Efficient higher-order PnL Explain

Niels Lykke Sørensen
PFA Pension, nls@pfa.dk

June 22, 2023

Abstract

Using a combination of nested forward-mode and reverse-mode automatic differentiation, we show how to uniquely decompose the change in a scalar-valued function incorporating all higher-order effects up to a given level.

The method is highly efficient. The complexity scales exponentially in included orders but is constant in the number of input variables. E.g. including all 1^{st}, 2^{nd} and 3^{rd} order effects has a computational time equal to $\sim 1 + 2 + 4 = 7$x of calculating 1^{st} order effects only.

Within finance, this may be used to fully attribute the net PnL in a trading book to each of the underlying input factors. Traditionally, doing such *PnL Explain* has been done by calculating all 1^{st} order effects but only selected 2^{nd} order effects due to time and memory complexities resulting in unexplained residuals.

1 Forward *vs* Reverse

The use of Algorithmic Adjoint Differentation (AAD) - also known as *reverse-mode automatic differentation* - is becoming more widespread within quantitative finance. See Savine[3] for implementation of a professional-grade quantitative risk library with integrated reverse-mode automatic differentation.

Without going into the specifics of reverse-mode, it is especially well suited for calculating the gradient of a scalar-valued function with many inputs. E.g. the gradient vector $\nabla f(x)$ for $f : \mathbb{R}^N \to \mathbb{R}$ may be calculated in a time approx. $\sim$ 3x the time required for a single evaluation of $f(x)$ - regardless of the size of $x \in \mathbb{R}^N$. This explains its popularity within finance, where the full gradient vector for a pricing function depending on a large number of inputs may be calculated fast and exact.

On the other hand, the dual of reverse-mode automatic differentation, *forward-mode automatic differention*[4], is less used within finance. Instead, forward-mode excels for functions $f : \mathbb{R}^M \to \mathbb{R}^N$ with $M << N$ - a case typically less frequently encountered.

In this paper, we show how the combination of nested forward- and reverse-mode automatic differentation may be used to efficiently calculate higher-order directional derivatives.

This allows for doing a full attribution, *PnL Explain*, of the Profit & Loss (PnL) for a portfolio of financial instruments to different input factors.

2 Traditional PnL Explain

Traditionally, attributing the changes in value (PV=Present Value) for a portfolio of financial instruments has been done with a 2nd order Taylor expansion around the initial state of input variables, $x \in \mathbb{R}^N$.

I.e. as x changes to $x + v$ for $v \in \mathbb{R}^N$ we have:

$$\mathrm{PV}(x+v) - \mathrm{PV}(x) = v^T \nabla_{\mathrm{PV}}(x) + \frac{1}{2} v^T H_{\mathrm{PV}}(x) v + \mathcal{O}(||v||^3) \tag{1}$$

Hence, decomposing the change in PV is a matter of succesively calculating the above while zero'ing selected entries in v. However, for big portfolios and/or complicated instruments, N may be very large - rendering the calculation of the Hessian $H_{\mathrm{PV}}(x) \in \mathbb{R}^{N \times N}$ infeasible.

Thus, usually market practioners will only include selected 2nd order effects leaving a potentially large residual from unexplained 2nd order effects as well as all higher order effects. Another drawback is that each product category then requires special implementation in order to select just *the* most important 2nd order effects.

3 An exact PnL attribution

The traditional approach to PnL Explain described above thus leaves some room for improvement.

Instead, defining $\phi(t) := \mathrm{PV}(x + tv)$ for a scalar $t \in \mathbb{R}$ we may write the change in PV as:

$$\begin{aligned}
\mathrm{PV}(x+v) - \mathrm{PV}(x) &= \phi(1) - \phi(0) && (2)\\
&= \int_0^1 \phi'(s) ds && (3)\\
&= v^T \int_0^1 \nabla_{\mathrm{PV}}(x + sv) ds && (4)\\
&= v^T \widehat{\nabla_{\mathrm{PV}}}(x; v) && (5)
\end{aligned}$$

The above integral, which we denote $\widehat{\nabla_{\mathrm{PV}}}(x; v)$, can be interpreted as the *average gradient* over the course of $x \to x + v$.

Hence, when doing the dot product with v, the result may be viewed as the infinite sum of all 1st order effects encountered as x moves along its path towards $x + v$.

An exact attribution of the net PnL to the input factors is then a matter of calculating $\widehat{\nabla_{\mathrm{PV}}}(x;v)$.

4 The *average* gradient

Assuming sufficient levels of differentiability of PV(x), we can calculate the average gradient defined above efficiently to arbitrary order (here 2nd order) with:

$$\begin{aligned}
\widehat{\nabla_{\mathrm{PV}}}(x;v) &= \int_0^1 \nabla_{\mathrm{PV}}(x+sv)ds && (6)\\
&= \int_0^1 \left[\nabla_{\mathrm{PV}}(x) + s\nabla'_{\mathrm{PV}}(x)v + \frac{1}{2}s^2 v^T \nabla''_{\mathrm{PV}}(x)v + \mathcal{O}(||sv||^3)\right]ds && (7)\\
&= \nabla_{\mathrm{PV}}(x) + \frac{1}{2}\nabla'_{\mathrm{PV}}(x)v + \frac{1}{6}v^T\nabla''_{\mathrm{PV}}(x)v + \mathcal{O}(||v||^3) && (8)\\
&= \kappa(0) + \frac{1}{2}\kappa'(0) + \frac{1}{6}\kappa''(0) + \mathcal{O}(||v||^3) && (9)
\end{aligned}$$

with[1]

$$\begin{aligned}
\kappa(t) &= \nabla_{\mathrm{PV}}(x+tv) && (10)\\
\kappa'(t) &= \nabla'_{\mathrm{PV}}(x+tv)v && (11)\\
\kappa''(t) &= v^T\nabla''_{\mathrm{PV}}(x+tv)v && (12)\\
\kappa'''(t) &= ... && (13)
\end{aligned}$$

$\kappa(t) : \mathbb{R}^1 \to \mathbb{R}^N$ takes only a scalar input and is thus efficiently differentiated using forward-mode automatic differentation. That also applies to κ', κ'' and all higher-orders. Hence, if needed, it would be easy (and feasible!) to include e.g. all 3rd, 4th and 5th order effects.

5 Geometric interpretation

Where PnL Explain has traditionally been conducted by calculating (parts of) the full $N \times N$ Hessian, $H(x) = \nabla'_{\mathrm{PV}}(x)$, this is generally a waste of time (and RAM).

The Hessian (and higher order tensors...) represent the higher order changes to the gradient in *all* directions. However, when decomposing PV changes that full level of generality is not really needed. We are only interested in *one* specific direction, i.e. the direction v, as our input factors move from x to $x+v$.

The 1st order change in the gradient at x in the direction v is thus given by the directional derivative $H(x)v$.

[1]Note, ∇''_{PV} is a 3-dimensional matrix (tensor). Thus, strictly speaking, $v^T\nabla''_{\mathrm{PV}}(x+tv)v$ is a misuse of notation. Nevertheless, all $\kappa^{(n)}(t)$ are vector-valued functions.

The trick to calculate $H(x)v$ using forward-mode automatic differentation of the gradient was first pointed out by Pearlmutter[2] in 1994. Although Pearlmutter does not use the term "forward-mode automatic differentation" - as this was not an established "thing" at the time - this is virtually the algorithm that he describes.

6 Real-world performance

We do a 1-day PnL explain of the PnL at April 11, 2023 for a 1m30y at-the-money payer swaption using the standard normal-vol model. In our setup the pricing function takes $N = 138$ inputs.

Order	$\kappa^{(n)}$	Unexplained residual (relative to PnL)	Cpu-time (μs) (new approach)	Cpu-time (μs) (old approach, theoretical)
1st	κ	16%	16	16
2nd	κ'	0.91%	21	256
3rd	κ''	0.13%	28	4,096
4th	κ'''	0.027%	46	65,536
5th	κ''''	0.00058%	97	1,048,576

Our pricing engine is implemented in Julia[1] with a custom implementation of reverse-mode automatic differentation. The forward-mode is a one-liner using the well-established 3rd party package ForwardDiff.jl[4].

7 Conclusion

Attributing realized PnL to market variables has a mirror problem; that of attributing expected future PnL to expected future changes in market variables. Also known as carry/roll calculations. This may be done using the exact same approach.

But in general, the method presented here is highly efficient algorithnm for attributing the change in a scalar function to the input factors with arbitrary precision. As such, it may also have other use cases outside the domain of finance.

E.g. for neural networks, the loss function is efficiently differentiated using backpropagation (reverse-mode automatic differentation). Presumably, the method in this paper could then be used to attribute a change in the loss function to changes in input factors.

References

[1] Jeff Bezanson, Alan Edelman, Stefan Karpinski, Viral B. Shah, *Julia: A fresh approach to numerical computing.* SIAM Review, Volume 59.

[2] Barak A. Pearlmutter, *Fast Exact Multiplication by the Hessian.* Neural Computation, Volume 6, Issue 1.

[3] Antoine Savine, *Modern Computational Finance: AAD and Parallel Simulations.* Wiley.

[4] Jarrett Revels, Miles Lubin, Theodore Papamarkou, *Forward-Mode Automatic Differentiation in Julia.* https://arxiv.org/abs/1607.07892