Universal Algorithmic Differentiation™ in the F3 Platform

Technical Paper
Version 1.1

Published
December 2014

Dr. Mark Gibbs
Dr. Russell Goyder

FINCAD®

Contents

Universal Algorithmic Differentiation™ in the F3 Platform

1.0 Abstract

We present Exposure Projection (EP), an approach to the analytic computation of first order exposure to risk factors in financial models which yields dramatic performance improvements over the use of finite differences in typical applications. We demonstrate a mature implementation of EP within FINCAD's F3 Enterprise Valuation and Risk Platform, called Universal Algorithmic Differentiation™ (UAD).

Exposure Projection represents an advance over the state of the art in analytic first order risk computation, in a number of ways. The set of quotes to which a portfolio is exposed is identified automatically. By avoiding operator overloading, we can choose the optimal granularity at which the chain rule encodes the differentiation and avoid the high storage and run-time costs associated with other implementations. We present three performance optimizations suited to calculation trees with specific structural properties and provide a generic approach to handling discontinuities, including those present in sorting algorithms. The analytic computation of first order risk has been popularized in the finance industry in recent years under the umbrella term "Automatic Differentiation" (AD). We conduct a brief survey of techniques available in the AD literature and compare them with EP.

UAD enables analytic exposure calculation within a generic architecture for derivative valuation, in contrast to the bulk of the literature, which presents model or trade-specific examples. This guarantees fast, analytic exposure calculation for all valuations, vanilla to exotic, single trade to portfolio, under all models and valuation methodologies. We demonstrate UAD in a closed-form setting by hedging the market risk of a multi-currency portfolio of derivatives, and in a Monte Carlo simulation by applying UAD to the portfolio's CVA in a 2-currency FX-rates-equity hybrid model.

2.0 Introduction

In times of crisis, the measurement, understanding and management of risk is brought into sharp focus. In the uncertain economic times of the current post-crisis landscape, exotic derivatives and the associated models are regarded with suspicion. The major challenges of quantitative finance are not in the realm of high theory for modeling a particular asset class for exotic valuation, but are in the areas of counterparty exposure calculation and the aggregation of risk across desks and businesses. Questions such as "How do I hedge my portfolio?" and "How does this trade impact the bank's capital requirements?" are in the spotlight.

There are many sources of risk: operational, market, liquidity and so on. Our focus in this article is market risk. Within market risk, the front and middle offices emphasize different aspects and calculations. Traders in the front office are concerned with profit and loss on their books and as such, for a calculation to be relevant to the business, it must impact the bottom line by assisting in decisions whose outcome can be monetized by trading in the markets. The canonical example of such activity is hedging, for which the first-order exposure (that is, sensitivities - like an option's Greeks such as delta and vega) of a trading book to market risk factors is commonly used.

2.0 Introduction

Before a trading book is hedged, it must be valued. Accurate valuations of derivative portfolios must account for counterparty risk. We described the construction of an appropriate and consistent collection of curves that forms the static part of a model, for the valuation of vanilla portfolios in the context of the relevant set of collateral agreements, in Gibbs and Goyder (2012). Exposure to counterparty default is also encoded in a Credit Value Adjustment (CVA) applied to a given trade, to calculate a new price that takes such risk into account. The calculation of both vanilla portfolio's value and CVA can result in outcomes that can be measured in the profit and loss of a trading desk.

This contrasts with middle office roles who are asked to calculate more subjective measures of risk such as Value-at-Risk (VaR) and Potential Future Exposure (PFE). These fall into a category of calculations based on statistics (such as percentiles) of distributions of portfolio value over potential market scenarios. The distributions chosen are not pricing distributions taken from the market, but adjusted in a variety of ways, usually so that the implied probability of future events matches some understanding of or assumptions about reality, which in turn is often based on an extrapolation of history.

This article focuses on the problem of computing of first-order exposure, which is of primary application to hedging in the front office. We introduce a new approach to this problem called Exposure Projection (EP), which displays a number of advantages over existing methods in the literature. In addition, we demonstrate a complete implementation of Exposure Projection called Universal Algorithmic Differentiation™ (UAD), within F3, a modern analytics platform whose architecture represents a distillation of the accumulated wisdom of over two decades of sell-side analytics platform development. While F3 supports the full spectrum of calculations described above, we concentrate here on first-order exposure.

Denote the exposure of a portfolio of value $V(s_1, s_2, \ldots, s_N)$ to one of the N market quotes on which its value depends, s_i, by Δ_i. This is a generalization of the meaning of Δ in the specific context of option pricing, where s_i is the spot price of the underlying. Δ_i is the partial derivative of V with respect to s_i:

$$\Delta_i = \frac{\partial V}{\partial s_i}.$$

Computing Δ_i is traditionally achieved by the method of finite differences, or "bump and grind". For some small (often one basis point) bump size δs_i, Δ_i is commonly approximated based on the forward difference between the portfolio value at each point, as follows. Using the short-hand , $V(s_i) = V(s_1, s_2, \ldots, s_i, \ldots, s_N)$ we have

$$(1) \qquad \Delta_{\delta s_i} = \frac{V(s_i + \delta s_i) - V(s_i)}{\delta s_i}$$

where $\Delta_i = \Delta_{\delta s_i} + O(\delta s_i^2)$. The cost of this approach scales linearly with the number of risk factors of interest. For all but the smallest vanilla portfolios, this means that exposure to every relevant quote is seldom calculated in practice. Rather, approaches such as bumping an entire collection of market quotes ("bumping the yield curve") are followed. Another consequence is that some quotes can be ignored and relevant exposure missed because intuition, not computation, is used when exploring exposure. Conversely, for a comprehensive calculation of portfolio exposure, hardware can be thrown at the problem, with the result that banks have some of the largest implementations of grid and cloud computing infrastructure in private enterprise.

2.0 Introduction

In contrast to this brute-force approach to exposure calculation, it is possible to compute Δ_i exactly, at a computational cost that is essentially constant with respect to the number of risk factors, by applying the chain rule of differential calculus. This represents a significant advance over the bump-and-grind status quo, resulting in many cases in several orders of magnitude of computational speedup.

Methods for implementing the chain rule are being popularized at the moment, under the umbrella of Automatic Differentiation (AD). While new to many, AD itself is decades old and a number of examples of such analytic exposure calculations are available in the academic literature. However, these methods suffer from a variety of drawbacks, including:

- The set of risk factors to which exposure is calculated, $\{s_i\}$, and therefore the size of that set, must be known in advance.
- The set of risk factors that can be handled is rather small.
- Implementations cover special cases. The academic financial engineering literature provides some ideas and techniques, along with some prototype implementations. Within industry, implementations do exist in production systems, but only for some trades in some areas of some institutions.
- Software tools attempt to add analytic exposure computation to existing code, rather than designing it in from the start, resulting in missed opportunities for optimization.
- Potentially troublesome storage requirements for the intermediate variables used in the calculation.

In contrast, Exposure Projection (EP) gives the relevant set of risk factors $\{s_i\}$ as an output for essentially any derivative or portfolio, in any supported valuation approach, whether Monte Carlo simulation, closed-form, or backward-propagation in Fourier space (Cherubini (2010)). EP was designed into F3 from the start, resulting in a mature, stable, comprehensive and efficient platform for analytic risk computations that is unique among analytics vendors and, to the best of our knowledge, unparallelled by any analytics platform on the planet.

This article explains how to construct such a capability. It starts with a description of Exposure Projection itself and the fundamental ideas on which EP is based in ***Sec. 3 (Page 6)***. We then move through the calculation stack in ***Sec. 4 (Page 17)*** and in doing so, cover the analytic calculation of the exposure of derivative payoffs to model parameters. In ***Sec. 5 (Page 21)*** we deal with the problem of propagating analytic exposure through an arbitrary calibration procedure, and then explain how to deal with discontinuous payoffs in ***Sec. 6 (Page 28)***.

Having established the fundamentals of Exposure Projection, in ***Sec. 7 (Page 31)*** we describe some refinements and optimizations of primary interest in specific computational settings, then demonstrate the capability, giving performance measurements, in ***Sec. 9 (Page 47)***. In ***Sec. 8 (Page 43)*** we conduct a survey of Automatic Differentiation and highlight the advances made by Exposure Projection before concluding in ***Sec. 10 (Page 54)***.

3.0 The Conceptual Structure of Exposure

The foundation of analytic exposure is differential calculus. For the portfolio V in ***Sec. 2 (Page 3)*** (whose value is given by $V(s_1, s_2, \ldots, s_N)$, we can write

$$\mathrm{d}V(s_1, s_2, \ldots, s_N) = \sum_{i=1}^{N} \frac{\partial V}{\partial s_i} \mathrm{d}s_i \equiv \sum_{i=1}^{N} \Delta_i \, \mathrm{d}s_i. \tag{2}$$

The content of any exposure calculation is embodied by the pairing of Δ_i with s_i. While $\{\Delta_i\}$ is just a set of numerical values, in order to form an adequate representation of the information associated with $\{s_i\}$, a richer conceptual structure is required. We explore this structure in ***Subsec. 3.1 (Page 6)*** before proceeding to an explicit treatment of exposure calculation in ***Subsec. 3.3 (Page 10)*** and ***Subsec. 3.4 (Page 13)***.

3.1 Market Data

While the final answer for any exposure calculation is the set $\{\Delta_i\}$, in order to construct a useful report, these numbers must be labelled in some manner. We can base such labels on information associated with the $\{s_i\}$ because they represent the factors to which the portfolio V is exposed. Such numbers are almost always grouped together in sets of a common type and that type is essentially the class of instrument for which they are quotes. For example, we might want to know the exposure of our portfolio to the par OIS rates used to build the discount curve, or the cross-currency swap par rates used to imply the discount curve in another currency.

We have, then, part of our labelling scheme - we need something that encodes the type of instrument quoted in the market. The remaining information required to uniquely identify one of the s_i is something that selects one instrument within the instrument type. For many instrument types, such as the swaps mentioned above, there is a clear one-to-one map between i and the maturity of the quoted instrument. Other instrument types however, require more than a single maturity to specify them completely. For example, swaption quotes populate a volatility cube, whose axes are defined by the strike and expiry of the option together with the length of the swap resulting from exercising the option. There is clearly an abstraction emerging, which we term QuoteSpecification, that encodes the idea of the information that maps to i for a given type of instrument. Before defining it explicitly, however, we must be more precise about what we mean by "type of instrument".

Here, we meet a very intuitive concept, for which we can easily give many examples, but whose rigorous definition is somewhat abstract, though highly useful. Examples include LIBOR swaps, cash deposits, OIS, European equity options, swaptions, futures and many more.

3.1 Market Data

It can be defined more rigorously however as follows.

Definition 1. InstrumentType concept

A mapping from

1. a QuoteSpecification,
2. a quote date,
3. a notional amount
4. and a trade direction

to a Product.

As described in Goyder and Gibbs (2012), a Product encodes the legal terms of a trade and forms one of the fundamental abstractions of F3. The utility of the InstrumentType concept resides in the ability to define a single operation which, for any instrument, constructs the corresponding Product. As demonstrated in Gibbs and Goyder (2012), the canonical application of this capability is to ensure that the calibration of a Model (again, see Goyder and Gibbs (2012)) is consistent with subsequent valuations based on the Model.

For example, the InstrumentType for a vanilla US dollar LIBOR swap includes information such as the definition of the rate paid by the floating leg (3-month USD LIBOR) and how to generate the payment schedule for each leg.

Given ***Definition 1***, we can now define the QuoteSpecification concept.

Definition 2. QuoteSpecification concept

The information required to uniquely identify one instrument among instruments of a common InstrumentType.

It may trouble the reader to observe that these two definitions are mutually recursive, but this just reflects that the border between InstrumentType and QuoteSpecification is essentially arbitrary, to be chosen based on practical concerns. The rule of thumb is that an InstrumentType identifies a screen or table of quotes in an individual's ideal market data management system, and a QuoteSpecification identifies a single entry on that screen.

On a practical note concerning the reporting of exposure, it is useful to be able to represent both InstrumentTypes and QuoteSpecifications as strings. We find it very useful to define two levels at which sorting and classification can be performed within such reports by forming a two-component string representation of an InstrumentType, called a marketdata_tag (or `tag` for short). On a practical note, these tags are readily used as keys in maps. A map of tags to collections of instruments, where each instrument is the triplet of an InstrumentType, QuoteSpecification and a quote, forms a set of market data in F3. ***Fig. 1*** illustrates this structure.

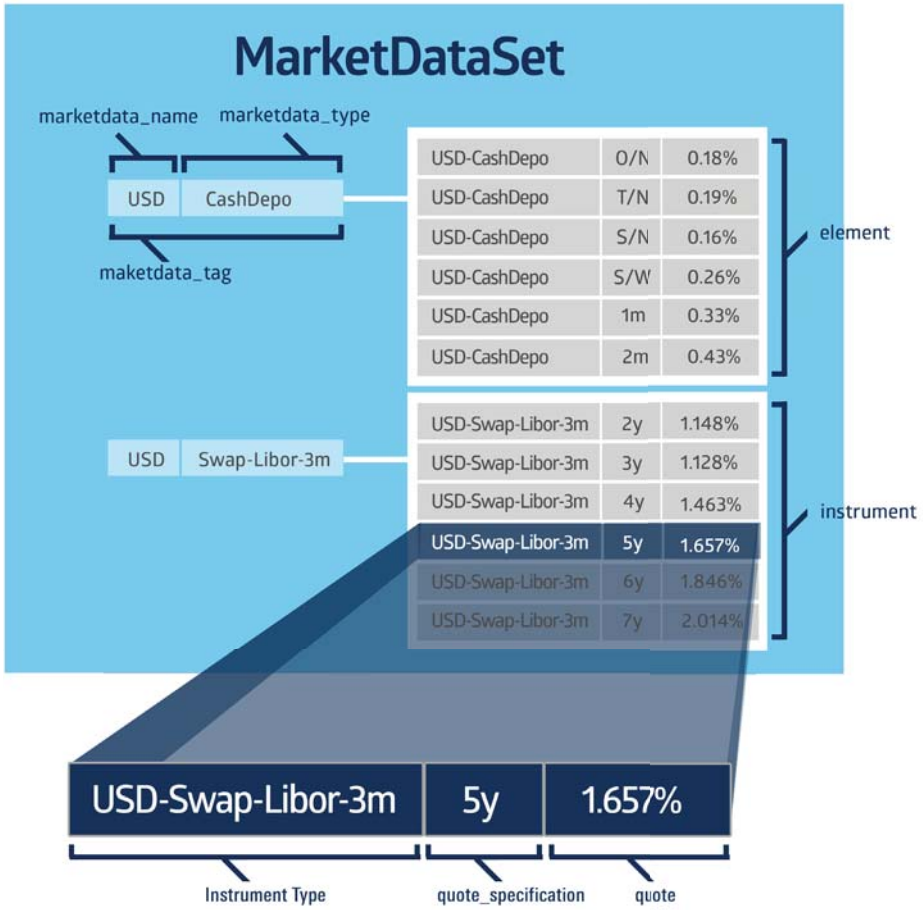

Fig. 1. *The structure of market data. A market data set comprises several elements, each identified by a tag. An element is a collection of instruments and each instrument comprises an instrument type, a quote specification and a quote. The remaining information required to form a real trade is a notional amount, trade date and trade direction (pay or receive, for example)*

3.2 Storing and Reporting Exposure

We have seen in ***Subsec. 3.1 (Page 6)*** how the risk factors to which our portfolio may be exposed are structured. In light of this structure, we can now tackle the problem of how to store and report exposure. We need an association of the numerical value Δ_i with the quote s_i (again just a number) and its corresponding QuoteSpecification and InstrumentType.

Conceptually, the structure we need is defined as follows.

3.2 Storing and Reporting Exposure

Definition 3. ExposureTarget concept

A mapping from

1. an InstrumentType
2. and a QuoteSpecification

to a numerical value.

Although this mapping is the fundamental form that exposure information takes, in practice it can be more convenient to work with market data tags and vectors of values, illustrated schematically as follows.

```
typedef std::pair< std::string, std::string > tag_t;
typedef std::vector< double > exposure_values_t;
typedef std::map< tag_t, exposure_values_t > exposures_t;
```

The above code shows a toy implementation of the ExposureTarget concept. In practice, both *tag_t* and *exposure_values_t* would be classes, in order to implement an appropriate comparison predicate, pretty-printing and other operations such as hash code calculations to support a hash-map, instead of a standard map, to optimize performance. While ***Definition 3*** is phrased in terms of the salient concepts of InstrumentType and QuoteSpecification, the exposures_t type works with a tag as a proxy for the InstrumentType and a numerical index (into the *exposure_values_t* vector) as a proxy for the QuoteSpecification. This works well in practice because it is only when generating a final report for the user that we need the full set of information, which can always be extracted from the Model.

To populate an ExposureTarget, we need another concept, which we term LeafExposure, which accumulates a value into one of the elements of the vector associated with a given tag. If *s_i* is an instance of LeafExposure whose internal state consists of a tag and the value *i*, then we seek behaviour along the following lines.

```
exposures_t target; // empty ExposureTarget
double Delta_i = getTheCorrectExposureValueSomeHow( some, arguments );
// details to follow
s_i.storeExposure( Delta_i, target );
// target[s_i.tag()][i] is now Delta_i;
```

This is not quite what happens in practice in F3's Universal Algorithmic Differentiation™, as we shall see in ***Subsec. 3.4 (Page 13)***, but serves to illustrate the general idea for now. For this reason, we defer discussion of any formal definition of LeafExposure until ***Subsec. 3.4 (Page 13)***, where we find that it is a specific example of a more general concept.

Given an ExposureTarget populated with the exposure of a portfolio, a natural next step is to form a report of its content to the user. Suppose Δ_i takes the value USD 523000 for our portfolio, the tag represents a vanilla LIBOR swap InstrumentType and that `i` corresponds to a QuoteSpecification encoding the idea of "a maturity of five years". A suitable report would then appear as follows.

Market Data Name	Market Data Type	Quote Specification	Quote	Currency	Exposure	Exposure Type
USD	SwapRates	5y	1.081%	USD	523000.00	<RawValueExposure>

The first four columns describe the quote to which exposure has been computed and the remaining columns display the exposure to that quote. The column headings are described as follows:

Market Data Name	First string which forms the tag for this market data element
Market Data Type	Second string which forms the tag for this market data element
Quote Specification	String representation of the information identifying this instrument within the element, such as its maturity
Quote	The quote to which exposure is reported
Currency	The currency of the exposure
Exposure	The value of the exposure, Δ_i
Exposure Type	A string which indicates the type of exposure being reported

In the example above the exposure type is *<RawValueExposure>*, indicating that the given value is the raw partial derivative Δ_i as opposed to some other way of presenting the information, such as a hedging notional (see ***Fig. 11***).

While a report of a portfolio's exposure to market quotes is of prime interest in any production setting, during the development, testing and debugging of valuation functionality it is often preferable to see exposure to model parameters, separate from any subsequent calculation concerning the relationship between calibrated model parameters and market quotes. While the above discussion was presented in terms of market quotes, we can easily calculate and report model parameter exposure, by simply generalizing the concept of "market data" in this context. When calculating exposure to the inputs of a valuation, "market data" is effectively defined as the inputs to which exposure should be reported, whether they have come from the financial markets or not. There is no constraint on where the inputs of a calculation have come from, only that they are labeled such that each occupies a separate location in an *ExposureTarget*.

3.3 The Mathematics of Exposure

Having described the structure of an exposure report and having seen how it can be populated with exposure values, we are now finished apart from the minor detail of the implementation of the function

```
getTheCorrectExposureValueSomeHow( some, arguments );
```

from ***Subsec. 3.2 (Page 9)***.

3.3 The Mathematics of Exposure

We shall see that there is a single unifying operation that encompasses all exposure calculations, including the final accumulation of weight into the ExposureTarget described in ***Subsec. 3.2 (Page 9)***.

In order to illustrate how this can be approached, consider the following example portfolio U consisting of two trades A and B, weighted by their notionals λ_A and λ_B, where A is a cross-currency swap between the numeraire currency of US dollars and the asset currency of Sterling, and B is a Sterling equity forward. If A and B are the (unweighted) US dollar values of the trades A and B, then the portfolio's value can be written as

$$U = \lambda_A A + \lambda_B B. \tag{3}$$

While the techniques we develop in this section are general and apply to any portfolio, for the sake of illustration we can take advantage of our knowledge of this small and relatively simple portfolio to see how the calculation can be broken down. In ***Subsec. 3.2 (Page 9)*** we examined the arbitrary portfolio V, exposed to the N quotes $\{s_i\}, i = 1 \ldots N$, without grouping those quotes into InstrumentTypes (see ***Definition 1***). For the portfolio U, we can assume that the cross-currency swap A is exposed to three InstrumentTypes:

- USD vanilla interest rates swaps. Denote quotes of this type by $\vec{u} = \{u_i\}\ i = 1 \ldots N_u$.
- GBP vanilla interest rate swaps. Denote quotes of this type by $\vec{v} = \{v_i\}\ i = 1 \ldots N_v$ and
- GBP-USD FX spot. Denote this quote by ϕ.

The value of the cross-currency swap is therefore a function of these three variables, $A = A(\phi, \vec{u}, \vec{v})$. Similarly, we know that the Sterling value C of the Sterling equity forward depends on:

- the GBP vanilla rates market $\vec{v}$ and
- equity-related factors such as the spot equity price and assumptions about future dividend payments, which we denote $\vec{w} = \{w_i\}\ i = 1 \ldots N_w$.

This means that B, the (unweighted) US dollar value of this Sterling equity forward, B, can be expressed as

$$B(\phi, \vec{v}, \vec{w}) = \phi C(\vec{v}, \vec{w}) \tag{4}$$

where $C(\vec{v}, \vec{w})$ is the Sterling worth of the equity forward. Listing these dependencies explicitly, ***Eq. (3)*** becomes

$$U(\phi, \vec{u}, \vec{v}, \vec{w}) = \lambda_A A(\phi, \vec{u}, \vec{v}) + \lambda_B \phi C(\vec{v}, \vec{w}).$$

Our eventual goal is to evaluate the exposure of U to the relevant market quotes:

$$\mathrm{d}U(\phi, \vec{u}, \vec{v}, \vec{w}) = \frac{\partial U}{\partial \phi}\mathrm{d}\phi + \frac{\partial U}{\partial \vec{u}}\mathrm{d}\vec{u} + \frac{\partial U}{\partial \vec{v}}\mathrm{d}\vec{v} + \frac{\partial U}{\partial \vec{w}}\mathrm{d}\vec{w} \tag{5}$$

where we have adopted the short-hand notation

$$\frac{\partial f(\vec{x})}{\partial \vec{x}}\mathrm{d}\vec{x} \equiv \sum_{i=1}^{N} \frac{\partial f(\vec{x})}{\partial x_i}\mathrm{d}x_i \tag{6}$$

3.3 The Mathematics of Exposure

to express the sums concisely. Given that $\phi, \vec{u}, \vec{v}$ and $\vec{w}$ are market quotes, once the corresponding weight factors such as $\frac{\partial U}{\partial \phi}$ have been calculated, they can be stored and reported by means of the LeafExposure concept as described in ***Subsec. 3.2 (Page 9)***. However, because we know the functional form ***Eq. (3)*** and ***Eq. (4)*** of the portfolio U, we can begin to calculate these weight factors explicitly. In doing so, we will be able to identify the common pattern that all exposure calculations take and provide a formal definition in ***Subsec. 3.4 (Page 13)***.

The functional relationships we have seen in our portfolio so far are a linear function $f(x, y) = \lambda_A x + \lambda_B y$ and a product function $g(x, y) = xy$. In terms of these functions, our portfolio's value can be expressed as

$$U(\phi, \vec{u}, \vec{v}, \vec{w}) = f(A(\phi, \vec{u}, \vec{v}), B(\phi, \vec{v}, \vec{w})) \tag{7}$$

$$B(\phi, \vec{v}, \vec{w}) = g(\phi, C(\vec{v}, \vec{w})). \tag{8}$$

Applying the chain rule of differential calculus to these functions yields the following expressions

$$\begin{aligned} \mathrm{d}f &= \frac{\partial f}{\partial x}\mathrm{d}x + \frac{\partial f}{\partial y}\mathrm{d}y = \lambda_A \mathrm{d}x + \lambda_B \mathrm{d}y \\ \mathrm{d}g &= \frac{\partial g}{\partial x}\mathrm{d}x + \frac{\partial g}{\partial y}\mathrm{d}y = y\mathrm{d}x + x\mathrm{d}y \end{aligned}$$

which, when applied to ***Eq. (7)*** and ***Eq. (8)*** give

$$\mathrm{d}U = \lambda_A \,\mathrm{d}A + \lambda_B \,\mathrm{d}B \tag{9}$$

$$\mathrm{d}B = \phi \,\mathrm{d}C + C \,\mathrm{d}\phi \tag{10}$$

This application of the chain rule is a recursive process. We could now apply our knowledge of the functional relationships present within A and C to repeat this process over and over again until we arrive at the end-points defined by LeafExposures which then store the result ready for reporting to the user. However, for our current purposes of identifying the common structure that pervades these calculations, we have already gone far enough in this direction. Before discussing this common structure explicitly in ***Subsec. 3.4 (Page 13)***, we consolidate our process so far by expressing the exposure of U in terms of market quotes. The exposure of A and C to the market is given by

$$\begin{aligned} \mathrm{d}A(\phi, \vec{u}, \vec{v}) &= \frac{\partial A}{\partial \phi}\mathrm{d}\phi + \frac{\partial A}{\partial \vec{u}}\mathrm{d}\vec{u} + \frac{\partial A}{\partial \vec{v}}\mathrm{d}\vec{v} \\ \mathrm{d}C(\vec{v}, \vec{w}) &= \frac{\partial C}{\partial \vec{v}}\mathrm{d}\vec{v} + \frac{\partial C}{\partial \vec{w}}\mathrm{d}\vec{w} \end{aligned}$$

which, when substituted into ***Eq. (9)*** and ***Eq. (10)***, gives an updated version of ***Eq. (5)***,

$$\mathrm{d}U = \left(\lambda_A \frac{\partial A}{\partial \phi} + \lambda_B C\right)\mathrm{d}\phi + \lambda_A \frac{\partial A}{\partial \vec{u}}\mathrm{d}\vec{u} + \left(\lambda_A \frac{\partial A}{\partial \vec{v}} + \lambda_B \phi \frac{\partial C}{\partial \vec{v}}\right)\mathrm{d}\vec{v} + \lambda_B \phi \frac{\partial C}{\partial \vec{w}}\mathrm{d}\vec{w}. \tag{11}$$

3.4
Exposure Projection

In the preceding section (***Subsec. 3.3 (Page 10)***) we explored the mathematical structure of exposure using the example of a portfolio U whose exposure was divided into four distinct categories, $U(\phi, \vec{u}, \vec{v}, \vec{w})$. We saw how the rules of elementary differential calculus allow us to calculate the values (Δ_i) that are stored in an ExposureTarget by considering two functional relationships present within U. Although we explicitly described the application of the chain rule for a linear function $f(x, y)$ and a product function $g(x, y)$, we also identified how subsequent functional relationships would be handled.

In doing so, at all times, we were working with equations of the same fundamental form as our original equation ***Eq. (2)***. This is the first key observation that makes it possible to handle exposure generically. It means that there is an operation, which we term exposure projection, that is common to all such calculations. In addition, in all of these formulae, the differential operator $\mathrm{d}x$ appears on both the left and right hand sides of the equations. There is clearly a recursive structure present, which means that the exposure projection operation will be performed as part of another (calling) exposure projection.

As a brief aside, the reason for using the term "projection" is the close analogy between exposure and a basis spanning a vector space. Given an orthonormal vector basis in N dimensions $\{\vec{e_i}\}\ i = 1 \ldots N$, we find the components $\{a_i\}$ of an arbitrary vector $\vec{a}$ by projecting it onto the basis:

$$\vec{a} = \sum_{i=1}^{N} a_i \vec{e_i} \qquad a_i = \vec{a} \cdot \vec{e_i}.$$

Given a function of N variables $a(x_1, x_2, \ldots, x_N)$, we can regard the N inputs as a basis for its exposure:

$$\mathrm{d}a(x_1, x_2, \ldots, x_N) = \sum_{i=1}^{N} a_i \, \mathrm{d}x_i \qquad a_i = \frac{\partial a(x_1, x_2, \ldots, x_N)}{\partial x_i}.$$

The "component" of $\mathrm{d}a$, the full exposure of $a(x_1, x_2, \ldots, x_N)$ in the $\mathrm{d}x_i$ "direction" is the partial derivative with respect to the corresponding variable, x_i.

The second key observation we make is that exposure to a given market factor, such as the spot FX rate ϕ, is composed of a simple sum of terms. This remains true even for nonlinear functions such as $g(x, y)$ due to the first-order nature of the calculation - we focus on small changes in ϕ and neglect higher-order terms, deferring their treatment to the general topic of scenario analysis. It does not make sense to construct this sum of terms explicitly, however, because portfolios are not organized according to the market data required to calibrate the models used for their valuation. Instead, it is best to regard the sum as an accumulation, with contributions from many different parts of a calculation, into the relevant part of some target object, in our case an ExposureTarget. Therefore, the exposure projection operation must take a reference to the target as one of its inputs.

3.4 Exposure Projection

The third key observation we make concerns the nature of the terms that are accumulated into the ExposureTarget. They consist of successive multiplicative factors, again because first-order exposure is fundamentally a linear operation. A new factor appears whenever a functional operation such as $f(x,y)$ is composed with another such as $g(x,y)$, as in ***Eq. (7)*** and ***Eq. (8)***. In other words, each time we encounter the derivative operation d applied to a function $g(\vec{x})$, we obtain the same operation, but applied to its arguments, and weighted by a numerical factor that is composed of the partial derivatives of g with respect to its arguments. In fact, if g itself is the argument of another function f, then $\mathrm{d}g$ will be weighted by $\frac{\partial f}{\partial g}$. We see therefore that a weight factor must be present in the exposure projection operation.

Given the above considerations, the exposure projection operation can be captured by the following interface:

```
struct ExposureProjector
{
  virtual void projectExposure( double weight,
                                exposures_t& target ) const = 0;
};
```

Each time we write an equation in terms of differentials as in ***Eq. (2)***, we can identify each operation with a call of some implementation of this interface. For example, the act of initiating the exposure calculation for U would be encoded as

```
exposures_t target;
U.projectExposure( 1.0, target );
```

Stepping into this function, we would see

```
// lambdaA * dA
A.projectExposure( weight * lambdaA, // weight is 1.0
                   target );
// lambdaB * dB
B.projectExposure( weight * lambdaB,
                   target );
```

which is the implementation of the binary linear sum operation of ***Eq. (9)***. The implementation of the first exposure projection operation (on A) is not shown here because we did not explore the form that A's dependence on its arguments ϕ, $\vec{u}$ and $\vec{v}$ takes. However, ***Eq. (10)*** represents the product function $g(x,y)$, which allows us to show the implementation of

```
B.projectExposure( weight * lambdaB, target );

  virtual void projectExposure( double weight,
                exposures_t& target ) const // member function of B's
  {
    // weight is now 1.0 * lambdaB
    C.projectExposure( weight * phi, // 1.0 * lambdaB * phi,
                       target );
    phi.projectExposure( weight * C, // 1.0 * lambdaB * C,
                         target );

}target );

}
```

As with A, we did not pursue the form that C's dependence on its arguments takes and so we do not show any implementation for it. However, phi is not a function of any other variables - it represents an end-point in this set of `projectExposure` calls and as such, it encodes the LeafExposure concept of ***Subsec. 3.2 (Page 9)***. In fact, the above code reveals that there is no need for a separate LeafExposure operation (called `storeExposure` in ***Subsec. 3.2 (Page 9)***) at all - it is just yet another example of the `projectExposure` interface at work. Its implementation in the case of LeafExposures will make no further `projectExposure` calls. Rather, it will, given knowledge of its tag and numerical index, insert a value into the appropriate location within the target.

We can now define the central concept that underpins EP:

Definition 4. Exposure concept
An object with the ability to perform an exposure projection.

We see that LeafExposure is a specific type of Exposure, one which populates an ExposureTarget.

The code `phi.projectExposure( weight * C, // 1.0 * lambdaB * C, target );` is, in fact, the first point in our example at which the exposure target is populated with any values. The value accumulated is readily identified with the term $\lambda_B C \mathrm{d}\phi$ in ***Eq. (11)***. When an exposure projection is performed on a composite object which in turn depends on other arguments, (a reference to) the target is simply passed down the execution stack. But whenever an end-point in this chain of calculation is reached, the target accumulates a value. In this way, the appropriate values are accumulated in the appropriate locations in the exposure target, in this case given by ***Eq. (11)***.

3.5 Storage Model

We can use the simple example from ***Subsec. 3.3 (Page 10)*** to illustrate the nature of the storage requirements of an exposure projection. The following pseudo-code represents the set of exposure projections examined in ***Subsec. 3.4 (Page 13)***, with the interface `projectExposure( weight, target )`abbreviated to `pE( weight )`. In other words, `projectExposure` has been shortened to `pE` and the ExposureTarget has been dropped from the notation (although it must remain within the scope of each function call), for brevity.

```
U.pE( w ) // overall weight w (= 1.0 above)
|-- A.pE( w * lambdaA ) // add w onto stack
|   |-- ... // add lambdaA (etc, according to internal structure of A) onto stack
|-- B.pE( w * lambdaB ) // remove everything but w
    |-- phi.pE( w * lambdaB * C ) // add lambdaB
    |   |-- target[phi] += w * lambdaB * C // add C, so w, lambdaB and C on stack.
    |-- C.pE( w * lambdaB * phi )
        |-- ... // add phi (etc, according to internal structure of C) onto stack
```

3.5 Storage Model

As each exposure projection function is called, a new frame is added to the stack. Each new frame stores the variables holding the partial derivatives that will be accumulated into the target when the program arrives at a leaf in the calculation tree. After that contribution to the total exposure has been accumulated (the `phi.pE` line), the stack is unwound up to the next call (in this case, `C.pE`). The result is that whenever an exposure is accumulated into the target (such as ϕ), only the variables that form the value to be accumulated (w, λ_B and C) are stored in memory. They are then released before the calculation proceeds. This efficient storage of internal variables is a natural consequence of traversing a tree of exposure projection calls, illustrated in the present context in ***Fig. 2***.

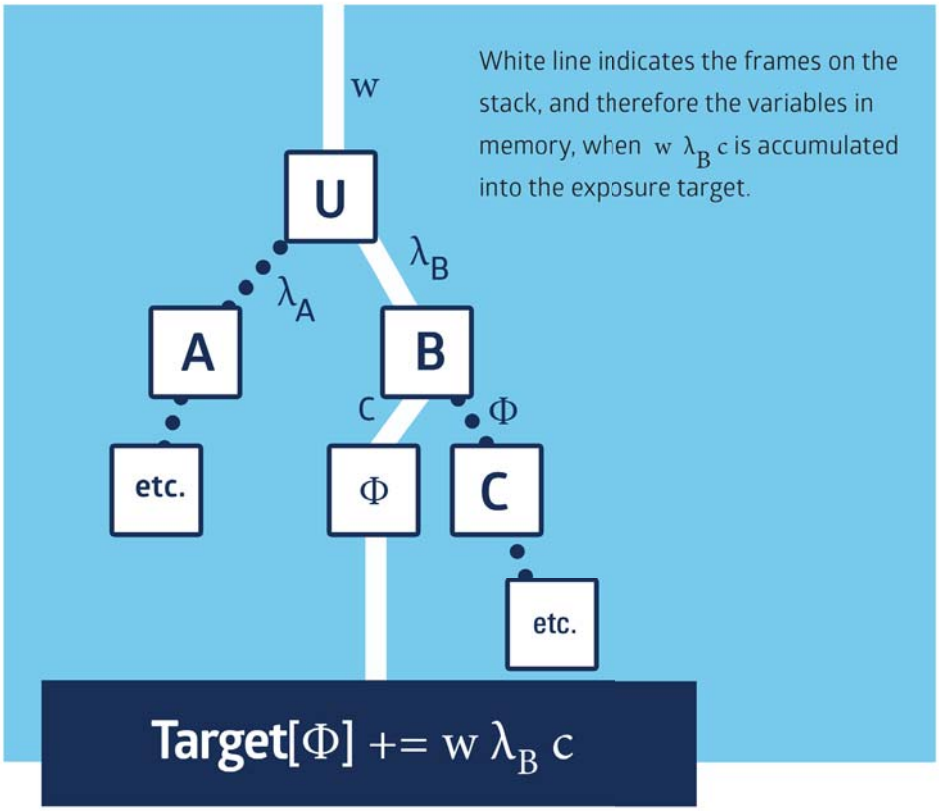

Fig. 2. *An illustration of the calculation tree implicit in an exposure projection for the simple example described in this section.*

We note in passing that, as long as code authors do not take special pains to avoid stack storage, exposure projection is threadsafe up to the shared ExposureTarget. If one target is used per thread then thread-safety is recovered, if the targets are consolidated after each thread has terminated.

4.0
The Valuation Stack

Having explored the structure at the heart of exposure calculation, what remains is essentially the application of the ideas developed so far. This is not to dismiss such an endeavour, however. Indeed, one of the unique differentiators of F3's Universal Algorithmic Differentiation™ (UAD) is the completeness of its implementation of analytic exposure.

A derivative's valuation can be divided into calculations corresponding to two components (see Goyder and Gibbs (2012)). The first is the Product, where the calculations performed are those encoding the rights and obligations specified in the legal terms of the deal (we can regard portfolios, for our purposes here, as trades with a simple nominal term sheet that aggregates the rights and obligations of it's constituents). The second is the Model, where the underlyings referenced by the derivative are modeled. In ***Subsec. 3.3 (Page 10)***, we examined part of the Product, but did not explore it very far. Except for the FX rate ϕ, we certainly did not move into any part of the Model. In this section we conduct a more thorough examination of the objects necessary to support the full analytic calculation of exposure for any trade. We begin at the opposite end of the calculation to Product, move through the Model (deferring a full discussion of calibration until ***Sec. 5 (Page 21)***) and make contact with Products once again in ***Subsec. 4.3 (Page 20)***.

4.1
Parameters

At the core of analytic exposure calculations is the pairing of a numerical value x and its exposure $\mathrm{d}x$. The structure of x is trivial - just a real number - and $\mathrm{d}x$ is an Exposure. The fundamental nature of this association prompts an explicit name for this type of object.

Definition 5. Parameter concept
A number, including its exposure - the pairing of a numerical value with its corresponding Exposure.

Given a collection of Parameters, we can form new parameters by performing functional operations on them. For example, consider a Parameter z formed from the ratio of two other Parameters x and y. Elementary differential calculus gives us the exposure projection for z, as follows.

$$\mathrm{d}z = \frac{1}{y}\,\mathrm{d}x - \frac{x}{y^2}\,\mathrm{d}y.$$

As we saw in ***Subsec. 3.4 (Page 13)***, the implementation of the exposure projection operation for z would consist of two calls to `projectExposure`, one on the Exposure for x with a weight of $1/y$ and the other on the Exposure for y with a weight of $-x/y^2$, where have dropped the overall weight factor supplied to the exposure projection for z.

This structure lends itself to the overloading of operators acting on Parameter instances which yields source code that looks very similar to the mathematics that it encodes and hence would provide a high degree of readability in the resulting program. However, it is important to avoid an overemphasis on calculations implemented in terms of Parameters, because they represent the finest possible choice of granularity for exposure projections. As explained at the end of ***Subsec. 3.4 (Page 13)***, the contribution of each level of function composition to the final value accumulated into the part of an ExposureTarget for a given market quote is stored as a separate entry in the call stack of the program. While in many scenarios working exclusively in terms of Parameters may be adequate, in general it places too tight a constraint on the ways in which exposure calculations may be optimized.

By choosing carefully where exposure projection is implemented, from the full range of objects that form part of an entire valuation, we can select a level of granularity that is appropriate to the given application. This yields optimized code both at runtime and during the development of the code itself. In fact, in F3, in order to encourage an explicit choice of such granularity, we have deliberately avoided the temptation of writing everything out in terms of Parameters by not overloading any of its arithmetic operators and working in terms of a set of factory functions to perform common operations such as the ratio described above.

A commonly used factory function is one that forms a Parameter by binding a function to a specific evaluation point. We consider such functions explicitly in ***Subsec. 4.2 (Page 18)***.

4.2 Function Exposure

Functional relationships are crucial to valuation. Common examples are:

1. Real-valued functions of a single real variable (that is one-dimensional functions), for example today's discount curve $D(t)$.
2. Real-valued functions of two real variables, for example, a volatility surface $\sigma(k,t)$ where k is the strike of an option and t the expiry.
3. Real-valued functions of three real variables, for example an interest rate futures convexity adjustment $A(L,t,T)$ where L is the expected value of the forward rate, t is the valuation time and T is the futures expiry time.
4. Complex-valued functions of a single complex variable, for example the characteristic function of a price process.
5. Functions of no arguments. We have met these already in the form of Parameters.

The key aspect of such functional relationships is that, while dependence on their arguments is explicit, in general they also have an implicit dependence on Parameters. Take, for example, the idealized discount curve

$$D(t) \equiv D(t;r) = \exp(-rt). \tag{12}$$

4.2 Function Exposure

We are making an explicit choice to regard $D(t)$ as a one-dimensional function, when we could instead work with the surface $D(t, r)$. We do this because the time-dependence is always there in a discount curve, whereas the dependence on r is only there if we choose the simple form of ***Eq. (12)***. We could have modeled the discount curve differently, such as via log-linear interpolation (equivalent to piecewise exponential decay, or constant forward rates):

$$D(t) \equiv D(t; s_0, s_1, \ldots, s_{n-1}, d_0, d_1, \ldots, d_{n-1},) = d_i \left(\frac{d_{i+1}}{d_i} \right)^{\left(\frac{t-s_i}{s_{i+1}-s_i} \right)} . \tag{13}$$

where i identifies the interval for which $s_i < t \leq s_{i+1}$. In objects representing these functions therefore, parameters such as r form arguments to the constructor, while t is supplied as an argument to its methods or member functions.

This distinction between the arguments of a function and a function's Parameters must manifest itself in the calculation of exposure. We prefer a very clear manifestation whereby exposure to arguments is handled in a separate part of the interface of a function object and exposure projection is augmented with the value of the argument at which exposure is to be projected. For example, here is some pseudo-code showing part of the exposure projection for the log-linear function of ***Eq. (13)***:

```
void projectExposure( double weight,
                      double t, // need this argument for functions
                      exposures_t& target ) const
                        // member function of a log-linear function class
{
  // extract the values for the interval for which s_i < t < s_{i+1}
  double s_i, s_i_plus_1, d_i, d_i_plus_1 = getValuesForThisInterval( t );
  // pre-calculate the exponent
  double p = ( t - s_i ) / ( s_i_plus_i - s_i );

  // project exposure to d_i_minus_1 first. Extract the corresponding Parameter
  Parameter d_i_plus_1_param = getTheParameter( t );
  // then we can project
  d_i_plus_1_param.projectExposure( weight * p * std::pow( d_i / d_i_plus_1, p),
                                    target );
  // the rest follow similarly...
}
```

The pseudo-code above is for the case of a one-dimensional real function. Given the wide variety of forms that functional relationships can take, it is useful to parametrize the exposure projection interface so it can work for any form of argument and return value. The templates mechanism in C++ provides a suitable mechanism for such a parametrization, as do Java's generics and similar facilities in other languages.

Parameters and functions in their various shapes form some of the basic building blocks of valuation. Their canonical application is in the encoding of the model parameters used to calculate derivatives' value and exposure, found (usually) by means of a calibration procedure. The management of relationships between Parameters (see Gibbs and Goyder (2012)) and their calibration by a Model for efficiency and consistency is described in Goyder and Gibbs (2012) and Gibbs and Goyder (2012) so we will not cover it here.

In F3, the object that performs the act of valuation, and therefore the main user of functions, is called an Engine. Its role will be described in detail in ***Subsec. 4.3 (Page 20)***.

4.3
Engine Exposure

The act of valuation in F3 is performed by calling the function ValueProduct, whose arguments are

1. a Model providing model parameters,
2. a Product encoding the term sheet,
3. a ValSpec specifying the valuation approach
4. and a collection of valuation requests specifying the desired output.

The above information is sufficient to identify the calculation required to value the Product, in the context of the given Model and under the given ValSpec. This calculation is contained within and managed by an object called an Engine, which can be defined as follows:

Definition 6. Engine concept

A provider of, as a minimum, the value and first-order exposure of a Product, in the context of a given Model and under a given ValSpec.

Note that there is nothing in the above definition to constrain a valuation to a single currency. To accommodate multi-currency trades, the value and exposure outputs from an engine take the form of a map from currency to value and a map from currency to an ExposureTarget. That said, if the ValSpec specifies a currency in which value and exposure may be reported, then such single currency reports may be requested by means of appropriate requests in the fourth argument of `ValueProduct`.

The Model, when combined with the ValSpec, produces an object which is capable of emitting the appropriate valuation Engine for any given Product. As such, we term it EngineSource and define it as follows.

Definition 7. EngineSource concept

A mapping from a Product to the appropriate valuation Engine.

In practice, Engines divide into two forms, for the valuation of Products and Indices (an Index represents a financial observable and is described in detail in Goyder and Gibbs (2012)). Consider the simple example of a single cash flow of LIBOR paid at time t with notional N and accrual fraction α whose present value in closed-form is given by

$$V = N\alpha L(s)D(t) \tag{14}$$

where $D(t)$ is given by ***Eq. (13)*** and the curve $L(t)$ (evaluated at time S, as appropriate for a payment at t) gives the expected value of LIBOR as

$$L(t) = \frac{1}{\tau}\left(\frac{D(a)}{D(b)} - 1\right) + S(t) \tag{15}$$

for some spread curve $S(t)$, where a and b denote the start and end of the corresponding borrowing period. In terms of the valuation stack described in this section, this LIBOR payment is organized as follows:

- A ProductEngine holds
  - N and α as floating point numbers,
  - the 1-d function $D(t)$ and
  - a closed-form IndexEngine to calculate LIBOR, which in turn holds
    - the 1-d function $L(t)$ and (after having checked the observation time to determine whether a fixing might be required) simply evaluates it. The $L(t)$ function holds two other functions:
      - $D(t)$, formed by interpolating a set of Parameters $\{\vec{s}, \vec{d}\}$ and
      - $S(t)$, also formed via interpolation of Parameters.

While the example of a single flow of LIBOR is very simple, this hierarchy of Parameters nested inside functions inside IndexEngines and ProductEngines applies to every Product, regardless of its complexity. Engines, functions and Parameters all implement the Exposure interface, which means exposure can always be projected down this valuation stack onto the Parameters.

Now, in the Libor cash flow example of ***Eq. (14)***, the Parameters are LeafExposures, which means that we will report exposure to those Parameters, that is, they form leaves in the calculation tree. Reporting exposure to model parameters is common during the development, testing and debugging of valuation functionality, but in a production deployment such model parameters are typically found by means of a calibration procedure, and so exposure would be projected through them onto market quotes. It is to this task - evaluating the exposure of calibrated model parameters to market quotes - that we turn next.

5.0 Calibration

In ***Sec. 4 (Page 17)*** we examined the components that comprise an analytic exposure calculation for a generic derivative valuation and saw that such calculations are initiated at the level of an Engine which encodes the trade's payoff and which is implemented in terms of functions and Parameters (which can be regarded as zero-dimensional functions).

Therefore exposure is projected through Engines into functions and, eventually, into Parameters such as the interpolation points of the discount curve $D(t)$.

In this section, we tackle the problem of projecting exposure through model parameters onto market data. Model parameters are implied from market data by means of some form of calibration, in which a model's parameters are adjusted according to some algorithm in order to provide a "good enough" match with market prices for some quoted instruments, when the instruments' price is calculated with the model. In fact, as described in Gibbs and Goyder (2012), this is an oversimplification. In general, calibration is the act of comparing two different approaches to valuing the same collection of instruments, and it happens that one approach, called the SourceValSpec, is often either trivial (for example, value to par) or matches an established recipe (for example, options in the Black model). The other approach (TargetValSpec) is based on the model to be calibrated.

Just as calibration is an inverse problem, so is the projection of exposure through a calibration. While the details are specific to the calibration algorithm and metric used to compare with market prices, the general approach benefits greatly from the linear nature of first-order exposure calculations; while inverting a pricing calculation is in general intractable and requires a numerical approach, the linear nature of the corresponding delta calculation ensures that we can perform an analytical calculation.

5.1 Root Searches

One common calibration algorithm is a root-search such as the Newton-Raphson method. All such methods are based on the following metric:

$$X(\vec{p}) = \sum_{i=1}^{n} |X_i| \tag{16}$$

where

$$X_i = S_i(\vec{a}, \vec{b}) - T_i(\vec{b}, \vec{c}, \vec{p}) \tag{17}$$

and where

- S_i denotes the value of the i^{th} instrument under the SourceValSpec
- T_i denotes the value of the i^{th} instrument under the TargetValSpec
- $\vec{a}$ labels the collection of LeafExposures to which only the source valuation is exposed
- $\vec{b}$ labels the collection of LeafExposures to which both valuations are exposed
- $\vec{c}$ labels the collection of LeafExposures to which only the target valuation is exposed
- $\vec{p}$ labels the collection of n parameters being found by calibration

5.1 Root Searches

All n-dimensional root-finding procedures using $X(\vec{p})$ as a metric proceed by finding $\vec{p^\star}$ the value of $\vec{p}$ that results in $X(\vec{p})$ vanishing (to within some suitable tolerance). This is equivalent to the n conditions

$$X_i = 0 \tag{18}$$

for $i = 1 \ldots n$. Our aim is to find the form of the target parameters' exposure $\mathrm{d}\vec{p^\star}$ implied by ***Eq. (18)***. To proceed, we take the total derivative

$$\mathrm{d}X_i(\vec{p^\star}) = 0, \quad i = 1 \ldots n. \tag{19}$$

Expanding the i^{th} such condition using the chain rule, we obtain

$$\frac{\partial X_i}{\partial \vec{a}}\mathrm{d}\vec{a} + \frac{\partial X_i}{\partial \vec{b}}\mathrm{d}\vec{b} + \frac{\partial X_i}{\partial \vec{c}}\mathrm{d}\vec{c} + \frac{\partial X_i}{\partial \vec{p^\star}}\mathrm{d}\vec{p^\star} = 0 \tag{20}$$

where we have used the compact sum notation from ***Eq. (6)***. The set of these equations for $i = 1 \ldots n$ forms a determined linear system which we can solve for $\mathrm{d}\vec{p^\star}$ with standard techniques from linear algebra.

To illustrate, let us consider one of the simplest possible examples, once the canonical bootstrapping problem but now largely of historical interest: constructing a discount curve from a series of swap quotes. To simplify even further, let us ignore any cash deposit or futures quotes and leap straight to the 1 year point d_1 implied by a quote c_1 for a 1-year vanilla interest rate swap. Given the interpolated discount curve of ***Eq. (13)***, the value V_1 of the 1-year swap for unit notional is given by

$$V_1(c_1, s, d_1) = c_1 A_1(d_1) - B_1(S, d_1) \tag{21}$$

where the 1-year annuity A_1 is given by

$$A_1 = \sum_{i=1}^{n_1} \alpha_i D(t_i)$$

for a set of payment dates $\{t_i\}, i = 1 \ldots n$ and associated accrual fractions $\{\alpha_i\}$, and the floating leg B_1 is given by

$$B_1 = \sum_{j=1}^{m_1} \beta_j L(s_j) D(t_j)$$

where the LIBOR rate $L(t)$ is observed at time s_j for payment at t_j. Let this rate be modeled as

$$L(s_j) = \frac{1}{\beta_j}\left(\frac{D(a_j)}{D(b_j)} - 1\right) + S \tag{22}$$

where a_j and b_j mark the start and end of the rate's period and S is a fixed and known (or assumed) spread. This is a special case of ***Eq. (15)***, with a constant spread, which we write as S (dropping the t -dependence from the notation).

5.1 Root Searches

The techniques described in this section do not require such simplifying assumptions - all of the more complex bootstraps described in Gibbs and Goyder (2012) can be treated with the same approach - but for the purpose of illustration assuming a constant spread affords brevity.

Our discount curve will be consistent with this market quote if this swap prices to par. In the language of ***Eq. (17)***, we require that the SourceValSpec sets the value of every instrument to zero, and the target valuation is as described above in ***Eq. (21)***. The parameter sets $\vec{a}, \vec{b}$ are empty, $\vec{c}$ consists of the two numbers c_1 and s and the vector $\vec{p}$ has just one element, d_1. ***Eq. (20)*** becomes

$$\frac{\partial V_1}{\partial c_1}\mathrm{d}c_1 + \frac{\partial V_1}{\partial S}\mathrm{d}S + \frac{\partial V_1}{\partial d_1}\mathrm{d}d_1 = 0$$

whose solution is

$$\mathrm{d}d_1 = \left(-1 \Big/ \frac{\partial V_1}{\partial d_1}\right)\left(\frac{\partial V_1}{\partial c_1}\mathrm{d}c_1 + \frac{\partial V_1}{\partial S}\mathrm{d}S\right). \tag{23}$$

We can therefore construct the Parameter $(d_1, \mathrm{d}d_1)$ and proceed to the second root-find calculation in the bootstrap, based on the 2-year swap quote c_2. By analogy with the above calculation, we can write down the exposure of the second discount curve point d_2 as

$$\mathrm{d}d_2 = \left(-1 \Big/ \frac{\partial V_2}{\partial d_2}\right)\left(\frac{\partial V_2}{\partial c_2}\mathrm{d}c_2 + \frac{\partial V_2}{\partial S}\mathrm{d}S + \frac{\partial V_2}{\partial d_1}\mathrm{d}d_1\right) \tag{24}$$

in which we note the presence of $\mathrm{d}d_1$. Such a recursive structure, with each newly determined curve point depending on points obtained previously, is to be expected in a bootstrap calculation.

In ***Subsec. 3.4 (Page 13)*** we saw that every appearance of a differential such as $\mathrm{d}d_1$ in our equations is encoded by an implementation of the Exposure concept in software. Such implementations may be close representations of the above mathematical forms - for example, $\mathrm{d}d_1$ may calculate two weights and perform two subsequent exposure projections as in ***Eq. (23)*** - or we may choose some alternative implementation if appropriate.

This simple example reveals a circumstance in which an alternative implementation is in fact appropriate. The recursive structure of bootstrap calculations means that whenever the target curve is involved in any subsequent exposure projection, all of the projections like those given by ***Eq. (23)*** and ***Eq. (24)*** are performed as long as the target curve is evaluated at a time later than 2 years. To avoid this inefficiency, we can optimize the calculation by actually performing the exposure projection for $\mathrm{d}d_1$ immediately after the root-find for that curve point has completed. In doing so, we cache the factors multiplying $\mathrm{d}c_1$ and $\mathrm{d}S$ in an ExposureTarget and implement the Exposure interface by storing the target and accumulating any subsequent weights in the relevant parts of the target. In other words, instead of carrying around a calculation tree, we flatten the exposure onto its leaves. This optimization is described in more detail in ***Subsec. 7.1 (Page 34)***.

There is one further subtlety in even this simple example. We have glossed over a distinction among the parameters to which the value $V_1(c_1, S, d_1)$ of the swap is exposed. While S and d_1 are model parameters, c_1 is not. It is part of the term sheet for the quoted instrument, which means that it is not known to the EngineSource constructed from the Model and TargetValSpec - it instead comes from the Product. Engines do not typically project exposure onto quantities in the Product, because they are fixed for a given trade. Any calculation that requires terms like $\mathrm{d}c_1$ must insert them explicitly and each engine which prices an instrument whose quote is intrinsic to the trade in this manner must provide access to the relevant exposure factor -
in this case $\frac{\partial V_1}{\partial c_1} = A_1$

In contrast, other types of instrument are quoted by supplying their price directly. Options are notable in this respect, even though the price is encoded as an implied volatility. We term the quotes for such instruments extrinsic because the quote is not written anywhere on the term sheet - the quote is not specified as part of the trade, but is simply the value of the trade itself. Each type of instrument participating in a calibration that supports a full analytic treatment of first-order exposure must advertise whether it is quoted intrinsically or extrinsically. A good rule of thumb is that instruments for "curve-building", whether in the vanilla rates, cross-currency or credit markets, are quoted intrinsically and are used with a par SourceValSpec and the volatility calibrations applied to dynamic models use a SourceValSpec chosen by convention, such as the Black formula, and are based on instruments quoted extrinsically.

5.2 Gradient Descent and Global Optimizers

While a root-finding approach to calibration is typically found in building curves, for the calibration of models for the dynamics of a market observable a minimization is more common. This is because the systems of equations appearing such calculations are usually over-determined. Rather than pricing each calibration instrument precisely to market (as in ***Eq. (16)***), such calibrations minimize some metric that measures the overall difference between model predictions and reality. The most common metric is that induced by assuming a distribution of errors that maximizes their entropy (Jaynes (2003)):

$$\chi^2(\vec{p}) = \sum_{i=1}^{n} w_i(\vec{u}, \vec{p}) \left(S_i(\vec{v}, \vec{w}) - T_i(\vec{w}, \vec{x}, \vec{p})\right)^2 \tag{25}$$

where the parameter sets $\vec{u}$, $\vec{v}$, $\vec{w}$ and $\vec{x}$ are known and the elements of the set $\{w_i\}$ weight the contribution of each instrument to the sum.

At the point $\vec{p^*}$ in parameter space that minimizes the value of this metric, we have the conditions

$$\frac{\partial \chi^2(\vec{p})}{\partial p_j} = 0 = \sum_{i=1}^{n} \left(\frac{\partial w_i}{\partial p_j} (S_i - T_i)^2 + 2 w_i (S_i - T_i) \frac{\partial T_i}{\partial p_j} \right) \tag{26}$$

for $j = 1 \ldots m$ where m is the number of parameters being calibrated.

5.2 Gradient Descent and Global Optimizers

As before in ***Eq. (19),*** we can take the total derivative of each side of this equation to obtain

$$0 = \sum_{i=1}^{n} \left(\mathrm{d}\left(\frac{\partial w_i}{\partial p_j}\right)(S_i - T_i)^2 + 2\frac{\partial w_i}{\partial p_j}(S_i - T_i)\,(\mathrm{d}S_i - \mathrm{d}T_i) + 2\left[\mathrm{d}w_i(S_i - T_i)\frac{\partial T_i}{\partial p_j} + w_i(\mathrm{d}S_i - \mathrm{d}T_i)\frac{\partial T_i}{\partial p_j} + w_i(S_i - T_i)\,\mathrm{d}\left(\frac{\partial T_i}{\partial p_j}\right)\right]\right). \tag{27}$$

While somewhat voluminous, the terms in this equation, as ever with exposure calculations by their very nature, are linear in the exposures themselves. Assuming that the functional forms involved are sufficiently well-behaved to allow the order of differentials to be interchanged, the factor of $\mathrm{d}\left(\frac{\partial w_i}{\partial p_j}\right)$ in the first term expands into

$$\mathrm{d}\left(\frac{\partial w_i}{\partial p_j}\right) \equiv \frac{\partial(\mathrm{d}w_i)}{\partial p_j} = \sum_{a=1}^{N_u} \frac{\partial^2 w_i}{\partial p_j \partial u_a}\mathrm{d}u_a + \sum_{k=1}^{m} \frac{\partial^2 w_i}{\partial p_j \partial p_k}\mathrm{d}p_k. \tag{28}$$

Given ***Eq. (25)***, we can expand $\mathrm{d}S_i$ and $\mathrm{d}T_i$ as

$$\mathrm{d}S_i = \frac{\partial S_i}{\partial \vec{v}}\mathrm{d}\vec{v} + \frac{\partial S_i}{\partial \vec{w}}\mathrm{d}\vec{w}$$

and

$$\mathrm{d}T_i = \frac{\partial T_i}{\partial \vec{w}}\mathrm{d}\vec{w} + \frac{\partial T_i}{\partial \vec{x}}\mathrm{d}\vec{x} + \frac{\partial T_i}{\partial \vec{p}}\mathrm{d}\vec{p} \tag{29}$$

using the compact sum notation of ***Eq. (6)***. Given ***Eq. (29)***, we can apply the same procedure as for ***Eq. (28)*** in order to express the factor of $\mathrm{d}\left(\frac{\partial T_i}{\partial p_j}\right)$ in the final term of ***Eq. (27)*** in terms of a projection onto $\mathrm{d}\vec{p}$ and the other parameters in the problem.

Thus, by repeating the above for each $p_j,\ j = 1 \ldots m$, we again have, in principle at least, a linear system which can be solved for $\mathrm{d}\vec{p}$. However, it is one that contains both first and second derivatives of the values of the calibration instruments which must be computed in order to project exposure through calibration algorithms that minimize χ^2. Should the ideas covered so far be extended to cover second-order derivatives? A straightforward generalization does exist, by just taking the second term in the Taylor expansion for a portfolio's value. However, the amount of work required to form second order derivatives scales as the square of the number of risk factors, which renders such an act impractical in general.

A more practical approach is suggested by ***Eq. (28)***, where we interchanged the order of the total derivative of w_i and the partial derivative with respect to p_j. Given the ability to calculate $\mathrm{d}w_i(p_1, p_2, \ldots, p_j, \ldots, p_m)$, as afforded by the exposure projection ideas described so far, we can apply m finite-difference calculations, varying p_j by a small amount δp_j in each one, recalculating $\mathrm{d}w_i(p_1, p_2, \ldots, p_j + \delta p_j, \ldots, p_m)$ and evaluating the difference between the two resulting ExposureTargets:

$$\frac{\partial\,(\mathrm{d}w_i)}{\partial p_j} \approx \frac{\mathrm{d}w_i(p_1, p_2, \ldots, p_j + \delta p_j, \ldots, p_m) - \mathrm{d}w_i(p_1, p_2, \ldots, p_j, \ldots, p_m)}{\delta p_j}$$

5.2 Gradient Descent and Global Optimizers

In fact, given this ExposureTarget-based finite-difference approach, we can construct a far neater algorithm by going back to the original condition ***Eq. (26)*** and interchanging the order of derivatives at that stage. Instead of taking the total derivative of the m equations ***Eq. (26)*** for $j = 1 \ldots m$, we can evaluate, by finite difference, the derivative with respect to p_j of the projected exposure of χ^2. In other words, evaluate $\mathrm{d}\chi^2$ twice at two nearby values of p_j, storing the results in an ExposureTarget each time, and calculate the difference between the two. Note that we are applying the finite difference technique to ExposureTargets, thereby forming second-order derivatives. The first-order exposure they contain is still calculated by means of exposure projection.

From this high vantage point of differentiating $\mathrm{d}\chi^2$, it is useful to group all the known parameters under a single symbol $\vec{a}$, where the i^{th} element a_i may represent an element from $\vec{u}$, $\vec{v}$, $\vec{w}$ or $\vec{x}$ depending on the value of i, which ranges from 1 to N_a the sum of the sizes of $\vec{u}$, $\vec{v}$, $\vec{w}$ and $\vec{x}$. Using this notation, we can express the total derivative of χ^2 as

$$\mathrm{d}\chi^2 = \sum_{i=1}^{N_a} \frac{\partial \chi^2}{\partial a_i} \mathrm{d}a_i + \sum_{k=1}^{m} \frac{\partial \chi^2}{\partial p_k} \mathrm{d}p_k.$$

At the minimum χ^2, the partial derivative of $\mathrm{d}\chi^2$ with respect to p_j vanishes for each j, giving

$$\sum_{k=1}^{m} \frac{\partial^2 \chi^2}{\partial p_j \partial p_k} \mathrm{d}p_k = -\sum_{i=1}^{N_a} \frac{\partial^2 \chi^2}{\partial p_j \partial a_i} \mathrm{d}a_i$$

Defining the m by m matrix P_{jk} by

$$P_{jk} = \frac{\partial^2 \chi^2}{\partial p_j \partial p_k}$$

and the m by N_a matrix Q_{ji} by

$$Q_{ji} = \frac{\partial^2 \chi^2}{\partial p_j \partial a_i},$$

we have the following linear system:

$$P_{jk} \mathrm{d}p_k = -Q_{ji} \mathrm{d}a_i$$

where we assume any repeated indices are summed over the relevant range. The solution is

$$\mathrm{d}p_k = -\left[P^{-1}\right]_{kj} Q_{ji} \mathrm{d}a_i \qquad (30)$$

which allows the Parameter $(p_j, \mathrm{d}p_j)$ to be constructed for each j. In passing we note that that the Jacobian of any calibration based on a χ^2 metric is

$$\frac{\partial p_k}{\partial a_i} = -\sum_{j=1}^{m} \sum_{l=1}^{N_a} \left[\frac{\partial^2 \chi^2}{\partial p_j \partial p_k}\right]^{-1} \frac{\partial^2 \chi^2}{\partial p_l \partial a_i}.$$

although in exposure projection it is never constructed explicitly as a matrix. Rather, it is implicit in the exposure projector ***Eq. (30)***.

6.0 Discontinuities

The ideas developed so far in this article provide a concrete solution to the problem of evaluating partial derivatives of complicated functional relationships by decomposing them into simple components and using the chain rule to assemble the pieces. A central assumption that underpins the entire development is that the relevant functionality relationships are differentiable. However, many functional relationships found within financial derivative contracts are not.

Even the canonical derivative - a European option on some stock S struck at k - has a value at expiry of

$$C(S) = \max(S - k, 0) \tag{31}$$

which is not differentiable at the point $S = k$ because its derivative with respect to S suffers a discontinuity, jumping from 0 to 1. We shall see shortly that any payoff that is conditional on some event can be expressed using a step (Heaviside), function, which has no finite derivative at the step. We shall also see that any payoff that involves sorting, such as mountain range options popular particularly in the early 90's, introduces similar non-differentiability. How can we apply differential calculus to functions that are not differentiable? It is this question that we address next.

The answer to this question is no different from the answer to the wider question of how singularities and infinite sets are treated in any practical implementation, in any modeling context from black holes to probability theory. We replace the singular or infinite system with a finite, parametrized one, such that it approaches the true system as a limiting case. Applied to derivative payoffs, this amounts to replacing all non-differentiable functions such as those described above with differentiable ones that tend to the original functions in a controllable limit.

As an example, take the function $f(x) = \max(x, 0)$ implicit in the European option payoff ***Eq. (31)*** and replace it with

$$\tilde{f}(x) = \begin{cases} 0, & \text{if } x < -\epsilon, \\ \frac{1}{4\epsilon}(x+\epsilon)^2, & \text{if } |x| \le \epsilon, \\ x, & \text{if } x > \epsilon. \end{cases} \tag{32}$$

The behaviour of this function is shown in ***Fig. 3***.

6.0 Discontinuities

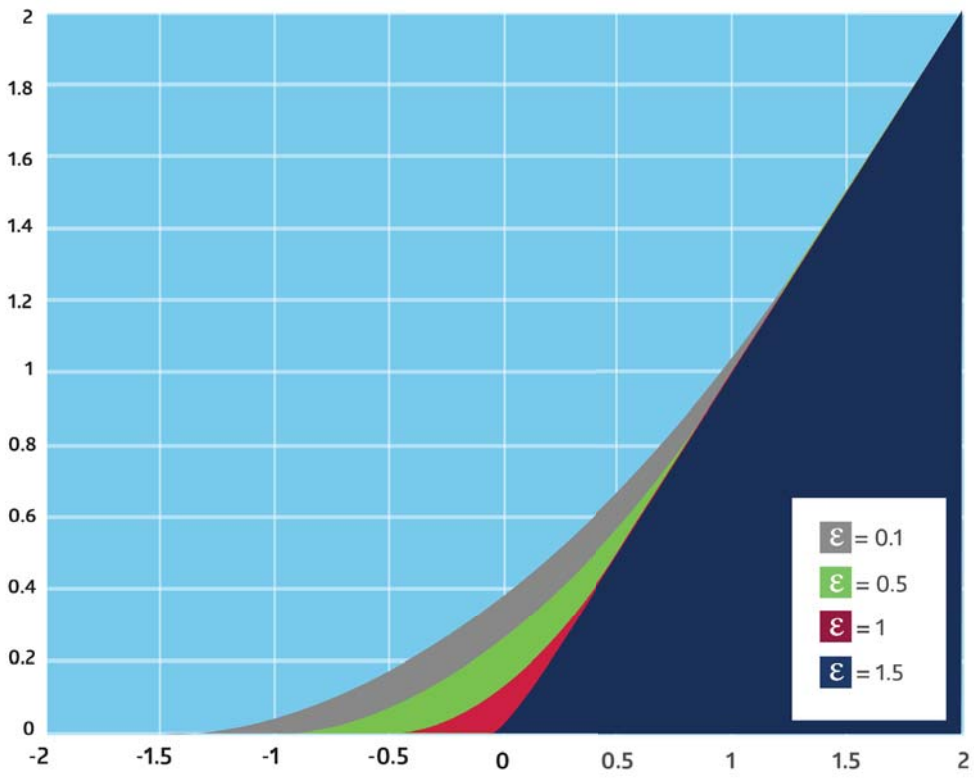

Fig. 3. *Smoothed, therefore differentiable, maximum function that approaches a European option payoff in the limit of vanishing smoothing zone*

In the small region of size 2ϵ around $x = 0$ (or $S = k$), $\tilde{f}(x)$ is a quadratic that matches both the value and all derivatives of $f(x)$ at the region's boundaries $x = \pm\epsilon$. Inside the smoothing region, the values of $f(x)$ and $\tilde{f}(x)$ do not quite match, but $\tilde{f}(x)$ remains differentiable and the discrepancy in value can be made to be arbitrarily small by adjusting the parameter ϵ. In particular, we have that

$$\lim_{\epsilon \to 0} \tilde{f} = f. \tag{33}$$

Given the techniques described in this article, there is nothing to stop us treating the parameter controlling this parametrization as just another variable to which exposure should be calculated. Outside the region $-\epsilon < x < \epsilon$, there is no exposure, but inside, we have

$$\frac{\partial \tilde{f}(x;\epsilon)}{\partial \epsilon} = \frac{x+\epsilon}{2\epsilon}\left(1 - \frac{x+\epsilon}{2\epsilon}\right).$$

6.0 Discontinuities

Having the exposure ϵ to available in our ExposureTarget is of immense value, because we can assess immediately the extent to which the results of our valuation depend on the modeling assumption encoded via ***Eq. (32)***. If the ϵ exposure is negligible, then the valuation is insensitive to the fact that the payoff has a kink at $S = k$ and we can proceed in our application of the techniques described in this article without worrying about the non-differentiability present in the problem.

If, however, we have a non-negligible exposure to ϵ, then the valuation is indeed probing the non-differentiable region and is therefore dependent on the smoothing methodology we have chosen to manage that non-differentiability. There is no automatic recipe to follow in such cases, although there are some good general guidelines. One is to marginalize over several smoothing methodologies or at least compare their results. Another is to perform an extrapolation based on a selection of values of ϵ, to estimate the result of taking the limit ***Eq. (33)***, although this extrapolation introduces further modeling assumptions.

Regardless of the choice of strategy for dealing with significant ϵ dependence, it is always valuable to possess the knowledge that a valuation result is model-dependent, and the degree to which this is so. It is far better to understand that a number should be treated carefully, and know how carefully, than it is to use the number while ignorant of potentially dangerous consequences of doing so.

We have shown one manner in which the discontinuity in gradient present in a European option payoff can be smoothed. The general approach for other non-differentiable functional forms is more of the same - every sharp edge must be smoothed and the true payoff can be approached as a limiting form of the replacement. We now give two further examples of this. The first case arises when a condition is present in the payoff. Consider, for example, a contract assigning rights or obligations contingent on some observable $X(t)$ breaching a barrier during some time interval $a < t \leq b$. The object of interest is the indicator function $I_{ab}(X)$, evaluating to true if the barrier was breached and false if not. If the two outcomes resulting from each Boolean value are the payoffs $F(t)$ and $G(t)$, then we write: "if $I_{ab}(X)$ then $F(t)$ else $G(t)$", which is not differentiable.

To make progress, we first relax the constraint that I_{ab} is Boolean-valued, replacing it with a real-valued function $C_{ab}(X)$ and interpreting values of 1 (or more) and 0 as true and false respectively. We then form the expression

$$(34) \qquad C_{ab}(X)F(t) + (1 - C_{ab})\,G(t)$$

which is equivalent to the original conditional "if" expression when $C_{ab}(X)$ evaluates to 1 or 0. We then apply the same approach as before, defining a smooth transition from 0 to 1 for $C_{ab}(X)$ over a range controlled by a parameter, resulting in a differentiable expression and the same modeling considerations as above. The same approach covers composite logical conditions by identifying Boolean and with multiplication and or with addition.

While not particularly common, it is possible to write a contract based on the location of elements in an ordered list. Mountain range options such as Himalayas that became popular in the early 90's are perhaps the clearest examples of such trades. In order to construct ordered lists, a collection of objects must be sorted. More so than with logical conditions, it is far from clear a priori how sorting algorithms might be rendered into a differentiable form. One solution to this problem follows from the fact that a sort can in fact be expressed as a known number of comparison operations (Batcher (1968)), which in turn can be expressed in the same manner as ***Eq. (34)***. Thus, sorting becomes a differentiable operation.

While the techniques described in this chapter can render a given discontinuous payoff differentiable, it is another matter to construct a platform that guarantees that all sharp edges are smoothed. Such an endeavour requires both an architecture and a development process that ensures a very high degree of modularity and component reuse. As with the `ExposureProjector` interface (***Subsec. 3.4 (Page 13)***), it is impractical to retrofit a pervasive smoothing capability - any successful implementation must be an initial design consideration and achieve the status of a first class architectural feature.

7.0 Optimizations

The concepts of Exposure and ExposureTarget given in ***Sec. 3 (Page 6)***, together with the ideas that describe how to project exposure through the valuation stack and calibrations (***Sec. 4 (Page 17)*** and ***Sec. 5 (Page 21)***) allow us to construct analytic calculations for the exposure of any derivative to all the market risk factors on which it depends. The chains of `projectExposure` calls that implement the chain rule of differential calculus can be visualized as an acyclic, but recombining graph. For example, consider an interest rate swap valued using a discount curve bootstrapped with the toy model of ***Subsec. 5.1 (Page 22)***, but with N quotes $\{c_i\}$ $i = 1 \ldots N$. Write its value as

$$V = mA - B \tag{35}$$

where m is the (annual) fixed coupon and the annuity A is given by

$$A = \sum_{i=1}^{N} \alpha_i D(t_i)$$

for a set of payment dates $\{t_i\}, i = 1 \ldots n$ and associated accrual fractions $\{\alpha_i\}$, and the floating leg B is given by

$$B = \sum_{j=1}^{M} \beta_j L(s_j) D(t_j)$$

where the LIBOR rate $L(t)$ (whose tenor is M/N) is given by Eq. (22). Then, the exposure projection can be visualized as the tree shown in Fig. 4. Each box displays one of the series of functional relationships present in the calculation of the swap's value, starting with Eq. (35) and descending through to the log-linearly interpolated discount curve of Eq. (13). The connecting arrows indicate the calls to projectExposure that encode each application of the d operation. Thicker lines denote a multiplicity of such calls, while thinner lines denote a single call.

7.0
Optimizations

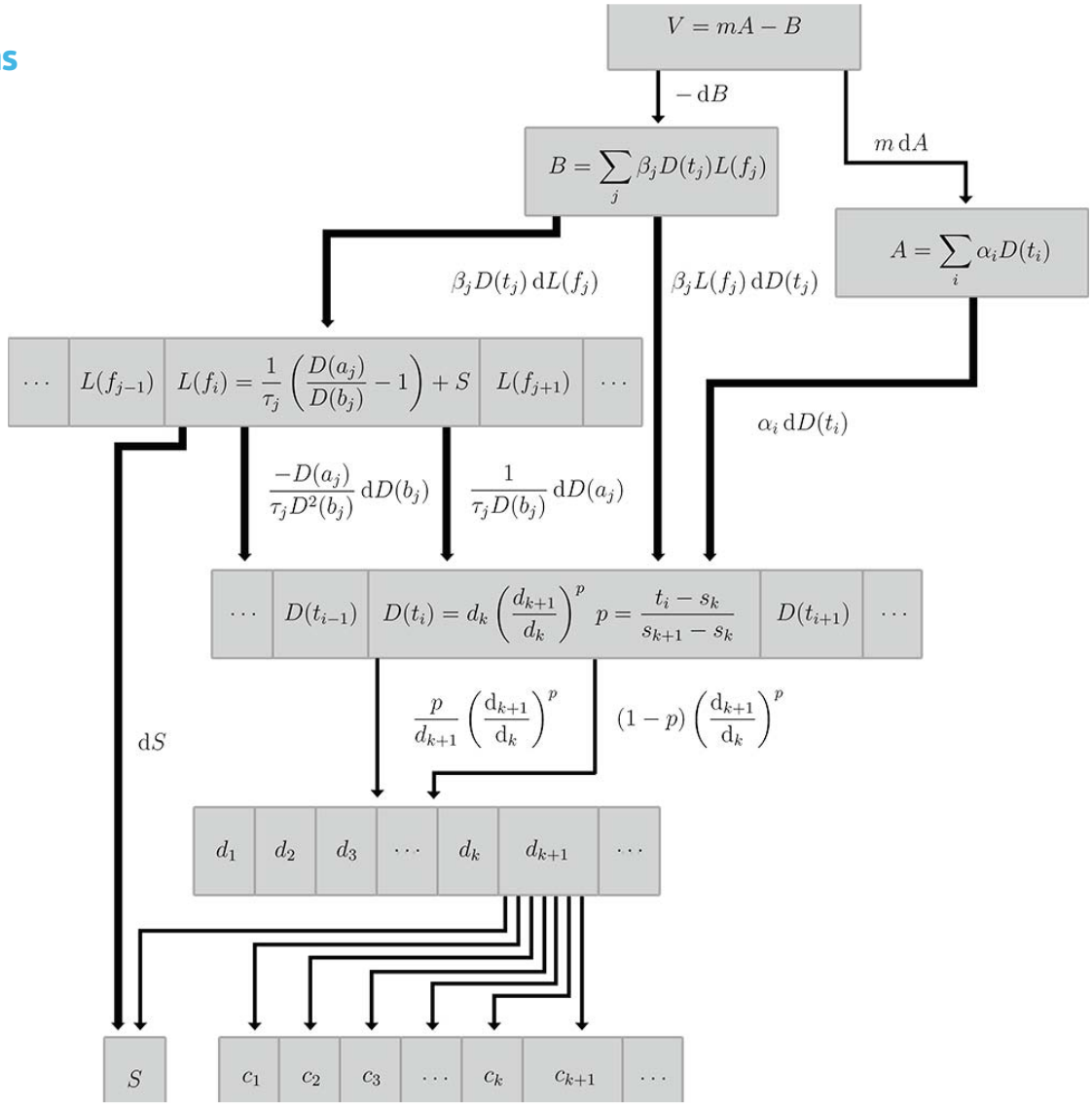

Fig. 4. *Calculation tree for the exposure projection of a vanilla LIBOR swap.*

Even in ***Fig. 4*** we are confronted by considerable complexity. We have ignored exposure to the set of times $\{s_i\}$ and have not drawn the many arrows that capture the exposure of each discount factor to all the quotes that mature before the corresponding cash flow. For typical portfolios, such calculations trees become vastly larger, and the larger they get, the more expensive it becomes to calculate exposure (though the cost of calculating value also increases, with the result that an analytic approach still affords a dramatic speed increase over bump-and-grind for any realistic number of quotes). When the calculation tree exhibits certain types of structure, it is possible to perform optimizations that leverage that structure. In the following sections, we describe a small, indicative set of such optimizations.

7.1 Flattening

The first optimization we consider is one we touched on in ***Subsec. 5.1 (Page 22)***. The leaves of the calculation tree shown in ***Fig. 4*** consist of a set of market quotes, $\{c_k\}$ (and the spread S which we ignore in this section for brevity). The discount curve $D(t)$ was implied by this set of quotes via the same root-finding calculation, with log-linear interpolation through a set of points $\{d_k\}$, that was described in ***Subsec. 5.1 (Page 22)***. The associated exposure calculation is given by ***Eq. (23)*** and ***Eq. (24)*** where in the latter equation, we see that d_2 is exposed to changes in both c_2 (directly) and c_1 (via d_1). This structure can be seen in ***Fig. 5*** for the first five such quotes and discount factor points.

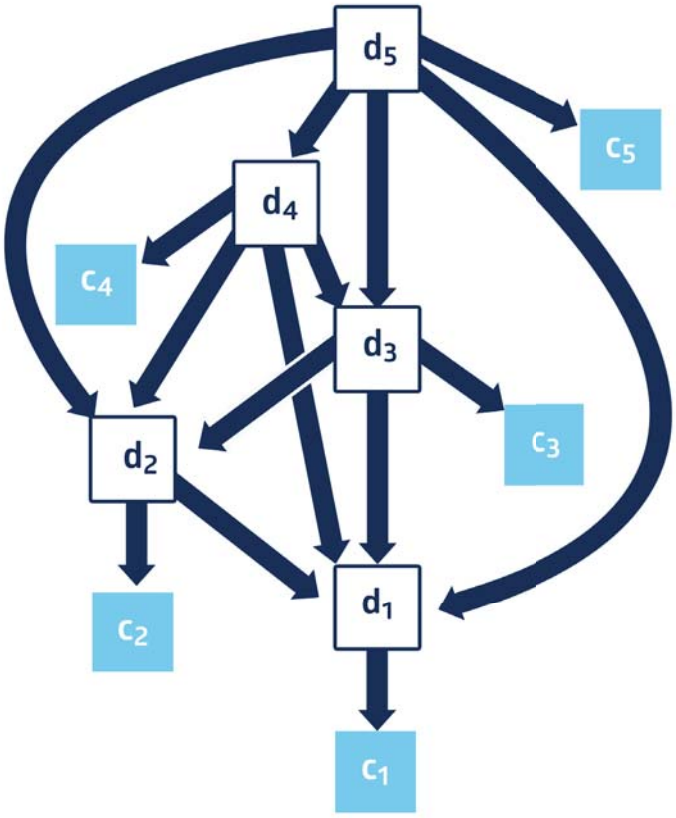

Fig. 5. *Graph showing the recursive structure implicit in the dependence of bootstrapped discount factor curve points on market quotes.*

7.1 Flattening

This is a recursive structure; each discount factor interpolation point d_k projects exposure onto c_k and onto d_i for $i = 1 \ldots k$. The cost of projecting exposure onto d_n is $O(n^2)$, but in many applications (such as valuing a series of cash flows) we require the exposure is projected in a loop from 1 to n, adding a further power of n to the computational complexity.

This scaling can be reduced to $O(1)$ (constant time, or $O(n)$ if in a loop over n) by flattening the calculation tree, so that it resembles ***Fig. 6***.

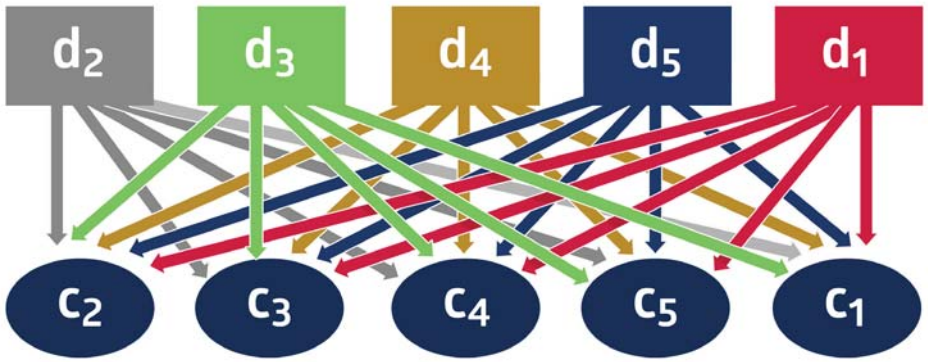

Fig. 6. Graph showing the dependence of bootstrapped discount factor curve points on market quotes after flattening the tree to remove any recursive structure.

This flattening can be achieved by changing the implementation of d_n's exposure projection operation to one which is based on exposure values for the leaves of the tree

$$\sum_k \frac{\partial d_n}{\partial c_k} \mathrm{d}c_k \tag{36}$$

stored in an ExposureTarget, say T_{stored}. When exposure is projected onto a new target T_{new}, the values from T_{stored} are added into the corresponding locations in T_{new}. The partial derivatives in ***Eq. (36)*** can be evaluated by simply asking d_n to project its exposure onto T_{stored}. If this is done as soon as the exposure calculation for d_n (described in ***Subsec. 5.1 (Page 22)***) is complete, then a full traversal of the recursive tree of ***Fig. 5*** is never performed.

7.2
Underlying Projectors

The idea of exposure projection described so far is one that avoids the need for explicit allocation of any storage for the intermediate Jacobians at each node in the calculation tree. Rather, as described at the end of ***Subsec. 3.4 (Page 13)***, the required storage is that for only one path from root to leaf as the calculation tree is traversed, and is on the program stack. The top-level exposure projection operation starts with an empty ExposureTarget, makes a single call to `projectExposure` and results in a full ExposureTarget whose contents can then be queried and reported.

There are, however, times where this one-branch-at-a-time approach is suboptimal, and instead we want to descend only to a given level in the calculation tree, stopping short of the leaves each time. In such scenarios we wish to break the chain that connects root to leaf at a key intermediate point or points, and then project exposure onto those intermediate points, thereby deferring the full projection until some later stage.

An important example of such a scenario is Monte Carlo simulation, where the value of a derivative is approximated by summing over samples from its risk-neutral distribution, generated by passing samples of the underlyings from their joint distribution through the contract's payoff function. First-order exposure may be approximated in the same way. For each iteration (path) in the simulation, exposure can projected onto an ExposureTarget and by doing so, the average value of each exposure will be accumulated. In order to illustrate how a full exposure projection is suboptimal here, consider the toy example of a vanilla European call option valued in the Black model, but in a Monte Carlo simulation (with static interest rates). Let the present value of the option be

$$V_0 = \frac{1}{N}\sum_{i=1}^{N} V_0^i$$

7.2 Underlying Projectors

where V_0^i is the i^{th} of N samples from the risk-neutral price distribution. The payoff for the i^{th} sample can be written as

$$V_0^i = D(T') \max\left(F_T^i(T) - k, 0\right)$$

where T' is the settlement time for expiry at T, $D(t)$ is the discount curve, the option is struck at k and state variable $F_T^i(t)$ is the value of a time T forward contract on the underlying as seen at t. In the Black model, we can write the state variable as

$$F_T^i(T) = F_T(0) \exp\left(\sigma(T)\sqrt{T}x_i - \frac{1}{2}\sigma^2(T)T\right) \tag{37}$$

in terms of the two model parameters $F_T(0)$, today's value of the forward and $\sigma(T)$, its realized volatility at T together with x_i, the i^{th} sample drawn from a standard normal distribution $N(0,1)$. Assume that we model $\sigma(T)$ by interpolating a collection of quoted volatilities $\vec{\sigma}$ (corresponding to the strike k) linearly in variance:

$$\sigma^2(t)\,t = \frac{t - t_i}{t_{i+1} - t_i}\sigma_{i+1}^2\,t_{i+1} + \frac{t_{i+1} - t}{t_{i+1} - t_i}\sigma_i^2\,t_i \tag{38}$$

where σ_i is the i^{th} element of $\vec{\sigma}$, for expiry t_i. Let the static part of our model be constructed as follows. Assume some dividend structure incorporating both a continuous dividend model $R(T;\vec{r})$ parametrized by a set of quantities $\vec{r}$ (such as a single dividend rate, or term structure of such rates) and a discrete dividend specification based on a set of quantities $\vec{q}$ (such as the expected absolute or relative dividend amounts). Then we can express the funding curve for the underlying as

$$X = X(T; R(T;\vec{r}), \vec{q})$$

without going further into the precise functional form. Our model for the forward curve is therefore

$$F_T(0) = \frac{S_0}{X(T; R(T;\vec{r}), \vec{q})} \tag{39}$$

given the underlying's spot price S_0. Lastly, assume that our discount curve $D(t)$ is that described in ***Subsec. 5.1 (Page 22)*** and so depends on the quotes $\vec{c}$.

As shown in ***Eq. (37)***, the value $F_T^i(T)$ (and therefore its exposure) calculated on each iteration is a function of the volatility model through its parameters $F_T(0)$ and $\sigma(T)$, and a pseudo- or quasi-random number generator. This means that the exposure values that are projected down the valuation stack are different on each iteration for the payoff and state-variable modeling layers, but the exposure of the model parameters to market data is not (***Eq. (38)*** and ***Eq. (39)***). The model is calibrated before the simulation, with the result that the relationship between the model parameters and the input market data remains fixed for all iterations. This structure is summarized in ***Fig. 7*** which shows the levels through which exposure must be projected in such a valuation, from V_0 at the top through to $\vec{\sigma}$, S_0, $\vec{r}$, $\vec{q}$ and $\vec{c}$ at the bottom.

7.2 Underlying Projectors

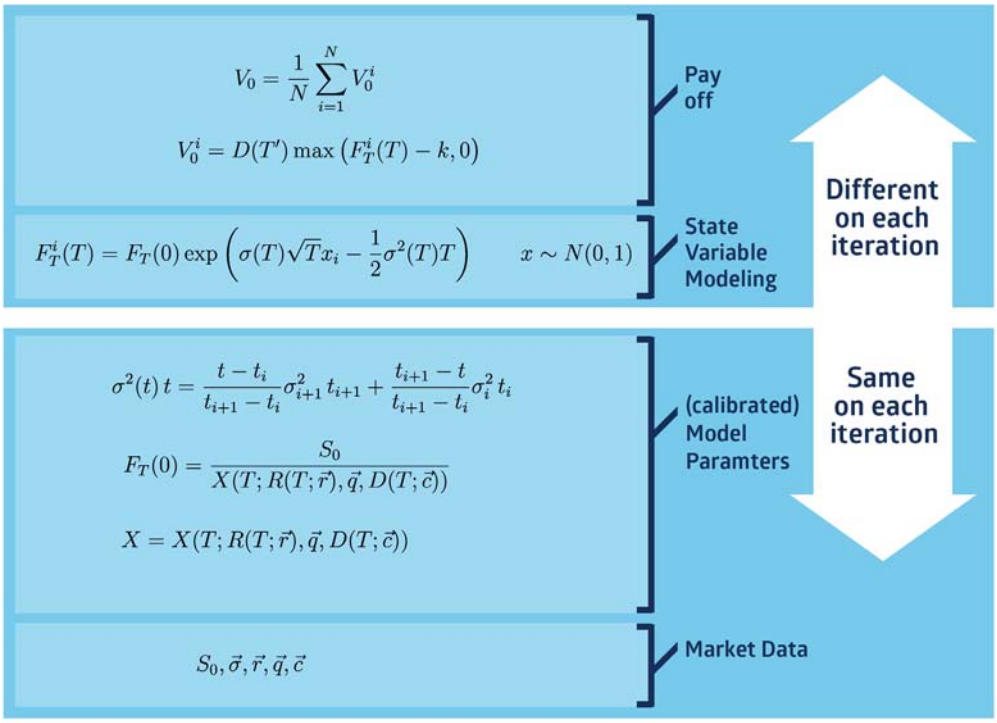

Fig. 7. *The valuation stack for the toy example of a European call option valued with a Monte Carlo simulation, showing the separation between per-iteration quantities and those that are constant across iterations. This section describes the capability to split the total exposure projection along the horizontal boundary shown by dashed line, delaying the full projection until later.*

The computational expense of projecting exposure through a calibration is a function of the complexity of the functional relationship that encodes the Jacobian between the model parameters and the market data. It is not a function of the weight supplied to the `projectExposure` call that initiates the projection. We would therefore waste effort by projecting exposure through the entire valuation stack on every iteration. Instead, while performing each iteration, we should treat the model parameters as the leaves of the calculation tree. Once the simulation is over and the (both value and exposure) contribution from all the iterations has been accumulated, we can complete the projection through the calibration.

In order to achieve this separation we require the ability to work with Jacobians directly. This approach is, in a sense, the opposite of exposure projection, which emphasizes root-to-leaf traversals of the calculation tree, whereas each column of a Jacobian pertains to the boundary between one node and its children.

7.2 Underlying Projectors

In the stack shown in ***Fig. 7***, exposure projection operates vertically but Jacobians are horizontal, connecting each layer to the next. For scalar-valued objects, this Jacobian capability is a specialization of the Exposure concept, because exposure projection can be implemented in terms of it. The following code provides a schematic example of what such an interface could look like.

```
struct UnderlyingProjections : public Exposure
{
  // Fill a vector of partial derivatives with respect to immediately underlying variables
  virtual partialDerivatives( std::vector< double >& target ) const = 0;
  // Provide access to the objects which can project exposure onto the underlying variables
  virtual const std::vector< const Exposure* >& exposureProjectors ( void ) const =
  // Implement full exposure projection in terms of the above interface
  virtual void projectExposure( double weight,
                                exposures_t& target ) const;
};
```

If an object $f(a_0, a_1, \ldots, a_N)$ implements this interface, then the two pure virtual member functions provide access to the collections $\left\{\frac{\partial f}{\partial a_i}\right\}$ and $\{\mathrm{d}a_i\}$ in the full exposure projection

$$\mathrm{d}f = \sum_{i=0}^{n} \frac{\partial f}{\partial a_i} \mathrm{d}a_i,$$

which can be implemented by looping over the exposure projectors returned by `exposureProjectors` calling each with a weight given by the corresponding element of the vector filled by `partialDerivatives`.

With an explicit Jacobian capability like that above, you can avoid needless per-iteration repetition of exposure projection through a calibration in a Monte Carlo simulation, which is just one of many possibilities for saving computational effort. Another important consequence of the ability to separate a group of levels in an exposure projection stack from another is to reduce the payload size in distributed and cloud computing scenarios. When transporting the information to run a subset of Monte Carlo iterations over a network, it is important to minimize the overhead of distribution. This means that we want to send the bare minimum information for a worker node to calculate. Such a worker node does not need to know anything about calibration, other than the resulting model parameters and how they are used to generate the relevant distributions.

Later, in ***Sec. 8 (Page 43)***, when we compare EP with alternative approaches available in the literature, we will see that those alternative approaches work directly in terms of Jacobian matrices (or vectors for scalar-valued functions, as above). The `partialDerivatives` member function provides access to this vector for a given node in the calculation tree. In contrast, its companion member function `exposureProjectors` is new, as is the associated concept of exposure projection as defined in ***Definition 4***.

In effect, slicing calculation trees (such as that shown in ***Fig. 4***) horizontally is a fundamental feature of alternative approaches found in the literature. To us, however, it is a suitable approach in some situations only and therefore available as an optimization, but important to separate from the fundamental abstractions such as ***Definition 4*** that underpin any generic approach to analytic exposure calculation.

7.3
Peeking Through

As our third and final illustrative example of how we can optimize exposure projections whose calculation trees fall into specific structural categories, consider a scenario where, as in ***Subsec. 7.2 (Page 38)***, we wish to query an object for an explicit list of partial derivatives and projectors.

Suppose that this object was an Engine that calculated the price of a barrier option by propagating the payoff at each time backward in time by means of an expectation over the underlying distribution's transition density (typically performed in Fourier space, see Cherubini (2010)). The number of backward propagations is equal to the number of times that a possible breach of the barrier is observed and can be quite high. Even a one-year option monitored daily requires approximately 250 observations of the underlying.

Each observation of the underlying requires an evaluation of the relevant model parameters, which in turn are likely to have a term structure. The model parameter term structures, if coming from a calibration, will have been calibrated to market quotes, but the number of maturities per year of suitable liquid calibration instruments is of the order of one, not hundreds.

Fig. 8 gives a schematic illustration of this scenario, where the calculation tree flares out when a large number of observations of underlyings and model parameters is made, but tapers back in when the tree recombines on the relatively small number of interpolation points in the term structure for each model parameter.

Fig. 8. *Illustration of the flaring out of an exposure projection tree when a large number of observations are made of an underlying, resulting in a large number of evaluations of a curve, formed by interpolating a much smaller number of points.*

A Fourier-space backward propagation calculation such as our current example works on a set of samples of the underlying. In order to achieve sufficient accuracy, it is usually necessary to adopt a sampling granularity that results in the Fourier integral being performed over hundreds or thousands of values of the integrand. When combined with hundreds or thousands of elements in the vectors populated by the `UnderlyingProjections` interface, the number of distinct projectors can approach 10^6.

The key to the "peek-through" optimization is the observation that there is nothing in the `UnderlyingProjections` interface that requires an object to report its exposure to its immediate underlyings. The only requirement is that it can split the calculation stack at some level and provide the corresponding list of partial derivative values and exposure projectors.

To provide a sketch of how this idea works in the context of ***Fig. 8***, write the barrier pricing engine's calculation of the option price P as

$$P = B(\{\sigma_i\}) \quad i = 1 \dots I$$

where the I model parameters $\{\sigma_i\}$ are obtained by evaluating a term-structure $\sigma(t)$ over a large collection of I times $\{t_i\}$.

$$\sigma_i = \sigma(t_i; \{q_j\}) \quad j = 1 \dots J,$$

where $\{q_j\}$ is a small collection of J interpolation points. We are free to relegate the $\{\sigma_i\}$ to the status of an internal implementation detail and instead have the `UnderlyingProjections` interface work in terms of

$$\mathrm{d}P = \sum_{j=1}^{J} \frac{\partial B}{\partial q_j} \mathrm{d}q_j$$

where the partial derivative values are calculated internally as

$$\frac{\partial B}{\partial q_j} = \sum_{i=1}^{I} \frac{\partial B}{\partial \sigma_i} \frac{\partial \sigma_i}{\partial q_j}.$$

This ability to group together an arbitrary number of levels in the calculation stack is another example of the kind of optimization that makes the difference between an interesting academic topic and a mature production implementation of an analytic exposure computation platform.

8.0 Automatic Differentiation and Exposure Projection

Some fundamental techniques for the analytic computation of partial derivatives have been known among computer scientists for several decades (Rall (1981)), Griewank (1989)), in the form of Automatic Differentiation (AD). Correspondingly, these techniques have been applied to problems in quantitative finance for many years; the analytic computation of discounting risk was commonplace in sell-side institutions in the mid-90's.

Due to the highly applied nature of quantitative finance, key innovations are almost always monetized before being published, with many implementation details remaining permanently beyond the realm of academic literature.

The application of AD to financial problems has accumulated, however, a body of literature. The popularization of AD in finance began with Giles and Glasserman (2006). Since then, a range of papers describing AD applied to a collection of specific financial problems has appeared (see Homescu (2011) for a useful review). In this section we compare EP to AD and find that many of the ideas presented in this article are new; EP is an alternative approach to those available in the AD literature, with several practical advantages. Where there is common ground it is that both methods apply the chain rule of differential calculus, which is a necessary similarity between AD and any alternative method that differentiates functional relationships in software.

Before examining the advantages of EP in ***Subsec. 8.2 (Page 45)***, we conduct a very brief survey of AD.

8.1 Brief Summary of Automatic Differentiation

As described at www.autodiff.org, Automatic Differentiation (AD) is a set of techniques based on the mechanical application of the chain rule to obtain derivatives of a function given as a computer program.

Consider the function $h : \mathbb{R}^m \to \mathbb{R}^n$ formed by composing the functions $f : \mathbb{R}^p \to \mathbb{R}^n$ and $g : \mathbb{R}^m \to \mathbb{R}^p$ such that

$$\begin{aligned} \vec{y} = h(\vec{x}) &= f(\vec{u}) \\ \vec{u} &= g(\vec{x}) \end{aligned}$$

for $\vec{x} \in \mathbb{R}^m$, $\vec{u} \in \mathbb{R}^p$ and $\vec{y} \in \mathbb{R}^n$. The chain rule allows us to decompose the $n \times m$ Jacobian J of h as follows:

$$J_{ij} = \left[\frac{\partial h(\vec{x})}{\partial \vec{x}}\right]_{ij} = \frac{\partial y_i}{\partial x_j} = \sum_{k=1}^{p} \frac{\partial y_i}{\partial u_k}\frac{\partial u_k}{\partial x_j} = \sum_{k=1}^{p} \left[\frac{\partial f(\vec{u})}{\partial \vec{u}}\right]_{ik} \left[\frac{\partial g(\vec{x})}{\partial \vec{x}}\right]_{kj} = \sum_{k=1}^{p} A_{ik}B_{kj} \tag{40}$$

where A and B are the Jacobians of f and g respectively. There is a straightforward generalization to an arbitrary number of function compositions, so we choose a single composition here for convenience and without loss of generality. Suppose that a computer program contains implementations f of g and explicitly, with h formed implicitly by supplying the output of g to a call to f.

AD defines two approaches to computing J, forward (or tangential) and reverse (or adjoint) accumulation. In forward accumulation, B is computed first, followed by A. In other words, the calculation tree for operation performed by h is traversed from its leaves to the root. The computational cost of such an approach scales linearly with the number of leaves because the calculation needs to be "seeded", that is repeatedly evaluated with $\vec{x} = \mathrm{diag}(\delta_{jq})$ on the q^{th}

accumulation. This is well-suited to problems where $n \gg m$, because it allows all m rows in J to be computed simultaneously for the q^{th} accumulation. Such problems are so hard to find in the context of financial derivative valuation that it is very rare to read of any application of forward accumulation in the literature.

In contrast, reverse accumulation is most efficient when $n \ll m$ and is closer in spirit to the idea of exposure projection presented in this article. It consists of two stages - a "forward sweep", where the relevant partial derivatives (termed "work variables") are formed, and then a "backward sweep" where the relevant products of partial derivatives are added into each element of A and B, which can then be multiplied to obtain the full Jacobian J.

In the AD literature, one finds main two approaches to implementation, whether forward or reverse accumulation, for the function h. Both approaches emphasize the problem of adding a differentiation capability to an existing codebase that computes the value alone. The first method, source code transformation is based on a static analysis of the source code for the function h. New source code is generated for a companion function that computes J, then both are compiled (or interpreted).

The second method, operator overloading, requires a modern language such as C++ or Java where basic types can be redefined and operators can be overloaded, so that existing code that performs these operations will trigger the corresponding derivative calculations also. Forward accumulation is easier to implement in an operator overloading approach than reverse. A common technique for reverse accumulation is to generate a "tape" that records the relevant set of operations, then interpret that tape in order to obtain the desired derivatives.

Both approaches suffer from high storage costs and long run-times, with the result that numerous implementation techniques have been devised to mitigate the performance challenges inherent with AD and it remains an active area of research.

8.2 Comparison with Exposure Projection

AD and EP both leverage the chain rule of differential calculus to compute derivatives analytically. Given that the chain rule is just the name given to the correct mathematics for differentiating nested functional relationships, it is not surprising that they share this common link.

In a modern programming language, once a numerical algorithm's data types are substituted with custom types supporting AD and the core operators are overloaded for those types, differentiation is truly automatic - the numerical algorithm's code remains unchanged and no further work is needed to obtain its derivatives. The penalty for this automation, however, is performance, with challenges in both storage and computational time. In contrast, EP is not automatic in this sense - it explicitly requires derivatives to be implemented for each Engine, function and Parameter used in the calculation. The simple requirement that the Exposure interface is implemented allows quants and developers to choose the optimal granularity for the problem at hand, yielding implementations that are efficient in both memory and time.

8.2 Comparison with Exposure Projection

In EP, further optimizations are possible, such as flattening and peek-through (***see Sec. 7 (Page 31)***), that facilitate the kind of fine-tuning of exposure calculations in a financial context that one expects to see in an approach that itself is optimized for the kind of problems found in financial derivative valuation and risk management. Like the conceptual framework of exposure projection described in ***Subsec. 3.4 (Page 13)***, these optimization are new.

Many AD tools are available which retrofit a differentiation capability to existing code. Whether based on operator overloading or source-code transformation, these tools are forced to work at the granularity of the expressions in the existing code. The choice of granularity available in EP arises because it was conceived before development started on F3 - it was built in from the start - and both object-oriented programming techniques and a careful development process ensure that all future valuation functionality supports EP.

Much of the AD literature describes techniques for computing partial derivatives (exposures) to a known number of independent variables (risk factors). This leads to exposure calculations whose structure is constrained by the set of quotes for a given valuation and results ultimately in systems that calculate exposure to a fixed number of quotes, or specific types of quotes only. In contrast, EP not only computes exposure to the relevant set of risk factors, but it also selects that subset of relevant risk factors from the total set of market observables in the Model, describing them in terms of the natural information content of financial market data, as described in ***Subsec. 3.1 (Page 6)***.

Papers in the AD literature almost exclusively consist of applications to specific problems, of calculating exposure to known risk factors in the context of a specific model and valuation methodology. A handful of papers conduct a more general discussion and, at the time of writing, a growing number of banks have publicly indicated that they are currently applying AD techniques in some systematic manner in their next-generation library development. This article is the first to introduce ideas that facilitate a truly generic implementation of analytic exposure calculation - a guarantee that analytic exposure is available for every valuation. In addition, we provide an example, in the form of F3, of a mature implementation.

One of F3's hallmarks is its generic nature - any derivative or portfolio can be valued under the joint distribution for an arbitrary set of underlyings (Gibbs and Goyder (2013)). With its implementation of EP, Universal Algorithmic Differentiation™ (UAD), the same is true of analytic exposure computation. The result is an analytic exposure capability that is truly universal, in its comprehensive coverage and stable, mature implementation. Every valuation, in every model and for every valuation method, from closed-form though backward propagation in Fourier space to hybrid Monte Carlo, has EP. Every sharp edge is smoothed, from a simple $\max$ through conditions to sorting operations. EP is available for all types of risk factors and in every valuation output, whether individual trades, portfolios or CVA on any type of trade. A rich set of applications that leverage analytic exposure computation, from hedging notionals, calculating hedging costs to generic gamma/ convexity and profit-and-loss attribution, is available. At every level, from end-user functionality to low-level debugging and testing parts of the API, EP output is available.

9.0 Applications

In this section, we show two applications of UAD. Both are based on the same underlying portfolio of about 250 derivative trades and all calculations are done on a Desktop PC with an Intel Core i7 CPU. Approximately 80% of the trades are 10-year vanilla swaps paying 3-month USD Libor against a semi-annual coupon. The remaining 20% comprises swaptions, CDS, FRA, vanilla EUR swaps, USD-EUR cross-currency swaps, FX forwards and equity options.

9.1 Hedging a Derivatives Portfolio

In this example we demonstrate the performance of F3's UAD in a closed-form valuation setting by performing the calculations necessary to hedge our portfolio's market risk. This market risk comprises:

- the equity's spot price, funding rate, expected dividend term-structure and each point in its volatility surface
- each cash rate, futures price and swap rate in each currency
- each point in the term-structure of volatility chosen for the futures convexity adjustment, in each currency
- each tenor basis spread in each currency
- each swaption quote used to construct the volatility cube
- each CDS quote used to build the reference entities' survival curves, and each point in the recovery rate term structure
- the FX spot quote

The total number of values in this set is 411. ***Fig. 9*** shows the time taken to calculate the exposure to each of these numbers using UAD and compares it to a finite difference calculation. In this particular example, UAD yields about a factor of 600 speed-up. The speed improvement afforded by UAD depends, naturally, on the content of the portfolio being valued and the modeling assumptions, and corresponding market data, used for the valuation. In practice, for most practical problems, it ranges from about a factor of 10 to a factor of 10,000.

9.1 Hedging a Derivatives Portfolio

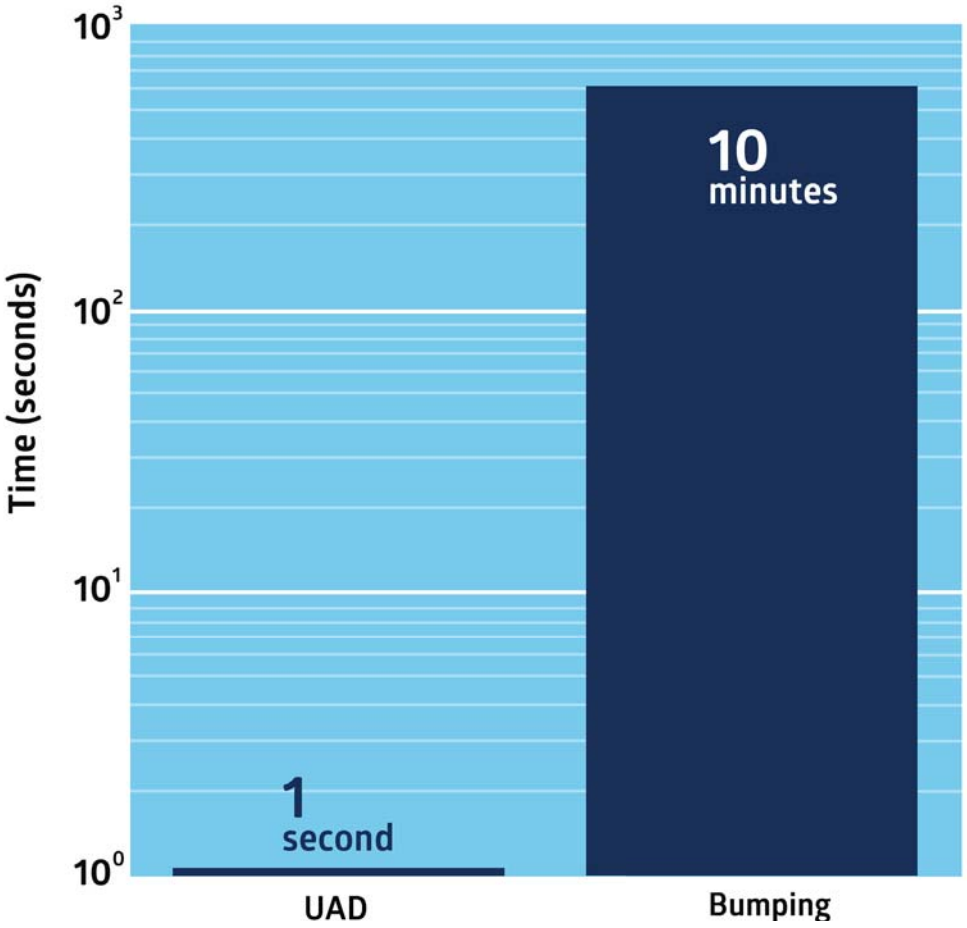

Fig. 9. *Time taken to calculate the exposure of a cross-asset, multi-currency portfolio of 242 vanilla trades to every quote on which the pricing depends, using F3's Universal Algorithmic Differentiation™ (UAD, left) and by finite difference (Bumping, right). Note the logarithmic scale on the y-axis. In this particular example, UAD yields about a factor of 600.*

The report generated by UAD for this portfolio provides a very detailed view of its risk profile and contains too much information to display here in its entirety. However, we can explore some specific areas to get a feel for the rich nature of the information content. For example, ***Fig. 10*** shows the effect of a percentage point move in each quoted option volatility for the equity. Such an exposure surface is provided by UAD for each equity underlying the derivatives in a portfolio.

9.1 Hedging a Derivatives Portfolio

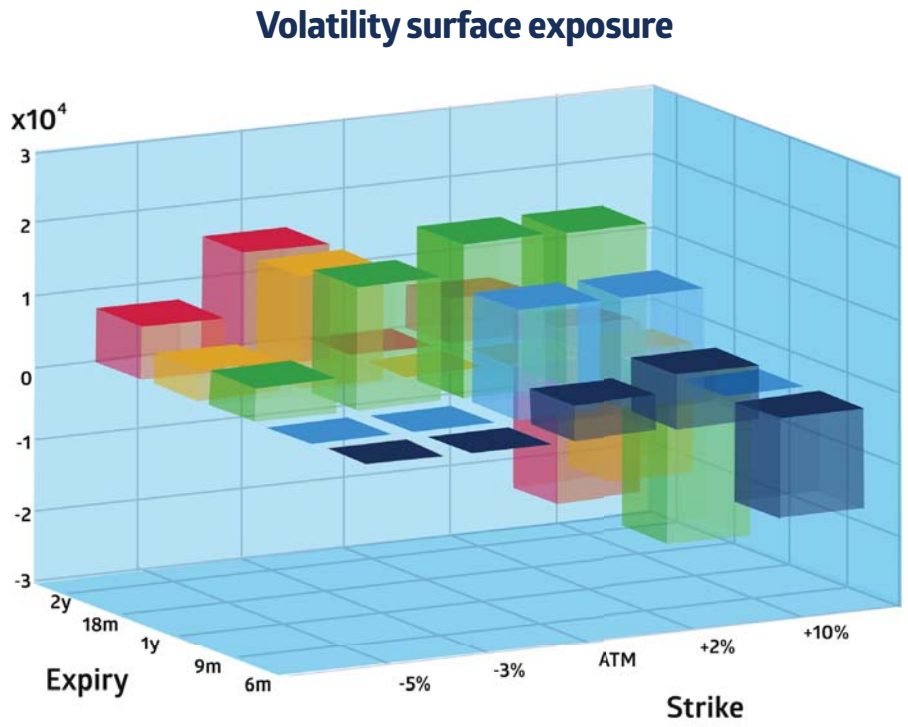

Fig. 10. *Change in portfolio value resulting from a one percentage point change in quoted option volatility for the underlying equity. UAD gives the exact partial derivative of portfolio with respect to each quote, which is then scaled by 1% to form the equity vega surface shown here. To obtain the overall vega value resulting from a parallel 1% shift in all quoted option volatilities, we simply sum the values shown in this plot.*

While the fundamental quantity calculated by UAD is exposure; the partial derivative of portfolio value with respect to a given quote, it is often more useful to present the information in terms of a corresponding hedge. Given the exposure $\Delta_i = \frac{\partial V}{\partial s_i}$ of the portfolio's value V to the i^{th} quote s_i, this is a straightforward calculation. Suppose s_i is the par rate of an interest rate swap with annuity A and floating leg value B (per unit notional), following the notation of ***Sec. 7 (Page 31)***. We seek the notional amount H_i of this swap that, when combined with our portfolio, eliminates the exposure to small moves in s_i. We require

$$0 = \frac{\partial}{\partial s_i}\left(V + H_i(s_i A - B)\right)$$

and so

$$H_i = -\frac{\Delta_i}{A}.$$

9.1 Hedging a Derivatives Portfolio

UAD provides H_i for each swap, and appropriate measures for other instruments such as the required number of futures contracts. Some of these values are displayed in ***Fig. 11*** for the current example portfolio, where we can see that our exposure to USD curve instruments is dominated by the T/N and 2-month cash deposit rates, and the 10-year swap rate.

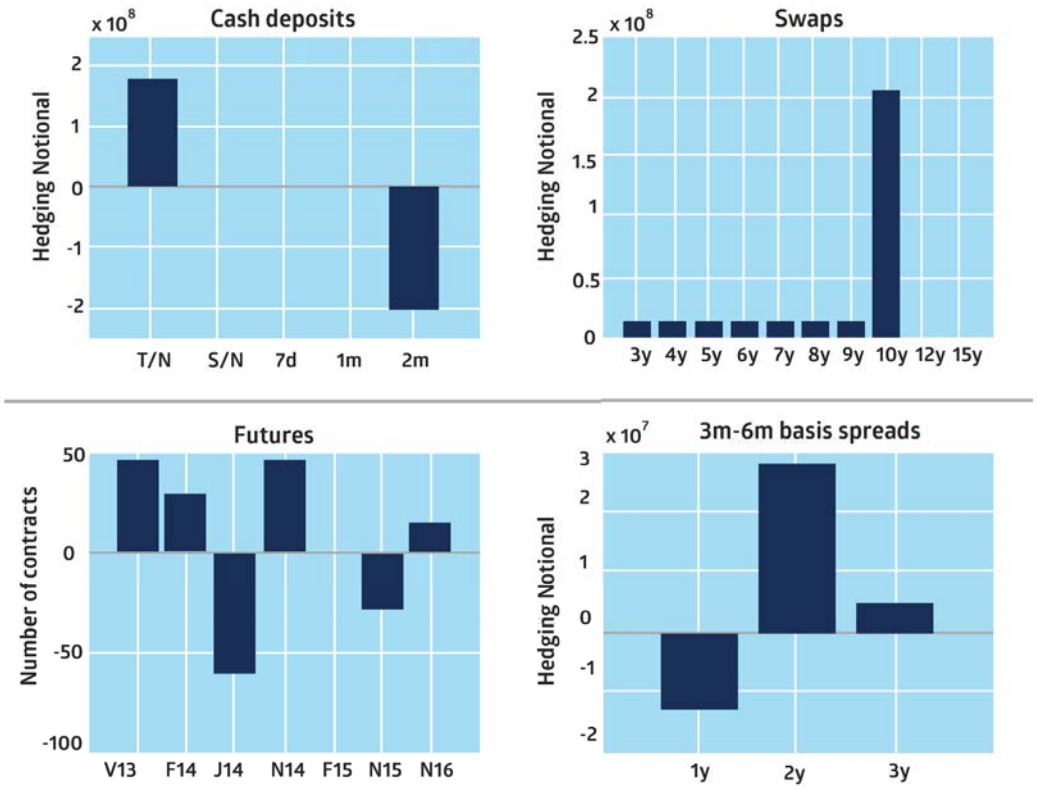

Fig. 11. *Portfolio exposure to USD curve instruments, given in terms of the equivalent hedge.*

The example shown here is representative, not exhaustive. UAD allows the calculation to scale to large portfolios, complex modeling assumptions and different valuation methodologies such as Monte Carlo and backward evolution approaches. In ***Subsec. 9.2 (Page 51)*** we show UAD at work in a Monte Carlo valuation.

9.2
Hedging CVA

We now switch to a Monte Carlo valuation setting and calculate the Credit Value Adjustment (CVA) for this portfolio. Then we apply UAD to the CVA itself, to study its response to market fluctuations and address the additional market risk that it introduces.

9.2.1
CVA Trade Structure

The CVA of the portfolio V can be expressed as

$$\text{CVA} = E\left[\int_0^T \max\left[\hat{V}(t), 0\right] \mathcal{N}(t)\,(1 - R(t))\, I_{\tau\in(t,t+\mathrm{d}t)}\right] \tag{41}$$

where $R(t)$ is the recovery rate at time t, $\hat{V}(t)$ the value at time t of the remainder of the underlying portfolio whose value is $V(t)$, $\mathcal{N}(t)$ the appropriate numeraire factor (for example $P(0,T_\star)/P(t,T_\star)$ in the $T_\star$-forward measure) for a cash flow at time t, τ the time of counterparty default and $I_{\tau\in(t,t+\mathrm{d}t)}$ is a default indicator, evaluating to 1 if τ lies between time t and $t+\mathrm{d}t$ and 0 otherwise. In order to proceed numerically, it is necessary to approximate this time integral. Subdividing into N time intervals (t_k, t_{k+1}) which are sufficiently small such that any variation of $\hat{V}(t)$ within them can be neglected, the integral is

$$\begin{aligned}\text{CVA} &= E\left[\sum_{k=1}^{N} \max\left[\hat{V}(t_k), 0\right] \int_{t_{k-1}}^{t_k} \mathcal{N}(t)\,(1 - R(t))\, I_{\tau\in(t,t+\mathrm{d}t)}\right] \\ &= \sum_{k=1}^{N} E\left[U_{k-1,k}\right] \qquad (42)\end{aligned}$$

where

$$U_{k-1,k} = \max\left[\hat{V}(t_k), 0\right] D(t_k) L(t_{k-1}, t_k) \tag{43}$$

$$L(t_{k-1}, t_k) = \frac{1}{D(t_k)} \int_{t_{k-1}}^{t_k} \mathcal{N}(t)\,(1 - R(t))\, I_{\tau\in(t,t+\mathrm{d}t)}. \tag{44}$$

Eq. (44) gives the rate of loss over the k^{th} time interval given that a credit event occurred within it and ***Eq. (43)*** describes the payoff of a product whose value is the loss that would result from counterparty default occurring within the k^{th} time interval. The expected value of this product is the contribution of the given time interval to the total integral in ***Eq. (41)*** which spans the length of the underlying product.

In this example, we calculate not only CVA, but the market risk of CVA for each of the several hundred quotes on which it depends. For the i^{th} market quote s_i, the exposure of CVA to it is defined as

$$\frac{\partial}{\partial s_i}(\text{CVA})$$

where CVA is given by ***Eq. (42)***. That CVA is a tradeable quantity is evident from ***Eq. (41)***, which expresses a just another derivative valuation, albeit a complex one. The derivative product comprises a portfolio of N options on the remainder of the portfolio V, scaled by a loss-given-default observable. F3 is capable of describing trades at a very generic level, which means that the necessary core operations are available and work with arbitrary underlying trades. These operations are:

1. **Portfolio:** form a trade that is the sum of others, as in ***Eq. (42)***.
2. **Date restriction:** form a trade by imposing a time window on underlying trade. If the time window extends beyond the latest payment in the underlying trade then we obtain the remainder $\hat{V}(t)$.
3. **Scaled cash flows:** form a trade from another by scaling each cash flow amount by an observable. If the observable is the loss-given-default from Eq. (44) then we obtain default-contingent flows.
4. **Conditional Choice:** Form a trade that represents the right to choose between two underlying trades at some future time. If the second trade is a null trade that is without value, then this yields the positive part of the first.

9.2.2 Modeling and Valuation

In order to calculate CVA, we need to evaluate each of the expectations in ***Eq. (42)***. It is common practice at this point to make various assumptions about the lack of correlation among the set of state variables in V and the credit variables $R(t)$ and $I_{\tau\in(t,t+\mathrm{d}t)}$ in order to simplify the valuation. Furthermore, it is typical to rework the implementation each time a modeling assumption changes, even though the CVA expressions above do not depend on modeling assumptions, except through the details of the expectation operator $E[\ ]$.

In contrast, F3 separates the description of what is valued (trades) from how it is valued (modeling). Changing modeling assumptions across the full spectrum from highly simplified uncorrelated treatments to a complex hybrid simulations becomes a matter of configuration, not implementation. In our example, we have the following state variables:

5. USD interest rate term structure
6. EUR interest rate term structure
7. The USD-EUR FX rate
8. One equity asset
9. The counterparty's survival

We choose low-dimensional Hull-White models for each interest rate term-structure, a lognormal assumption for the FX rate and the Heston model, with time-dependent parameters, for the equity. While we are free to model wrong-way risk by choosing a dynamical model for the survival of the counterparty, in this example we neglect its contribution by choosing static credit model, thereby decoupling the dynamics of survival from the rest of the state variables.

9.2.2 Modeling and Valuation

The calculation is performed by Monte Carlo simulation, using a technique called Automatic Numeraire Corrections (Gibbs and Goyder (2013)) in order to reconcile the different measures in which each marginal distribution is simulated. The resulting joint distribution is self-consistent and accounts for the correlation between the four state variables in the simulation. Each conditional choice arising from the $\max$ in ***Eq. (43)*** is processed via a backward Monte Carlo algorithm, which is general enough to handle essentially arbitrary trades within the underlying portfolio. For example, the European swaptions in portfolio V could be replaced with Bermudan swaptions, or some callable Libor exotic, or even exotic hybrid trades, and the same valuation instructions would yield the CVA calculation appropriate for the modeling assumptions described above.

We used a Sobol quasi-random number generator and found that $2^{15} - 1$ iterations were sufficient to achieve convergence in this example. Owing to an additional volatility cube and other modeling parameters, including correlations between the Brownian motions driving the simulation, the number of quotes to which the CVA is exposed was 542. The computation time for calculating CVA was about 10 minutes on a desktop machine. Consequently, by employing traditional finite difference methods, exploring the degree to which the CVA depends on local changes in each of the 542 market risk factors, in order to find the relevant set, is impractical without significant computational resources and implementation effort to distribute the work.

In practice, intuition is employed. A CDS on the counterparty, if one exists, is the canonical hedge for CVA. However, there is no fundamental rule that says CVA is a stronger function of the counterparty's CDS spread than other market risk factors - in practice, it depends on the underlying portfolio. With F3's UAD, it is not necessary to fall back on intuition because calculating the exposure of CVA to each of the 542 quotes takes just 4 times as long as the valuation of CVA. The resulting risk report can then be sorted in order of decreasing exposure size to reveal which quotes have the strongest influence on changes to CVA. ***Fig. 12*** shows the first 50 of these 542 exposures. In this example, we can see that although the counterparty's CDS spreads rank in the top 20, the exposure is dominated by the US vanilla rates market.

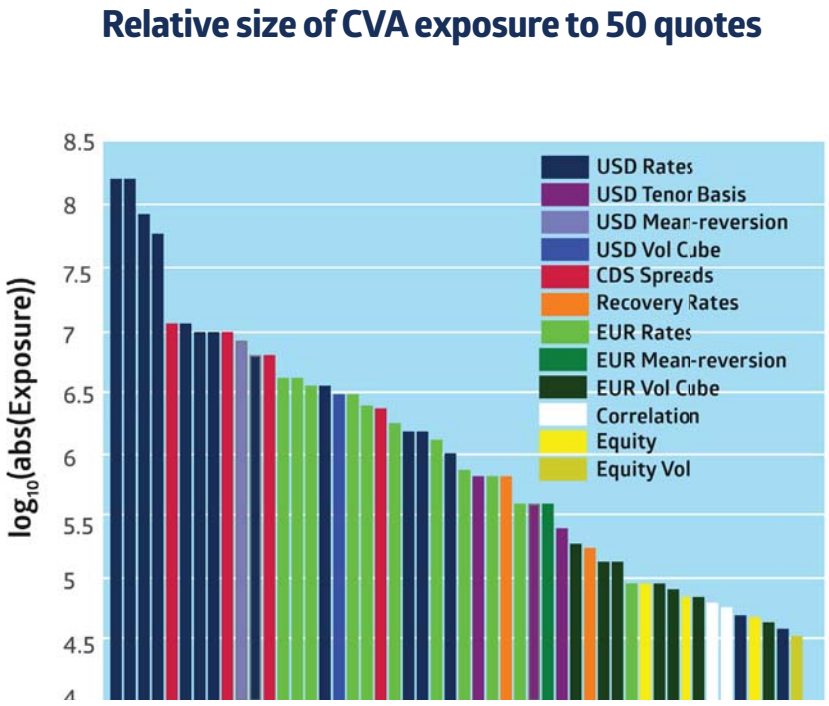

Fig 12. *Ordered list of the 50 highest values of the market risk of our example portfolio. Each bar is (the logarithm of the magnitude of) the partial derivative of CVA with respect to an individual market quote. The bars have been grouped into categories, shown by color. The entire set of 542 exposures, of which the top 50 is shown, took approximately 4 times as long as the valuation of the CVA itself.*

10 Conclusion

We introduced Exposure Projection (EP) in ***Sec. 3 (Page 6)***, a new method for the fast calculation of first-order exposure that yields several advantages over existing methods in the literature. Existing approaches - notably (adjoint) Automatic Differentiation (AD) - suffer from challenges in both storage and run-time performance, when managing large collections of exposures to many different factors. In contrast, implementations of EP (such as F3's Universal Algorithmic Differentiation™) naturally induce an efficient storage scheme within a program's stack and afford opportunities for optimization. The resulting implementation exhibits efficiencies in both storage requirements and speed.

Despite the simplicity of the underlying mathematics, implementations of AD remain focused on specific applications, calculating some risk factors for a particular model or class of models, or for a specific set of trades, or both. With EP, however, it is possible to address the issue of analytic exposure calculation generically - once for all combinations of trade types, models and valuation methodologies. To this end, we focused on the principles underlying the construction of a generic valuation platform in which EP takes center stage at all levels of the valuation stack, from Parameters at the bottom through functions to Engines at the top. The generic form of the exposure of calibrated model parameters to market data was given in ***Sec. 5 (Page 21)***, a treatment of discontinuities, including those present in sorting algorithms was covered in ***Sec. 6 (Page 28)*** and an indicative sample of the types of optimization that are possible in EP was presented in ***Sec. 7 (Page 31)***. Finally, in ***Sec. 8 (Page 43)***, we conducted a comparison of the salient aspects of AD with EP and listed a number of ways in which EP represents an advancement over the state of the art.

F3 is a generic valuation platform that contains a complete implementation of EP that draws on all of the ideas presented in this article, called Universal Algorithmic Differentiation™ (UAD). With UAD, the entire exposure of virtually any derivative or portfolio can be calculated in any model and in under any valuation approach. In ***Sec. 9 (Page 47)*** we demonstrated the application of UAD to a multi-currency, cross-asset portfolio of about 250 trades and calculated its exposure to over 400 quotes in 1 second. We then demonstrated UAD in a hybrid Monte Carlo setting, calculating over 500 hedge factors for the portfolio's CVA.

11 Acknowledgements

The authors are grateful to Lois Patterson for help in reviewing the manuscript and Geoff Lynch, Anindya Mukherjee and Glen Goodvin for help in preparing the examples.

Bibliography

[1] Gibbs and Goyder (2012), The Past, Present and Future of Curves, Technical Article, FINCAD, http://www.fincad.com/resources/resource-library/whitepaper/past-present-and-future-curves-fundamentals-modern-curve

[2] Cherubini (2010), Fourier Transform Methods in Finance, Wiley

[3] Goyder and Gibbs (2012), Optimal Architecture for Modern Analytics Platforms, Technical Article, FINCAD Inc., http://www.fincad.com/resources/resource-library/whitepaper/technical-paper-optimal-architecture-modern-analytics

[4] The Standard Template Library, ISO/IEC 14882:2003(E) Programming Languages - C++ Sec.17-27

[5] Jaynes (2003), Probability Theory, Cambridge University Press

[6] Batcher (1968), Sorting Networks and their Applications, in Proc. AFIPS Spring Joint Comput. Conf., 307-314

[7] Rall (1981), Automatic Differentiation: Techniques and Applications, Lecture Notes in Computer Science, Springer, 120

[8] Griewank (1989), On Automatic Differentiation, Mathematical Programming: Recent Developments and Applications, Kluwer Academic Publishers

[9] Giles and Glasserman (2006), Smoking Adjoints: fast evaluation of Greeks in Monte Carlo Calculations, Risk Magazine, (19:1), 92-96

[10] Homescu (2011), Adjoints and automatic (algorithmic) differentiation in computational finance, Social Sciences Research Network

[11] Gibbs and Goyder (2013), Automatic Numeraire Corrections for Generic Hybrid Simulation, Technical Article, FINCAD, http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2311740

Disclaimer

FINCAD makes no warranty either express or implied, including, but not limited to, any implied warranties of merchantability or fitness for a particular purpose regarding these materials, and makes such materials available solely on an "as-is" basis. In no event shall FINCAD be liable to anyone for special, collateral, incidental, or consequential damages in connection with or arising out of purchase or use of these materials. This information is subject to change without notice. FINCAD assumes no responsibility for any errors in this document or their consequences, and reserves the right to make improvements and changes to this document without notice.

Copyright

Trademarks

Revisions

Every effort has been made to ensure the accuracy of this document. FINCAD regrets any errors and omissions that may occur and would appreciate being informed of any errors found. FINCAD will correct any such errors and omissions in a subsequent version, as feasible. Please contact us at:

FINCAD
Central City, Suite 1750
13450 102nd Avenue
Surrey, BC V3T 5X3 Canada

or

Block 4, Blackrock Business Park
Carysfort Avenue, Blackrock
Dublin, Ireland

www.fincad.com

Document Information

Document Name: Universal Algorithmic Differentiation™ in the F3 Platform