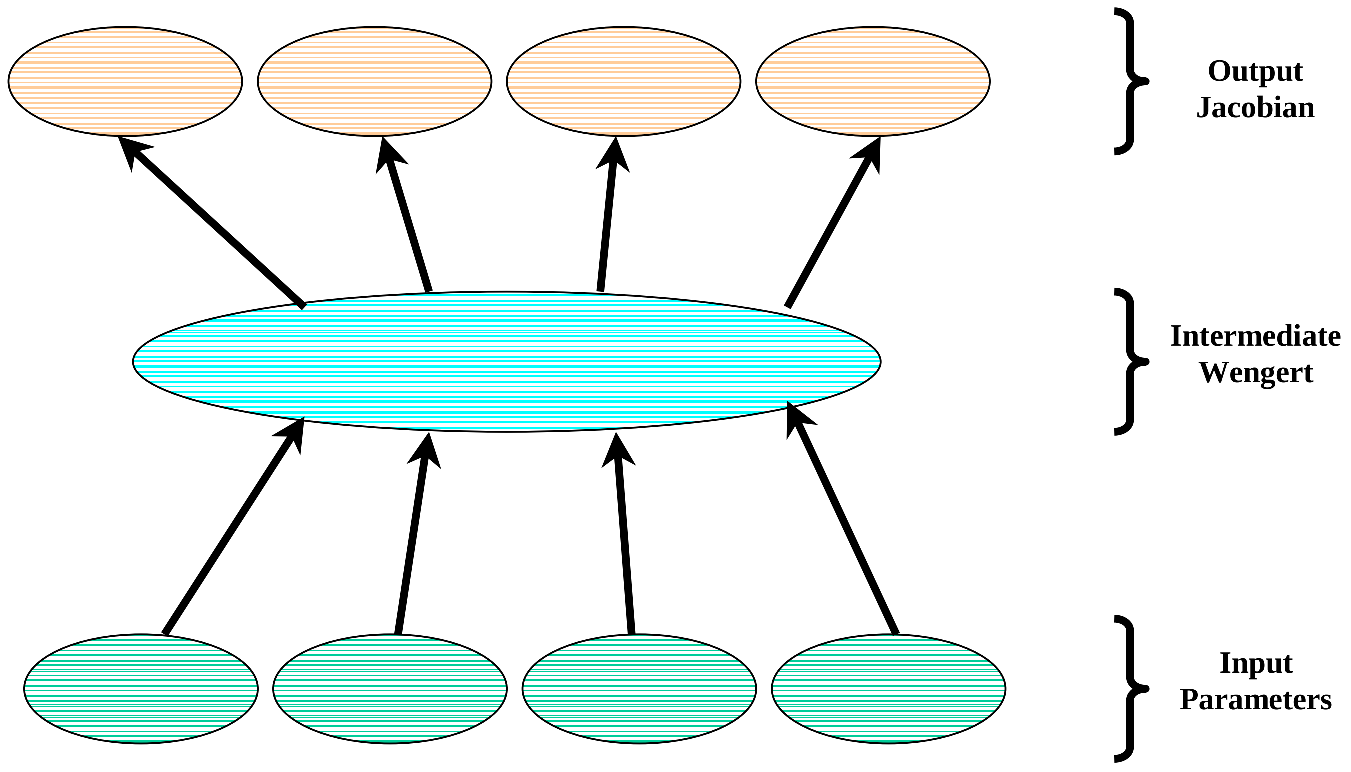


Figure 2: Computation Financial Object Scheme

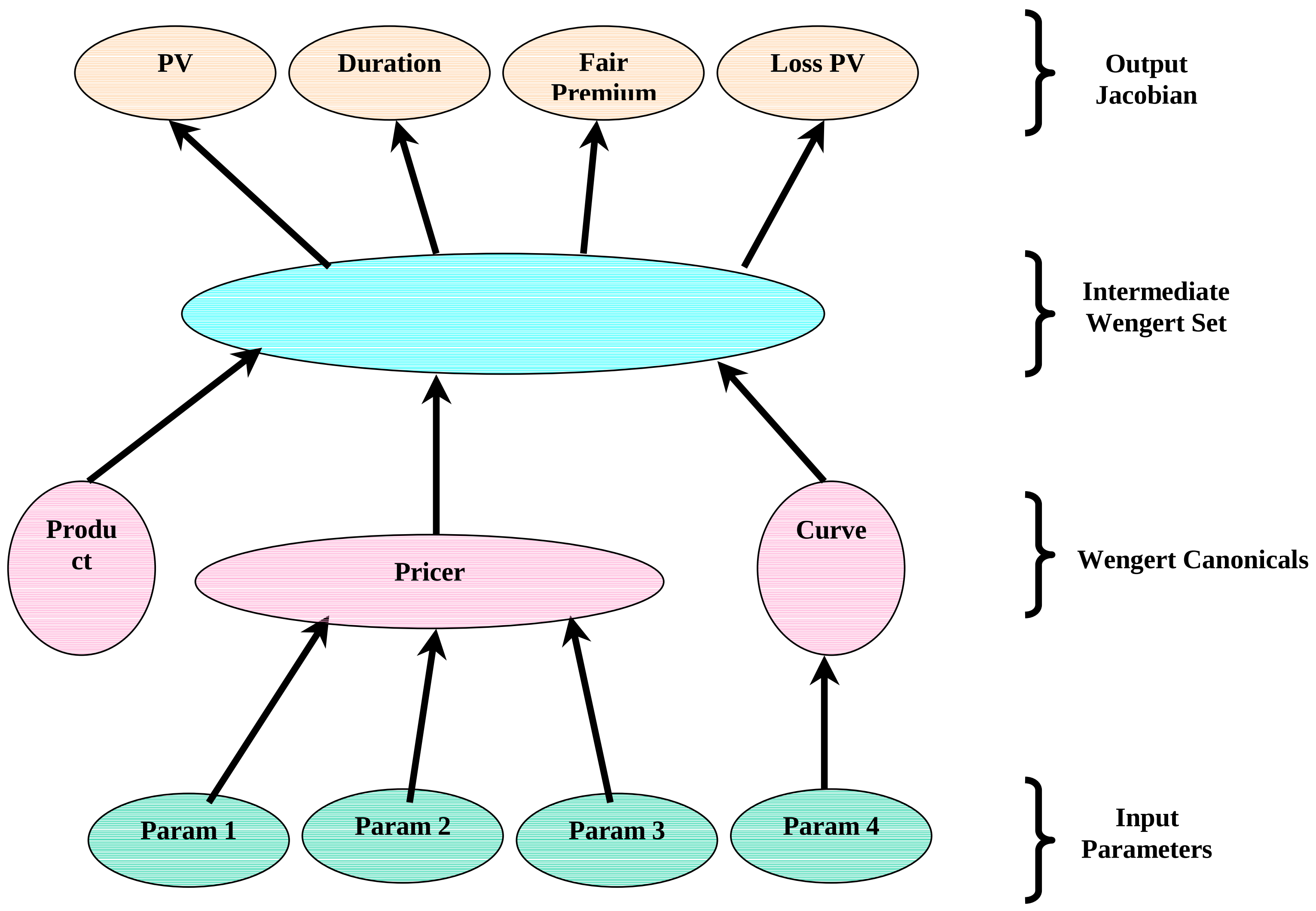
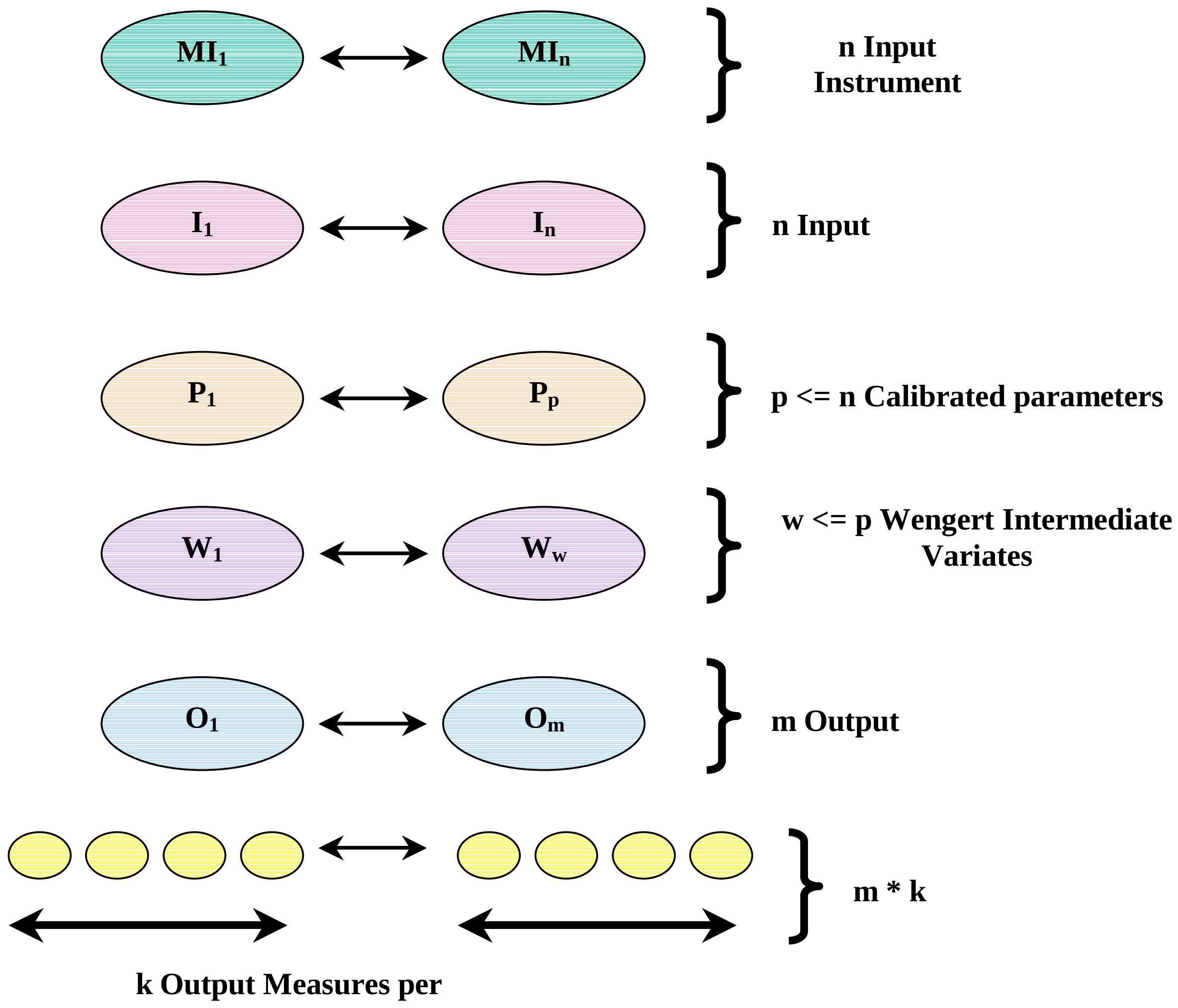


Figure 3: Wengert Fan-in and fan-out



Stochastic Analyzer Software Components

While bulk of stochastic analyzer functionality is formulation based, the algorithmic differential functionality is available across 2 core functional packages.

- Univariate function package
- Univariate Calculus package

Univariate Function Package (org.drip.quant.function1D)

The univariate function package implements the individual univariate functions, their convolutions, and reflections. It contains the following classes/interfaces:

1. AbstractUnivariate: This abstract class provides the evaluation of the given basis/objective function and its derivatives for a specified variate. Default implementations of the derivatives are for black box, non-analytical functions.
2. UnivariateConvolution: This class provides the evaluation of the point value and the derivatives of the convolution of 2 univariate functions for the specified variate.
3. UnivariateReflection: For a given variate x, this class provides the evaluation and derivatives of the reflection at $1-x$.
4. Polynomial: This class provides the evaluation of the n^{th} order polynomial and its derivatives for a specified variate. The degree n specifies the order of the polynomial.
5. BernsteinPolynomial: This class provides the evaluation of Bernstein polynomial and its derivatives for a specified variate. The degree exponent specifies the order of the Bernstein polynomial.
6. NaturalLogSeriesElement: This class provides the evaluation of a single term in the expansion series for the natural log. The exponent parameter specifies which term in the series is being considered.
7. ExponentialTension: This class provides the evaluation of exponential tension basis function and its derivatives for a specified variate. It can be customized by the choice of exponent, the base, and the tension parameter.

8. HyperbolicTension: This class provides the evaluation of hyperbolic tension basis function and its derivatives for a specified variate. It can be customized by the choice of the hyperbolic function and the tension parameter.
9. LinearRationalShapeControl: This class implements the deterministic rational shape control functionality on top of the estimate of the basis splines inside - $[0,...,1)$ - Globally $[x_0,...,x_1)$: $y = \frac{1}{1+\lambda x}$ where is the normalized ordinate mapped as $x = \frac{x - x_{i-1}}{x_i - x_{i-1}}$.
10. QuadraticRationalShapeControl: This class implements the deterministic rational shape control functionality on top of the estimate of the basis splines inside - $[0,...,1)$ - Globally $[x_0,...,x_1)$: $y = \frac{1}{1+\lambda x(1-x)}$ where is the normalized ordinate mapped as $x = \frac{x - x_{i-1}}{x_i - x_{i-1}}$.
11. LinearRationalTensionExponential: This class provides the evaluation of the Convolution of the Linear Rational and the Tension Exponential Function and its derivatives for a specified variate.

Univariate Calculus Package (org.drip.quant.calculus)

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

1. DerivativeControl: DerivativeControl provides bumps needed for numerically approximating derivatives. Bumps can be absolute or relative, and they default to a floor.
2. Differential: Differential holds the incremental differentials for the variate and the objective functions.

3. WengertJacobian: WengertJacobian contains the Jacobian of the given set of Wengert variables to the set of parameters. It exposes the following functionality:
    - Set/Retrieve the Wengert variables
    - Accumulate the Partials
    - Scale the partial entries
    - Merge the Jacobian with another
    - Retrieve the WengertJacobian elements
    - Display the contents of the WengertJacobian
4. Integrator: Integrator implements the following routines for integrating the objective functions:
    - Linear Quadrature
    - Mid-Point Scheme
    - Trapezoidal Scheme
    - Simpson/Simpson38 Schemes
    - Boole Scheme