Remarks on Automatic Adjoint Differentiation for gradient descent and models calibration

Dmitri Goloubentsev*, Evgeny Lakshtanov†

Abstract

In this work, we discuss the Automatic Adjoint Differentiation (AAD) for functions of the form $G = \frac{1}{2}\sum_1^m (Ey_i - C_i)^2$, which often appear in the calibration of stochastic models. We demonstrate that it allows a perfect SIMD[1] parallelization and provide its relative computational cost. In addition we demonstrate that this theoretical result is in concordance with numeric experiments.

Key words: Automatic Adjoint Differentiation, automatic vectorization, Single instruction multiple data, AAD-Compiler

The Automatic Adjoint Differentiation (AAD) is a rapidly growing field with a wide range of applications including image restoration [5], computer vision [9] and maching learning in general see e.g. [2], [10]. The long list of AAD applications can be found in the site of *Autodiff* community, see [1].

For the additional reading one can mention monograph by Antoine Savine [12], articles by L.Capriotti [4] and by other authors e.g. [13], [14].

The AAD has become a widespread tool in applications due to the following property: If one has an algorithm for a function $f : \mathbb{R}^n_x \to \mathbb{R}^m_y$, then AAD maps each $\lambda \in \mathbb{R}^m$ to the linear combinations of $\partial_i f$

$$\left\{ \lambda \cdot \frac{\partial f(x)}{\partial x_i}, \quad i = 1 \dots n \right\} \tag{1}$$

*MathLogic LTD, London, UK., dmitri@matlogica.com
†CIDMA, Department of Mathematics, University of Aveiro, Aveiro 3810-193, Portugal and MathLogic LTD, London, UK., lakshtanov@matlogica.com

[1]Single Input Multiple Data

the computation cost does not exceed that of f which is usually between 2 and 10 (depending on a specific AAD tool used).

The utilization of parallel computations appears naturally in calculation of expectations since it assumes the numerous independent calculation of an integrand. Suppose that one needs to calculate the $\frac{d}{dx}Ey$, the algorithm for the $\frac{d}{dx}y$ is first determined and then the average using parallel computations. From that point of view, the AD should avoid differentiate expectations. However, in case of the Adjoint AD, this approach becomes mandatory. The point is that the adjoint differentiation algorithm is not local (e.g. [4]) i.e. it requires analysis of the whole algorithm for f, thus the differentiation of expectation requires much more memory than taking expectation of the derivative.

Some important problems involve computations that include an expectation as an intermediate operation. It is not a trivial question how one can use AAD and avoid differentiation of expectations. In a recent article [7], Fries suggest a general recipe for this problem (see Appendix). However, Fries did not provide the accurate analysis for the computational cost of the proposed algorithm. This work conducts a step-by-step computation of the following simple but important example.

Consider a functional of the type

$$G = \frac{1}{2}\sum_{1}^{m}(Ey_i - C_i)^2, \tag{2}$$

where y_i are random variables on a filtered probability space $(\Omega, \mathbb{Q}, \{\mathcal{F}_t\})$ and C_i are some given target conditions, $i = 1, \dots n$. It is assumed that we are provided with a *forward* algorithm

$$F \ : \ \mathbb{R}^{M+N} \to \mathbb{R}^m_y,$$

which calculates y_i for a given set of M parameters and N independent random variables in frames of a time-discretization of the stochastic process.

We assume that the AAD tool provides the algorithm R (so called "reverse" algorithm, e.g. [4]) which calculates the full set of derivatives (1)[2] We also assume that the AAD tool produces parallelized versions F_v and R_v. This means that F_v (or R_v) provides the execution of F (or R) for c

[2]it can be applied only after the F was launched.

independent sets of input data. For example, the natural value of c for the case of an AAD-tool tuned to the Intel AVX512 architecture is

$$c = 8 \times \text{Number\_of\_Cores}. \tag{3}$$

Is noted that the first factor in (3) equals to 1 in all known to us C++ AAD tools.

We consider the operations $F \to F_v$ and $F \to R_v$ as a simple way to parallelize calculations and we are interested to check if there is a way to use this tool effectively.

Correction coefficients K_F and K_R are defined via the following expression

$$\text{Cost}(\text{F}_\text{v}) = \frac{\text{K}_\text{F}}{\text{c}}\text{Cost}(\text{F}), \quad \text{Cost}(\text{R}_\text{v}) = \frac{\text{K}_\text{R}}{\text{c}}\text{Cost}(\text{F}) \tag{4}$$

and they reflect the quality of the AAD tool. Ideally, $K_F = 1$, but in practice, the different software optimization and different hardware specifics can never make it perfect. One reason is that the executable version of F is produced with all compiler's optimization abilities turned On, including the ability SIMD vectorization.

Some remarks on the Monte-Carlo simulations. We approximate expectations by

$$Ey \sim \frac{1}{\text{number\_of\_Paths}} \sum_{k=1}^{\text{number\_of\_Paths}} y(\omega_k),$$

where the set $\{\omega_i\}$ contains Monte-Carlo simulations of a given stochastic process and each ω_i is a simulated sample path of sequences of random variables. It is assumed that drawings are $\mathbb{Q}$-uniform.

In particular,the cost of the independent evaluation for the set $\{Ey_i, i = 1, \ldots, m\}$ is

$$\text{Cost}(\{\text{Ey}\}) = \frac{\text{K}_\text{F}}{\text{c}} \times \text{number\_of\_Paths} \times \text{Cost}(\text{F}). \tag{5}$$

Calculation of the gradient of G (introduced in (2)). For any parameter x_j we get that

$$\frac{\partial G}{\partial x_j} = E\left(\sum_{i=1}^{m}(Ey_i - C_i)\frac{\partial y_i}{\partial x_j}\right), \quad j = 1, \ldots, M.$$

It leads to the following algorithm :

1. Calculate Ey_i using F_v. The costs of this calculation is given by (5).

2. Fix a path ω. Using formula (1) for the vector λ with components $\lambda_i = Ey_i - C_i$ we get the set

$$\left\{ \sum_{i=1}^{m} (Ey_i - C_i) \frac{\partial y_i(\omega)}{\partial x_j}, \quad j = 1 \dots M \right\}$$

   For c paths, it costs $(K_F + K_R)\mathrm{Cost(F)}$ since for each path ω the reverse algorithm can be executed only after the forward algorithm has been launched, unless execution results of the first step can be stored in the memory for each ω. In case memory usage is not constrained, the cost is $K_R\mathrm{Cost(F)}$

3. Summation over paths ω. To calculate the cost of this operation we take into account that the integrand should be calculated number_of_Paths times.

Summarizing, we get the total cost

$$\mathrm{Cost(G)} = \frac{2\mathrm{K_F} + \mathrm{K_R}}{\mathrm{c}} \times \mathrm{number\_of\_Paths} \times \mathrm{Cost(F)}. \tag{6}$$

For C++, AAD-compiler produced by MathLogic LTD can be rewritten as

$$\mathrm{Cost(G)} = \frac{2\mathrm{K_F} + \mathrm{K_R}}{8 \times \mathrm{number\_of\_Cores}} \times \mathrm{number\_of\_Paths} \times \mathrm{Cost(F)}.$$

where 8 reflects that the AVX512 architecture allows 8 doubles per vector register.

Test on the Heston Stochastic Local Volatility model calibration. For financial institutions, the model calibration is a routine day-to-day procedure. Although the Partial Differential Equation (PDE)-based techniques became quite popular e.g. [6], [11], practitioners more often use direct numeric simulations due to the simplicity of the technique.

Consider the Heston SLV model given by a process

$$\begin{cases} dS_t = \mu S_t dt + \sqrt{V_t} L(t, S_t) S_t dW_t^S, \\ dV_t = \kappa(\theta - V_t)dt + \xi\sqrt{V_t} dW_t^V, \\ dW_t^S dW_t^v = \rho dt. \end{cases}$$

Our implementation utilizes the standard Euler discretization (e.g. 3.4 of [8]) and European options in the quality of the $y_i, i =, \ldots, m$. The set of optimization parameters consists values of the piecewise constant Leverage function $\{L(t_i, S_j)\}$ (where the set of interpolation nodes $\{t_i, S_j\}$ is fixed) and five standard Heston model parameters. We applied the AADC[3] by MathLogic LTD and observed the following values for coefficients K_F and K_R defined in (4).

N.Cores=1	K_F/c	K_R/c	$\frac{\text{Cost(G)}}{\text{number\_of\_Paths}\times\text{Cost(F)}}$
AVX2	0.39	0.23	1.01
AVX512	0.23	0.12	0.58

Tests were executed on one core. The values of the aforementioned coefficients became almost constant when number of time intervals and number of optimization instruments m grow. As mentioned, all optimization parameters (like vectorization) were turned on while the evaluation of the Cost(F).

Conclusion. We consider a situation when a straighforward application of the AAD software is not possible i.e. for functions of the form $G = \frac{1}{2}\sum_1^m (Ey_i - C_i)^2$. We propose a two-step algorithm which leads to the estimate (6) and therefore, provides a perfect SIMD parallelization. A numeric implementation of the algorithm is realized for the Heston model using the AAD-Compiler by matlogica.com. Presented results are in good concordace with (6).

Appendix. Expected Backward Automatic Differentiation Algorithm following [7]. Consider a scalar function y given by a sequence of operations:

$$y := x_N \tag{7}$$

$$x_m := f_m(x_{\tau_m 1}, \ldots, x_{\tau_m i_m}), \quad 1 \le m < N. \tag{8}$$

where the number of variables i_m is either 0 and it means that the x_m is an independent variable or $1 \le i_m < m$. Evidently for function τ_m we have $1 \le \tau_m < m$. We assume that $k-$th operator is an expectation operator and others $f_m, m \ne k$ given by a closed-form[4].

Sequentially,

[3] C++ AAD-tool produced by matlogica.com

[4] i.e. it can be evaluated in a finite number of the "well-known" operations, see e.g. wikipedia https://en.wikipedia.org/wiki/Closed-form_expression for the detailed definition.

- Initialise $\overline{D}_N = 1$ and $\overline{D}_m = 0,\ m < N$.
- For all $m = N, N-1, ..., 1$ (iterating backward through the operator list)
  - for all $j = \tau_m 1, ..., \tau_m i_m$ (iterating through the argument list)

$$\overline{D}_j = \begin{cases} \overline{D}_j + \overline{D}_m \frac{\partial f_m}{\partial x_j}, & m \neq k, \\ \overline{D}_j + E\overline{D}_m, & m = k. \end{cases}$$

Then, for all $1 \leq i \leq N$

$$E\frac{\partial y}{\partial x_i} = E\overline{D}_i.$$

Acknowledgments. E.L. was partially supported by Portuguese funds through the CIDMA - Center for Research and Development in Mathematics and Applications and the Portuguese Foundation for Science and Technology ("FCT–Fundção para a Ciência e a Tecnologia"), within project UID/MAT/0416/2019.

References

[1] `http://www.autodiff.org/?module=Applications`

[2] Baydin, Atilim Gunes, et al. (2018), Automatic differentiation in machine learning: a survey. Journal of Marchine Learning Research, 18, 1-43.

[3] Bartholomew-Biggs, M., Brown, S., Christianson, B., Dixon, L. (2000). Automatic differentiation of algorithms. Journal of Computational and Applied Mathematics, 124(1-2), 171-190.

[4] Capriotti, L. (2011). Fast Greeks by algorithmic differentiation. The Journal of Computational Finance, 14(3), 3.

[5] Goossens, B., Luong, H., Philips, W. (2017, September). GASPACHO: a generic automatic solver using proximal algorithms for convex huge optimization problems. In Wavelets and Sparsity XVII (Vol. 10394, p. 1039410). International Society for Optics and Photonics.

[6] Crépey, S. (2003). Calibration of the local volatility in a generalized black–scholes model using tikhonov regularization. SIAM Journal on Mathematical Analysis, 34(5), 1183-1206.

[7] Christian P. Fries. Stochastic Automatic Differentiation: Automatic Differentiation for Monte-Carlo Simulations. In: SSRN (2017). DOI: 10.2139/ ssrn.2995695.

[8] Glasserman, P. (2013). Monte Carlo methods in financial engineering (Vol. 53). Springer Science & Business Media.

[9] Pock, T., Pock, M., Bischof, H. (2007). Algorithmic differentiation: Application to variational problems in computer vision. IEEE Transactions on Pattern Analysis and Machine Intelligence, 29(7), 1180-1193.

[10] Srajer, F., Kukelova, Z., Fitzgibbon, A. (2018). A benchmark of selected algorithmic differentiation tools on some problems in computer vision and machine learning. Optimization Methods and Software, 1-14.

[11] Saporito, Y. F., Yang, X., Zubelli, J. P. (2017). The Calibration of Stochastic-Local Volatility Models-An Inverse Problem Perspective. arXiv preprint arXiv:1711.03023.

[12] Savine, A. (2018). Modern Computational Finance: AAD and Parallel Simulations. John Wiley & Sons.

[13] Hans-Jørgen F., Huge B., Savine A. Practical implementation of aad for derivatives risk management, xva and rwa. Global Derivatives (2015).

[14] Savine A., Computation Graphs for AAD and Machine Learning Part I: Introduction to Computation Graphs and Automatic Differentiation. Wilmott, 2019, 2019.104: 36-61.