Efficient Computation of Hessian Matrices in a Monte Carlo Setting using Automatic Differentiation

by

Xixuan Yu

A research paper
presented to the University of Waterloo
in partial fulfillment of the
requirement for the degree of
Master of Mathematics
in
Computational Mathematics

Supervisor: Prof.Thomas.F.Coleman

Waterloo, Ontario, Canada, 2016

Abstract

The evaluation of differentiable functions often requires computation of the first and second derivatives. Automatic differentiation is a good method to compute the derivatives, but it can be costly to compute the Hessian when involving large number of variables. In this paper, we develop a new structured AD technique which is applicable when the objective function is in structured form, such as in a Monte Carlo setting. Both the theory and experimental results show that our new structured AD has a very high efficiency in computational space utilization for the Hessian evaluation.

Acknowledgements

I would like to appreciate my supervisor Prof.Thomas.F.Coleman, for his patient guidance and generous support on my research paper. I would like to thank all my classmates and professors in University of Waterloo. I had a wonderful master experience in Computational Math program.

Dedication

This is dedicated to the one I love.

Table of Contents

List of Tables **viii**

List of Figures **ix**

1 Introduction **1**

1.1 Hessian Background 1

1.2 Automatic Differentiation Advantage 2

1.3 Implement AD in Monte Carlo Settings 3

1.4 Paper Overview 4

2 Hessian Computation Path by Path **5**

2.1 General Structured Function 5

2.2 Structured Function in a Monte Carlo setting 7

2.3 Structured AD for the Hessian Computation by MC paths 10

3 The Hessian Computation by Monte Carlo Segment **14**

3.1 Compute the Hessian segment by segment 14

3.2 Final Hessian evaluation of all paths 19

4 Experimental Results **22**

4.1 A Financial application 22

4.2 Comparison results 24

4.3 Experimental results for only increasing Asset number 25
4.4 Experimental results for only increasing MC-Path number 26
4.5 Experimental results for only increasing Segments number 30

5 Conclusion **31**

References **33**

List of Tables

4.1 Initial parameters of x in Heston Model 25
4.2 Only increasing Asset number 26
4.3 Only increasing MC-Path number 27
4.4 Only increasing segments number 30

List of Figures

2.1 Principle of path-wise computation. 8

3.1 The computational graph of composite function 15

4.1 Tape Comparison Asset=40, Segments=50 28
4.2 Time Comparison Asset=40, Segments=50 28
4.3 Tape Ratio Asset=40, Segments=50 29
4.4 Time Ratio Asset=40, Segments=50 29

Chapter 1

Introduction

Derivative calculation is an important part of many scientific applications, especially in financial engineering [3]. In quantitative finance, derivatives are used to measure the sensitivity of financial instruments, such as options and futures. Sensitivity can effectively reflect the reaction of financial instruments to the fluctuations of underlying factors, therefore helping in a hedging strategy. Thus, how to efficiently calculate derivatives is a critical part in quantitative finance.

Evaluating first and second derivatives are the most two common cases in derivative calculation. In particular, the repeated evaluation of the gradient is extremely popular in finance, since the first derivative is necessary for most hedging techniques. Nevertheless, the second derivative also plays an important role when hedging complex financial derivatives. For example, in gamma hedging, the second derivative best explains the local profit and loss movements [3]. In addition, many constrained optimization methods need to use second derivatives information to solve real world problems. Thus, it is often desirable to compute, second derivatives.

1.1 Hessian Background

The Hessian of a function f: $R^n \to R$ is its n by n symmetric matrix of second derivatives. The local behavior of a twice continuously differentiable function at any point of f can be accurately described by its gradient and Hessian[9] (evaluated at the current point x). The Hessian is the key to many non-linear optimization algorithms. However, there are two main issues for Hessian computation: how to twice differentiate and code the second derivative of functions, and how to determine the cost of the corresponding calculation (at each iterate x). Computing all second derivatives consumes a great deal of time and CPU memory, which makes the Hessian calculation become much more expensive than the relatively simple gradient computation.

Although many numerical methods are developed to compute or approximate the Hessian matrix, each method has its drawbacks. For example, one popular method, finite differences, can yield poor accuracy in computing the second derivatives. In addition, computing all of the second derivatives via finite differencing may require a prohibitive number of function evaluations, especially for objective functions with high dimensions. The advent of automatic differentiation [2] can offer a significant improvement on these two issues.

1.2 Automatic Differentiation Advantage

Automatic Differentiation (AD) is a chain-rule-based [8, 9, 12, 6] technology for computing derivatives, with respect to the input variables. In calculus, the chain rule is a rule for computing derivatives based on functions composition. In particular, the basic idea of AD is that all computer programs, regardless of complexity, can be defined by a relatively small number of elementary operations in programming language. The output values or functions in computer programs are simple combinations of these elementary operations. Since all partial derivatives of elementary operations are known, the overall derivatives can be obtained by using the chain rule.

Automatic Differentiation has two basic modes of operations: the forward mode and the reverse mode. These two modes differ in the order of chain rule execution. In forward mode, the evaluation of derivatives is propagated directly using the chain rule. In the reverse mode, AD computes derivatives in the reverse order: evaluating objective functions first, saving all intermediate variables afterwards. The forward mode operation has computational advantages when the input number is small; the reverse mode is advantageous for the small number of outputs with a large number of inputs. Reverse mode AD [10] can be much faster in the cases where there is a small number of output variables, but may require more computer space to save intermediate variables for the whole calculation.

Xu, Chen and Coleman [13] have developed a new structured AD view to optimize the calculation of AD when functions are structured. They believe that most functions used in modeling are structured. If a function can be presented in a structured form, AD can be tailored to that form, and the hidden sparsity [11] in the corresponding Hessian matrix can be exploited. Thus, the new structured (sparse) AD technique can calculate the Hessian matrix much more efficiently than a straightforward AD approach.

In contrast to the traditional finite differencing method, structured automatic differentiation [1] has two main advantages in computation: accuracy and efficiency. In addition, structured automatic differentiation method can be more applicable in many applications, since it can be flexibly applied to any analytical or numerical method (such as Monte Carlo simulation) without restrictions and make impressive efficiency gains. The structured AD technique perseveres the numerical properties of the original methods, such as stability, convergence and accuracy.

1.3 Implement AD in Monte Carlo Settings

Monte Carlo simulation [5, 4] approximates the value of stochastic functions based on the expectation of its generated random objects. Derivatives calculation via Monte Carlo simulation involves generating a large number of stochastic paths with different random inputs. In particular, each generated path can be broken into a finite set of segments or timesteps. Rather than simply drawing points randomly from space $[0,1]$ or $[0,1]^d$, Monte Carlo process explores particular paths from generated samples in the objective space. Thus, the Monte Carlo long-run average value will be equal to the true value when the number of path is sufficient large. Since the simulation precision is determined by the number of paths, the more paths, the less variability and the higher accuracy. Therefore, Monte Carlo simulation is a suitable and applicable numerical simulation method for derivatives calculation.

In order to guarantee the precision of simulation results, generating a large number of paths is necessary, but limits the feasibility of traditional methods [14] to compute derivatives. However, the structured AD technique can easily compute derivatives in Monte Carlo settings with a relatively small number of evaluations. The structured AD method [13, 14] relies on the fact that each sub-function in the structured form system can be generated by a corresponding Monte Carlo path. When the evaluated function is presented in such structured form, each sub-function in that form has similar structure and can be composited by similar elementary operations, thereby leading a more efficient computation. Thus, the combination of structured AD technique and Monte Carlo simulation can perfectly exhibit advantages and offset drawbacks in derivatives computation.

1.4 Paper Overview

In this paper, we focus on how to efficiently compute Hessian matrices via a structured AD method in a Monte Carlo setting. In chapter 2, based on the general structured form background, we design a new structured form for evaluated function $f(x)$ in a Monte Carlo setting, and develop two algorithms for computing the Hessian by Monte Carlo path. Afterwards, we theoretically compare the computational time and space usage between structured AD and straightforward AD.

In chapter 3, we consider the evaluated function as a composite function, which is highly recursive in the Monte Carlo process. In this case, each Monte Carlo path can be divided into T segments (or Timesteps), and the first and second derivatives can be computed segment by segment. We design another structured form for the composite function and develop two new algorithms for computing the Hessian with respect to segments. Then discuss the improvement of computational time and space utilization of structured AD, comparing to the straightforward AD. Numerical experiments of the Hessian computation are executed in chapter 4. Experimental results reveal a significant difference between the two AD methods: structured AD completes the Hessian computation in a very high efficient space utilization, saving much more computer space than straightforward AD, especially for the large number of paths. Finally, we make a conclusion in Chapter 5.

Chapter 2

Hessian Computation Path by Path

Many objective functions in practical applications have the structured form as (2.1) [13, 14]. If the objective function can be presented in this explicit structured form, efficient and accurate methods can be employed to evaluate the function, thereby reducing the difficulty of derivatives calculation, such as the gradient and the Hessian calculation. We begin at reviewing the general structured form of a function, and later showing how this form improves the computational efficiency.

2.1 General Structured Function

We define a general structured function as follows. Assume to evaluate the scalar-valued function, $f(x) : R^n \to R$, the computation is expressed:

$$\begin{cases} \text{solve for } y_1 : & F_1(x) - M_1 y_1 = 0 \\ \text{solve for } y_2 : & F_2(x, y_1) - M_2 y_2 = 0 \\ \text{solve for } y_3 : & F_3(x, y_1, y_2) - M_3 y_3 = 0 \\ \vdots & \vdots \\ \text{solve for } y_p : & F_p(x, y_1, y_2, \ldots, y_{p-1}) - M_p y_p = 0 \\ \text{solve for } z : & \bar{f}(x, y_1, y_2, \ldots, y_p) - z = 0 \end{cases} \tag{2.1}$$

where $F_i^{'} s(i = 1 : p)$and $\bar{f}$ are intermediate functions, vectors $y_i(i = 1 : p)$ are intermediate variables of various dimensions. The matrix M_i is non-singular. In many applications, M_i is a simple diagonal matrix, often the identity matrix.

Note that the evaluation of any deterministic continuous function f can, in principle, be presented as following form (2.1). We assume that each F_i is a continuously differentiable vector function. The Jacobian matrix of (2.1) is:

$$J^E = \left[\begin{array}{cccccc} J_x^1 & -M_1 & & & & \\ J_x^2 & J_{y_1}^2 & -M_2 & & & \\ J_x^3 & J_{y_1}^3 & J_{y_2}^3 & -M_3 & & \\ \vdots & \vdots & \vdots & & \ddots & \\ J_x^p & J_{y_1}^p & J_{y_2}^p & & J_{y_{p-1}}^p & -M_p \\ 0 & \nabla \bar{f}_{y_1}^T & \nabla \bar{f}_{y_2}^T & \nabla \bar{f}_{y_3}^T & \dots & \nabla \bar{f}_{y_p}^T \end{array}\right]. \tag{2.2}$$

Note that the Jacobian matrix of F_i, *i.e.*, $(J_x^i, J_{y_1}^i, J_{y_2}^i, \dots, J_{y_{i-1}}^i, M_i)$ is often sparse.

We partition J^E into four parts:

$$J^E = \left[\begin{array}{c|ccccc} J_x^1 & -M_1 & & & & \\ J_x^2 & J_{y_1}^2 & -M_2 & & & \\ J_x^3 & J_{y_1}^3 & J_{y_2}^3 & -M_3 & & \\ \vdots & \vdots & \vdots & & \ddots & \\ J_x^p & J_{y_1}^p & J_{y_2}^p & & J_{y_{p-1}}^p & -M_p \\ \hline 0 & \nabla f_{y_1}^T & \nabla f_{y_2}^T & \nabla f_{y_3}^T & \dots & \nabla f_{y_p}^T \end{array}\right] = \left[\begin{array}{c|c} \hat{J}_x^E & \hat{J}_y^E \\ \hline \nabla f_x^T & \nabla f_y^T \end{array}\right]. \tag{2.3}$$

Then the gradient of f can be expressed [13]:

$$\nabla f_x^T = \nabla \bar{f}_x^T - \nabla \bar{f}_y^T (\hat{J}_y^E)^{-1} \hat{J}_x^E. \tag{2.4}$$

To develop the structured Hessian view, we define function g :

$$g(x, y, \omega) = \bar{f}(x, y) + \sum_{i=1}^{p} \omega_i^T F_i(x, y), \tag{2.5}$$

where vector variable ω is the solution of $(J_y^E)^T \omega = -\nabla \bar{f}_y$, $\omega^T = (\omega_1^T, \omega_2^T, \dots, \omega_p^T)$ corresponds to vector $y^T = (y_1^T, y_2^T, \dots, y_p^T)$.

Next, we define the new 'auxiliary Hessian matrix' H^a by differentiating (2.5) with respect to three variables (x, y, ω) :

$$H^a = \left[\begin{array}{ccc} \hat{J}_x^E & \hat{J}_y^E & 0 \\ \nabla_{yx}^2 g & \nabla_{yy}^2 g & (\hat{J}_y^E)^T \\ \nabla_{xx}^2 g & \nabla_{xy}^2 g & (\hat{J}_x^E)^T \end{array}\right]. \tag{2.6}$$

We assume H^a is sparse; the Hessian of (2.5), $\nabla^2_{xx} g(x, y, \omega)$ is evaluated by applying sparse AD to $g(x, y, \omega)$.

Similarly, we partition H^a into four parts:

$$H^a = \left[\begin{array}{c|cc} \hat{J}^E_x & \hat{J}^E_y & 0 \\ \nabla^2_{yx} g & \nabla^2_{yy} g & (\hat{J}^E_y)^T \\ \hline \nabla^2_{xx} g & \nabla^2_{xy} g & (\hat{J}^E_x)^T \end{array}\right] = \left[\begin{array}{c|c} A & L \\ \hline B & M \end{array}\right]. \tag{2.7}$$

Based on Schur complement computation, the Hessian of f related to H^a is [13]:

$$H = B - ML^{-1}A. \tag{2.8}$$

2.2 Structured Function in a Monte Carlo setting

Monte Carlo simulation is broadly used in the financial field for approximating values of evaluated functions. In this chapter, we are concerned with how to apply the structured AD method in a Monte Carlo setting to perform a highly efficient evaluation of Hessian matrices via each Monte Carlo path.

We assume that function $U(x, Z)$ contains initial parameters x and random innovations Z [13]. The expectation of this function is:

$$E(U(x, Z)) = \int U(x, Z)\rho(Z)\delta Z, \tag{2.9}$$

where $\rho(Z)$ is the probability density function of Z. $\rho(Z)$is not influenced by the set of initial parameters x. In our assumption, function $U(x, Z)$ satisfies the certain regularity conditions, and integration and differentiation order can be interchanged as:

$$\frac{\partial E(\cdot)}{\partial x} = \frac{\partial}{\partial x}\int U(x, Z)\rho(Z)dZ = \int \frac{\partial U(x, Z)}{\partial x}\rho(Z)dZ. \tag{2.10}$$

Another expression of formula (2.10) is:

$$\frac{\partial E(U(x, Z))}{\partial x} = E\left(\frac{\partial U(x, Z)}{\partial x}\right). \tag{2.11}$$

It is usually difficult to calculate the expectation $E(x)$, but it is relatively easier to estimate the integral value in the Monte Carlo simulation. Note that the random innovation

Z can be broken down to p discrete vectors Z_i, and each vector corresponds to a simulation path. The new expression of $E(x)$ in the Monte Carlo setting is:

$$\widehat{E(x)} = \frac{1}{p}\sum_{i=1}^{p} U(x, Z_i) = \frac{1}{p}\sum_{i=1}^{p} \widehat{U}_i, \tag{2.12}$$

where $\widehat{U}_i$ represents simulation paths. Since paths are independent with each other in Monte Carlo simulation, $\widehat{U}_i$ can be estimated path by path (named path-wise). The principle of path-wise computation is illustrated in Figure 2.1.

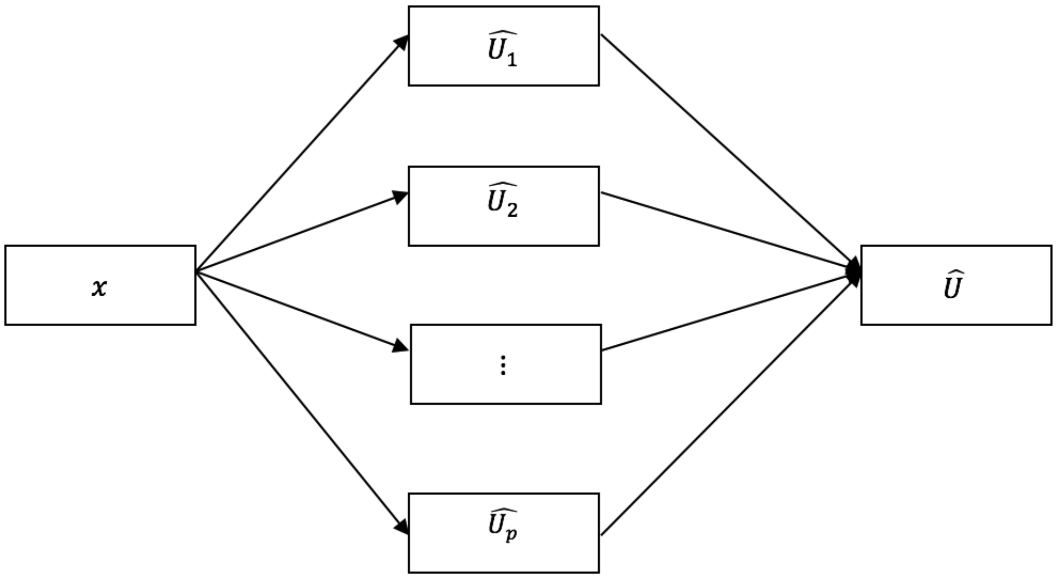

Figure 2.1: Principle of path-wise computation.

Based on structure (2.9), we have a corresponding relationship:

$$E(\widehat{U(x, Z)}) \equiv \bar{f}. \tag{2.13}$$

Each $y_i = F_i(x, y_1, y_2, \ldots, y_{i-1})$ is consistent with path $\hat{U}_i$ in $y_i = \hat{U}_i = \widehat{U(x, Z_i)}$. Note in a Monte Carlo setting, y_i has no dependence on vector y_k, any $k < i, k, i = 1 : p$. In this situation, we can simplify the structure as:

$$\begin{cases} F_1(x) - y_1 = 0 \\ F_2(x) - y_2 = 0 \\ F_3(x) - y_3 = 0 \\ \vdots \\ F_p(x) - y_p = 0 \\ \bar{f}(x, y_1, y_2, \ldots, y_p) - z = 0 \end{cases} \tag{2.14}$$

where $F_i(x) = \hat{U}(x, Z_i), i = 1, \ldots, p$.This specific form (2.14) is known as Generalized Partially Separable Function, and is also the structured function for the path-wise computation of Hessian matrices. $F_i(x)$ is independent with each other, and only depends on the initial control parameters x. Then we can obtain $\bar{f}(x, y_1, y_2, \ldots, y_p) = \frac{1}{p}\sum_{i=1}^{p} \hat{U}(x, Z_i)$. A simplified corresponding extended Jacobian matrix is:

$$J_1^E = \left[\begin{array}{c|ccccc} J_x^1 & -I & & & & \\ J_x^2 & & -I & & & \\ J_x^3 & & & -I & & \\ \vdots & & & & \ddots & \\ J_x^p & & & & & -I \\ \hline 0 & \nabla\bar{f}_{y_1}^T & \nabla\bar{f}_{y_2}^T & \nabla\bar{f}_{y_3}^T & \ldots & \nabla\bar{f}_{y_p}^T \end{array}\right] = \left[\begin{array}{c|c} \hat{J}_x^E & \hat{J}_y^E \\ \hline \nabla\bar{f}_x^T & \nabla\bar{f}_y^T \end{array}\right], \tag{2.15}$$

where the gradient satisfies:

$$\nabla f_x^T = 0 - \nabla\bar{f}_y^T \cdot I \cdot \hat{J}_x^E = \sum_{i=1}^{p} \nabla\bar{f}_{y_i}^T \hat{J}_x^i. \tag{2.16}$$

In order to evaluate the Structured Gradient, Algorithm 1 is developed:

Algorithm 1: *Implicit_Structured_Gradient_Compute (evaluation of $\nabla\bar{f}_x^T$)*

1. Evaluate $z = \bar{f}(x, y_1, y_2, \cdots, y_p)$ and apply reverse-mode AD to $\bar{f}$ to obtain $\nabla\bar{f}^T = (\nabla_x\bar{f}^T, \nabla_{y_1}\bar{f}^T, \nabla_{y_2}\bar{f}^T, \cdots, \nabla_{y_p}\bar{f}^T)$.

2. (a) Evaluate $y_i = F_i(x)$, $i = 1, ..., p$.

(b) Evaluate and differentiate $\bar{f} = (y_1, y_2, \cdots, y_p)$ using reverse-mode AD to get $\omega_i = \nabla \bar{f}_{y_i}$, $i = 1, ..., p$.

(c) Compute by reverse-mode AD: $v_i^T = \omega_i^T \cdot J_x^i$, where J_x^i is the Jacobian of $F_i(x)$, $i = 1, ..., p$.

(d) Set $\nabla f(x) \leftarrow \sum_{i=1}^p v_i$.

In Algorithm 1, for structured AD, time is proportional to $\omega(f)$, and space usage σ satisfies $\sigma \sim \max\{\omega(\bar{f}), \omega(F_i(x)), i = 1, \ldots, p\}$. Although the time complexity of both AD methods is the same, memory requirement of structured AD becomes much less. Compared to the space utilization of straightforward AD: $\sigma \sim \omega(f(x))$, structured AD only requires $\sigma \sim \max\{\omega(\bar{f}), \omega(F_i(x)), i = 1, \ldots, p\}$. Thus, the larger p, the less memory increase in structured AD algorithm. If p is very large, $\omega(f) \gg \max\{\omega(\bar{f}), \omega(F_i(x)), i = 1, \ldots, p\}$. The structured AD becomes much more efficient in space utilization [13].

2.3 Structured AD for the Hessian Computation by MC paths

For the Hessian matrix calculation, we define:

$$h(x, y, \omega) = \bar{f}(x, y) + \sum_{i=1}^{p} \omega_i^T F_i(x, y). \tag{2.17}$$

In a particular Monte Carlo setting, (2.17) can be simplified as:

$$\begin{aligned} h(x, y, \omega) &= \bar{f}(x, y) + \sum_{i=1}^{p} \omega_i^T U(x, Z_i) \\ &= \bar{f}(x, y) + \sum_{i=1}^{p} \omega_i^T \hat{U}_i \\ &= \frac{1}{p} \sum_{i=1}^{p} \hat{U}_k + \sum_{i=1}^{p} \omega_i^T F_i(x, y) \end{aligned} \tag{2.18}$$

where vector variable ω satisfies $(\hat{J}_Y^E)^T\omega = -\nabla_y \bar{f}$. In particular, $(\hat{J}_Y^E)^T$ is made up of identity matrices:

$$(\hat{J}_Y^E)^T = \begin{bmatrix} -I & & & & \\ & -I & & & \\ & & -I & & \\ & & & \ddots & \\ & & & & -I \end{bmatrix}. \tag{2.19}$$

Thus, ω can be written as:

$$\omega^T = \nabla_Y \bar{f}^T = \begin{pmatrix} \nabla \bar{f}_{y_1}^T & \nabla \bar{f}_{y_2}^T & \nabla \bar{f}_{y_3}^T & \cdots & \nabla \bar{f}_{y_p}^T \end{pmatrix}. \tag{2.20}$$

Then the specific 'auxiliary Hessian matrix' H_1^a in a Monte Carlo setting is:

$$H_1^a = \left[\begin{array}{c|cc} \hat{J}_X^E & \hat{J}_Y^E & 0 \\ \nabla_{yx}^2 h & \nabla_{yy}^2 h & (\hat{J}_Y^E)^T \\ \hline \nabla_{xx}^2 h & \nabla_{xy}^2 h & (\hat{J}_X^E)^T \end{array}\right] = \left[\begin{array}{c|c} A_1 & L_1 \\ \hline B_1 & M_1 \end{array}\right]. \tag{2.21}$$

In order to obtain $H_1 = B_1 - M_1 L_1^{-1} A_1$, we decompose L_1 as [14]:

$$L_1 = \begin{pmatrix} \hat{J}_Y^E & 0 \\ \nabla_{yy}^2 h & (\hat{J}_Y^E)^T \end{pmatrix} = \begin{pmatrix} I & 0 \\ \nabla_{yy}^2 h (\hat{J}_Y^E)^{-1} & (\hat{J}_Y^E)^T \end{pmatrix} \begin{pmatrix} \hat{J}_Y^E & 0 \\ 0 & I \end{pmatrix}$$

where both decomposed matrices are non-singular. Thus, L_1 has inverse matrix [14]:

$$L_1^{-1} = \begin{pmatrix} \hat{J}_Y^E & 0 \\ 0 & I \end{pmatrix}^{-1} \begin{pmatrix} I & 0 \\ \nabla_{yy}^2 h (\hat{J}_Y^E)^{-1} & (\hat{J}_Y^E)^T \end{pmatrix}^{-1},$$

and

$$M_1 L_1^{-1} A_1 = M_1 \begin{pmatrix} \hat{J}_Y^E & 0 \\ 0 & I \end{pmatrix}^{-1} \begin{pmatrix} I & 0 \\ \nabla_{yy}^2 h (\hat{J}_Y^E)^{-1} & (\hat{J}_Y^E)^T \end{pmatrix}^{-1} A \tag{2.22}$$

Applying (2.22), H_1 becomes:

$$H_1 = \nabla_{xx}^2 h + \nabla_{xy}^2 h \cdot y_x + y_x^T \nabla_{yx}^2 h + y_x^T \nabla_{yy}^2 h \cdot y_x, \tag{2.23}$$

where intermediate variables $y_x = -(\hat{J}_y^E)^{-1}\hat{J}_x^E = \hat{J}_x^E$.

In particular, the final path-wise Hessian computation formula in a Monte Carlo setting is:

$$H_{MC}^{Path} = \nabla_{xx}^2 h + \nabla_{xy}^2 h \cdot (\hat{J}_x^E) + (\hat{J}_x^E)^T \nabla_{yx}^2 h + (\hat{J}_x^E)^T \nabla_{yy}^2 h \cdot (\hat{J}_x^E), \tag{2.24}$$

where $\nabla^2_{xx}h, \nabla^2_{xy}h, \nabla^2_{yy}h$ can be computed by the structured reverse-mode AD.

Algorithm2 is developed for evaluating (2.24) to obtain the Hessian matrix.

Algorithm 2: *Algorithm Explicit-Structured-Hessian MC Compute*

1. Compute ω_i and y_i by using Algorithm 1;
2. Set $\nabla^2_{xx}h = \nabla^2_{xx}\bar{f}, \nabla^2_{xy}h = \nabla^2_{xy}\bar{f}, \nabla^2_{yy}h = \nabla^2_{yy}\bar{f}$ by applying sparse AD on $\bar{f}$;
   Set $y_x = 0$;
3. For $i = 1, 2, 3, ..., p$, update partial derivatives by applying sparse AD on functions $(\omega_i^T \cdot \hat{U}_i)$

$$\nabla^2_{xx}h = \nabla^2_{xx}h + \nabla^2_{xx}\left(\sum_{i=1}^{p} \omega_i^T \cdot \hat{U}_i\right)$$

$$\nabla^2_{xy}h = \nabla^2_{xy}h + \nabla^2_{xy}\left(\sum_{i=1}^{p} \omega_i^T \cdot \hat{U}_i\right)$$

$$\nabla^2_{yy}h = \nabla^2_{yy}h + \nabla^2_{yy}\left(\sum_{i=1}^{p} \omega_i^T \cdot \hat{U}_i\right)$$

$$y_x = \hat{J}^E_x;$$

4. Matrix multiplication: $C' = (\nabla^2_{xy}h) \cdot y_x, R' = y_x^T \cdot \nabla^2_{yy}h \cdot y_x$;
5. Finally: $H_{MC} = \nabla^2_{xx}h + C' + C'^T + R$.

To compute the sparse Hessian matrix, time and space complexity are $\omega \sim \chi(\nabla^2 f) \cdot \omega(f)$ and $\sigma \sim \omega(f)$ [14]. In a Monte Carlo setting, we can obtain ω_k from $\nabla \bar{f}_y^T$ directly, and $\hat{J}^E_y$ is the identity matrices. The total time and space usage in Algorithm2 decreases to

$$\omega \sim \sum_{i=1}^{p} \left[\chi(J_x^i) + \sum_{j=1}^{i} \chi\left(1 + \chi\left(\nabla^2(\omega_i^T F_i)\right)\right) \right] \cdot \omega(F_i) + \chi(\nabla^2 \bar{f}) \cdot \omega(\bar{f})$$

and

$$\sigma_p \sim |\nabla^2 h|_{nnz} + \max_i \{\omega(F_i), \omega(\bar{f})\}.$$

It is hard to say whether the structured reverse-mode AD costs less time than the straightforward AD, since it depends on the evaluated functions. However, the space requirement of structured AD has a significant decrease. The space requirement of straightforward AD is:

$$\sigma_{P-unsAD} \sim |\nabla^2 h|_{nnz} + \{\sum_{i=1}^{p} \omega(F_i) + \omega(\bar{f})\},$$

where $\{\sum_{i=1}^{p} \omega(F_i) + \omega(\bar{f})\}] \gg \max_i\{\omega(F_i), \omega(\bar{f})\}$(structured AD space usage), especially for large p. Since in straightforward reverse-mode AD, saving all intermediate variables is unavoidable, which consumes massive CPU memory. Instead of saving the whole computational graph, structured AD only saves useful information, which dramatically decreases the memory usage. Therefore, the structured AD technique is much more efficient in space utilization.

Chapter 3

The Hessian Computation by Monte Carlo Segment

In chapter 2, we developed the Monte Carlo path-wise computation of Hessian matrices for the structured function. In this chapter, we expose to a deeper computational level, by noting that each Monte Carlo path represents a composite function.

3.1 Compute the Hessian segment by segment

We consider the objective function f as a highly recursive function, and each sub-function as (2.1) of it is a composite function. Recall Figure 2.1, each Monte Carlo simulation path U_i represents a composite function, and has formula:

$$\hat{U}_i = z^i = \bar{f}^i\left(\tilde{F}_T\left(\tilde{F}_{T-1}\left(\ldots \tilde{F}(x^i)\ldots\right)\right)\right), \tag{3.1}$$

where $\tilde{F}_j, j = 1 : T$ represents the highly recursive vector-valued function in path i, and $\bar{f}^i$ is a continuously differentiable scalar-valued function.

The path computation (3.1) can be presented in the structured form:

$$\begin{cases} \text{solve for } \tilde{y}_1 : & \tilde{F}_1(x^i) - \tilde{y}_1 = 0 \\ \text{solve for } \tilde{y}_2 : & \tilde{F}_2(\tilde{y}_1) - \tilde{y}_2 = 0 \\ \text{solve for } \tilde{y}_3 : & \tilde{F}_3(\tilde{y}_2) - \tilde{y}_3 = 0 \\ \vdots & \vdots \\ \text{solve for } \tilde{y}_T : & \tilde{F}_T(\tilde{y}_{T-1}) - \tilde{y}_T = 0 \\ \text{solve for } z^i : & \bar{f}(x^i, \tilde{y}_1, \tilde{y}_2, \ldots, \tilde{y}_T) - z^i = 0 \end{cases} \tag{3.2}$$

where $\bar{f}^i$ is a scalar-valued function; $\tilde{F}_j$ is a vector-valued function, and both of them can be continuously differentiated.

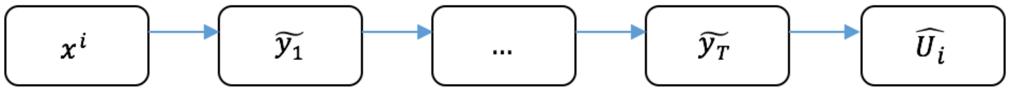

Figure 3.1: The computational graph of composite function

Based on the structured form (3.2), the corresponding extended Jacobian matrix of f^i is:

$$J_i^E = \left[\begin{array}{c|ccccc} \frac{\partial \tilde{F}_1(x^i)}{\partial x^i} & -1 & & & & \\ 0 & \frac{\partial \tilde{F}_2(\tilde{y}_1)}{\partial \tilde{y}_1} & -1 & & & \\ 0 & & \frac{\partial \tilde{F}_3(\tilde{y}_2)}{\partial \tilde{y}_2} & -1 & & \\ \vdots & & & \ddots & \ddots & \\ 0 & & & & \frac{\partial \tilde{F}_T(\tilde{y}_{T-1})}{\partial \tilde{y}_{T-1}} & -1 \\ \hline \nabla f_{x^i}^T & \nabla f_{\tilde{y}_1}^T & \nabla f_{\tilde{y}_2}^T & \nabla f_{\tilde{y}_3}^T & \cdots & \nabla f_{\tilde{y}_T}^T \end{array}\right] = \left[\begin{array}{c|c} \hat{J}_{x^i}^E & \hat{J}_{\tilde{y}}^E \\ \hline \nabla f_{x^i}^T & \nabla f_{\tilde{y}}^T \end{array}\right] \tag{3.3}$$

The gradient of $\bar{f}^i$ can be presented as:

$$\nabla f_{x^i}^T = \nabla \bar{f}_{x^i}^T - \nabla \bar{f}_{\tilde{y}}^T \cdot (\hat{J}_{\tilde{y}}^E)^{-1} \cdot (\hat{J}_{x_i}^E) \tag{3.4}$$

where $\hat{J}_{x_i}^E$ is the sparse matrices and $\nabla f_{x^i}^T = 0$.

Next, we define a vector variable $\omega_i^T = \left(\omega_{i1}^T, \ldots, \omega_{iT}^T\right)$ to satisfy $\omega_{ij}^T = \nabla f_{\tilde{y}}^T \cdot (\hat{J}_{\tilde{y}}^E)^{-1}, j =$

$1 : T$ in a transposed form:

$$\begin{bmatrix} -1 & \frac{\partial \tilde{F}_1(\tilde{y}_1)}{\partial \tilde{y}_1} & & & & \\ & -1 & \frac{\partial \tilde{F}_2(\tilde{y}_2)}{\partial \tilde{y}_2} & & & \\ & & -1 & \frac{\partial \tilde{F}_3(\tilde{y}_3)}{\partial \tilde{y}_3} & & \\ & & & \ddots & \ddots & \\ & & & & -1 & \frac{\partial \tilde{F}_T(\tilde{y}_{T-1})}{\partial \tilde{y}_{T-1}} \\ & & & & & -1 \end{bmatrix} \begin{bmatrix} \omega_{i1} \\ \omega_{i2} \\ \omega_{i3} \\ \vdots \\ \omega_{i(T-1)} \\ \omega_{iT} \end{bmatrix} = \begin{bmatrix} \nabla_{\tilde{y}_1} \bar{f^i}^T \\ \nabla_{\tilde{y}_2} \bar{f^i}^T \\ \nabla_{\tilde{y}_3} \bar{f^i}^T \\ \vdots \\ \nabla_{\tilde{y}_{T-1}} \bar{f^i}^T \\ \nabla_{\tilde{y}_T} \bar{f^i}^T \end{bmatrix} \tag{3.5}$$

In (3.5), the reverse-mode operation of AD has the best performance to calculate $\omega_{ij}, j = 1 : T$: compute ω_{iT} first and then compute $\omega_{i(T-1)}$ based on ω_{iT}, and etc., until ω_{i1} is obtained. The most advantage of the reverse mode is that we can obtain all ω_{ij} without actually computing the whole extended Jacobian matrix $\hat{J}^E_{\tilde{y}}$, which contributes a significant reduction in computation. Thus, equation (3.4) can be simplified as:

$$\nabla f^T_{x^i} = -\omega_{i1} \cdot \frac{\partial \tilde{F}_1(x^i)}{\partial x^i} \tag{3.6}$$

Next, we define a new function $h(x^i, \tilde{y}, \omega_i)$ for the Hessian computation:

$$h(x^i, \tilde{y}, \omega_i) = \bar{f}^i(x^i, \tilde{y}) + \sum_{j=1}^{T} \omega^T_{ij} \cdot \tilde{F}_j(x^i, \tilde{y}_j) \tag{3.7}$$

where the vector-variable ω_{ij} is the same as (3.5). Scalar-valued function $\bar{f}^i$ represents the output of (3.2), and $\tilde{y}$ represents all the intermediate variables of path i in (3.2). Then, we define the new 'auxiliary Hessian' matrix for the composite function:

$$H^s = \left[\begin{array}{c|cc} \hat{J}^E_{x^i} & \hat{J}^E_{\tilde{y}} & 0 \\ \nabla^2_{\tilde{y}_j x^i} h_i & \nabla^2_{\tilde{y}_k \tilde{y}_j} h_i & (\hat{J}^E_{\tilde{y}})^T \\ \hline \nabla^2_{x^i x^i} h_i & \nabla^2_{x^i \tilde{y}_j} h_i & (\hat{J}^E_{x^i})^T \end{array}\right] = \left[\begin{array}{c|c} A_i & L_i \\ \hline B_i & M_i \end{array}\right]. \tag{3.8}$$

Note $j, k = 1, \ldots, T$.

The Hessian of f^i related to H^s satisfies:

$$\begin{aligned} H^s &= B_i - M_i L_i^{-1} A_i \\ &= \nabla^2_{x^i x^i} + \nabla^2_{x^i \tilde{y}_j} h_i \cdot y_{x^i} + y^T_{x^i} \cdot \nabla^2_{\tilde{y}_j x^i} h_i + y^T_{x^i} \cdot \nabla^2_{\tilde{y}_j \tilde{y}_k} h_i \cdot y_{x^i} \end{aligned} \tag{3.9}$$

where $y_{x^i} = -(\hat{J}^E_{\tilde{y}})^{-1} \cdot \hat{J}^E_{x^i}$.

In addition, among ω^T_{ij}, $\hat{J}^E_{\tilde{y}}$, and $-\nabla \bar{f}^T_{\tilde{y}}$, we have:

$$\hat{J}^E_{\tilde{y}} \cdot \omega^T_{ij} = -\nabla_{\tilde{y}} \bar{f}^T \tag{3.10}$$

and

$$\omega^T_{ij} \cdot \hat{J}^E_{x^i} = -\nabla \bar{f}^T_{\tilde{y}} \cdot (\hat{J}^E_{\tilde{y}})^{-1} \cdot \hat{J}^E_{x^i} = \nabla \bar{f}^T_{\tilde{y}} \cdot y_{x^i} \tag{3.11}$$

Thus, the intermediate parameters y_{x^i} can be computed by:

$$y_{x^i} = \nabla \bar{f}^{-T}_{\tilde{y}_j} \cdot \omega^T_{ij} \cdot \hat{J}^E_{x^i} \tag{3.12}$$

We develop the Algorithm 3 for the segment-wise gradient computation in a Monte Carlo setting:

Algorithm 3: *Compute Implicit_Structured_Gradient by segment*

```
1. Evaluate all $\tilde{y}_1, \tilde{y}_2, \ldots, \tilde{y}_T$, and store $\tilde{y}_T$;
2. Evaluate $\bar{f}(x^i, \tilde{y}_1, \tilde{y}_2, \ldots, \tilde{y}_T) = z^i$,
   and obtain $\nabla_{x^i} \bar{f}^T = (\nabla \bar{f}^T_{x^i}, \nabla \bar{f}^T_{\tilde{y}_1}, \nabla \bar{f}^T_{\tilde{y}_2}, \nabla \bar{f}^T_{\tilde{y}_3}, \ldots, \nabla \bar{f}^T_{\tilde{y}_T})$;
3. Calculate gradient:
   Initial $v_j = 0, j = 1 : n, \nabla f^T_{x^i} = \nabla \bar{f}^T_{x^i}$.
   For $j = T, T-1, \ldots, 1$
   $-\omega_{ij} = \nabla \bar{f}_{\tilde{y}_j} - v_j$;
   Evaluate $g(\tilde{y}_{T-1}, z^i)$, and use reverse-mode AD with $\omega_{ij}$
   to compute $\omega_{ij} \cdot (0, \ldots, \frac{\partial \tilde{F}_j(\tilde{y}_{j-1})}{\partial \tilde{y}_{j-1}}, -1, 0, \ldots, 0)$. Set $v_j = v_j + \omega_{ij} \cdot \frac{\partial \tilde{F}_j(\tilde{y}_{j-1})}{\partial \tilde{y}_{j-1}}, j = 1, \ldots, T$;
4. Update $\nabla f^T_{x^i} = \nabla \bar{f}^{-T}_{x^i} - \omega_{i1} \cdot \frac{\partial \tilde{F}_1(x^i)}{\partial x^i}$;
5. For $j = 1, 2, \ldots, T$
   Evaluate and store $y_{x^i j} = \nabla \bar{f}^{-T}_{\tilde{y}_j} \cdot \omega_{i1} \cdot \frac{\partial \tilde{F}_1(x^i)}{\partial x^i}$.
```

Algorithm 4 is for the segment-wise Hessian (H_s^i) computation:

```
Algorithm 4: Compute-Structured-Hessian in segment level

1. Obtain $\omega_{ij}$ and $y_{x^i j}$ from Algorithm 1;

2. Set $\begin{cases} \nabla^2_{x^i x^i} h(x^i, \tilde{y}_j, \omega_i) = \nabla^2_{x^i x^i} \bar{f}(x^i, \tilde{y}_j) \\ \nabla^2_{x^i \tilde{y}_j} h(x^i, \tilde{y}_j, \omega_i) = \nabla^2_{x^i \tilde{y}_j} \bar{f}(x^i, \tilde{y}_j) \\ \nabla^2_{\tilde{y}_k \tilde{y}_j} h(x^i, \tilde{y}_j, \omega_i) = \nabla^2_{\tilde{y}_k \tilde{y}_j} \bar{f}(x^i, \tilde{y}_j) \end{cases}$;

3. For $j = 1, 2, \ldots, T; k = 1, 2, \ldots, T$
   Update partial derivatives directly:
   $\begin{cases} \nabla^2_{x^i x^i} h(x^i, \tilde{y}_j, \omega_i) = \nabla^2_{x^i x^i} h(x^i, \tilde{y}_j, \omega_i) + \nabla^2_{x^i x^i} \omega_{ij}^T \cdot \tilde{F}_j(x^i, \tilde{y}_j) \\ \nabla^2_{x^i \tilde{y}_j} h(x^i, \tilde{y}_j, \omega_i) = \nabla^2_{x^i \tilde{y}_j} h(x^i, \tilde{y}_j, \omega_i) + \nabla^2_{x^i \tilde{y}_j} \omega_{ij}^T \cdot \tilde{F}_j(x^i, \tilde{y}_j) \\ \nabla^2_{\tilde{y}_k \tilde{y}_j} h(x^i, \tilde{y}_j, \omega_i) = \nabla^2_{\tilde{y}_k \tilde{y}_j} h(x^i, \tilde{y}_j, \omega_i) + \nabla^2_{\tilde{y}_k \tilde{y}_j} \omega_{ij}^T \cdot \tilde{F}_j(x^i, \tilde{y}_j) \end{cases}$;

4. Vector multiplication:
   $\begin{cases} C_i = \nabla^2_{x^i \tilde{y}_j} h(x^i, \tilde{y}_j, \omega_i) \cdot y_{x^i} \\ R_i = y_{x^i}^T \cdot \nabla^2_{\tilde{y}_k \tilde{y}_j} h(x^i, \tilde{y}_j, \omega_i) \cdot y_{x^i} \end{cases}$;

5. Segment level Hessian matrix is:
   $H_s^i = \nabla^2_{x^i x^i} h(x^i, \tilde{y}_j, \omega_i) + C_i + C_i^T + R_i.$
```

In Algorithm 4, we assume that $h(x^i, \tilde{y}_j, \omega_i)$ is not a function of x^i:

$$\nabla^2_{x^i x^i} h(x^i, \tilde{y}_j, \omega_i) = \nabla^2_{x^i \tilde{y}_j} h(x^i, \tilde{y}_j, \omega_i) = \nabla^2_{\tilde{y}_j x^i} h(x^i, \tilde{y}_j, \omega_i) = 0 \tag{3.13}$$

Thus, H_i^s can be simplified as:

$$H_i^s = y_{x^i}^T \cdot \nabla^2_{\tilde{y}_k \tilde{y}_j} h(x^i, \tilde{y}_j, \omega_i) \cdot y_{x^i} \tag{3.14}$$

where $\nabla^2_{\tilde{y}_k \tilde{y}_j} h(x^i, \tilde{y}_j, \omega_i) = 0$, if $k \neq j$. Therefore, $\nabla^2_{yy} h$ is a block diagonal matrix. A simplified version for the Algorithm 4 is:

```
Algorithm 4 (simplified): Compute-Structured-Hessian in segment level

1. Obtain $\omega_{ij}$ and $y_{x^i j}$ from Algorithm 3;
2. Set $\begin{cases} \nabla^2_{x^i x^i} h(x^i, \tilde{y}_j, \omega_i) = 0 \\ \nabla^2_{x^i \tilde{y}_j} h(x^i, \tilde{y}_j, \omega_i) = 0 \\ \nabla^2_{\tilde{y}_k \tilde{y}_j} h(x^i, \tilde{y}_j, \omega_i) = \nabla^2_{\tilde{y}_k \tilde{y}_j} \bar{f}(x^i, \tilde{y}_j, \omega_i) \end{cases}$;
3. For $j = 1, 2, \ldots, T; k = 1, 2, \ldots, T$
   Update partial derivatives directly:
       $\nabla^2_{\tilde{y}_k \tilde{y}_j} h(x^i, \tilde{y}_j, \omega_i) = \nabla^2_{\tilde{y}_k \tilde{y}_j} h(x^i, \tilde{y}_j, \omega_i) + \nabla^2_{\tilde{y}_k \tilde{y}_j} \omega_{ij}^T \cdot \tilde{F}_j(x^i, \tilde{y}_j)$;
4. Segment level matrix addition:
       $H_i^s = y_{x^i}^T \cdot \nabla^2_{\tilde{y}_k \tilde{y}_j} h(x^i, \tilde{y}_j, \omega_i) \cdot y_{x^i}$.
```

3.2 Final Hessian evaluation of all paths

We can apply the segment-wise structured AD method to obtain $z^i = f^i(x)$ and $\nabla f_x^i (i = 1, \ldots, p)$ in each Monte Carlo path. The final Hessian of the original evaluated function f can be obtained by an appropriate combination of all the Hessian result in each path.

In order to compute the final Hessian, we recall the Monte Carlo evaluation principle in Figure 2.1, and obtain the outputs $z^i = f^i(x)$ of all paths. The evaluated function f can be presented in the structured form:

$$\begin{cases} \text{solve for } z^1: & f^1(x) - z^1 = 0 \\ \text{solve for } z^2: & f^2(x) - z^2 = 0 \\ \text{solve for } z^3: & f^3(x) - z^3 = 0 \\ \quad\vdots & \quad\vdots \\ \text{solve for } z^p: & f^p(x) - z^p = 0 \\ \text{solve for } z: & \bar{f}(x, z^1, z^2, z^3, \ldots, z^p) - z = 0 \end{cases} \tag{3.15}$$

where each z^i is a scalar-valued function. $\bar{f}$ is a simple, often linear, function. In the Monte Carlo setting, we have $\bar{f}(x, z^1, z^2, z^3, \dots, z^p) = \frac{1}{p}\sum_{i=1}^{p} z^i$.Then, the final extended Jacobian matrix of all paths is:

$$J^E = \left[\begin{array}{c|ccccc} (\nabla f^1(x))^T & -1 & & & & \\ (\nabla f^2(x))^T & & -1 & & & \\ (\nabla f^3(x))^T & & & -1 & & \\ \vdots & & & & \ddots & \\ (\nabla f^p(x))^T & & & & & -1 \\ \hline 0 & \frac{\partial f}{\partial z^1} & \frac{\partial f}{\partial z^2} & \frac{\partial f}{\partial z^3} & \cdots & \frac{\partial f}{\partial z^p} \end{array}\right] = \left[\begin{array}{c|c} \hat{J}^E_X & \hat{J}^E_Y \\ \hline \nabla_X f^T & \nabla_Y f^T \end{array}\right]. \tag{3.16}$$

The corresponding auxiliary Hessian matrix of evaluated function f is:

$$H^p = \left[\begin{array}{c|cc} \hat{J}^E_X & \hat{J}^E_Y & 0 \\ \nabla^2_{yx} h & \nabla^2_{yy} h & (\hat{J}^E_Y)^T \\ \hline \nabla^2_{xx} h & \nabla^2_{xy} h & (\hat{J}^E_X)^T \end{array}\right] = \left[\begin{array}{c|c} A & L \\ \hline B & M \end{array}\right]. \tag{3.17}$$

The final Hessian $H^p = B - ML^{-1}A$ can be presented as:

$$H^{Final}_{MC} = \nabla^2_{xx} h + \nabla^2_{xy} h \cdot y_x + y_x^T \cdot \nabla^2_{yx} h + y_x^T \cdot \nabla^2_{yy} h \cdot y_x, \tag{3.18}$$

where $y_x = -(\hat{J}^E_Y)^{-1} \cdot \hat{J}^E_X = \hat{J}^E_X$. Then, the final Hessian H^{Final}_{MC} for all paths is:

$$H^{Final}_{MC} = \nabla^2_{xx} h + \nabla^2_{xy} h \cdot (\hat{J}^E_X) + (\hat{J}^E_X)^T \cdot \nabla^2_{yx} h + (\hat{J}^E_X)^T \nabla^2_{yy} h \cdot (\hat{J}^E_X) \tag{3.19}$$

Moreover, in Monte Carlo settings, we assume that $z^i = f^i(x)$ is not the function of x_i, since x_i represents the initial value of input parameters. Thus, we have $\nabla^2_{xx} h = \nabla^2_{xy} h = \nabla^2_{yx} h = 0$. Therefore, H^{Final}_{MC} is actually the summation of H^s_i of all paths:

$$H^{all-paths}_{MC} = (\hat{J}^E_X)^T \cdot \nabla^2_{yy} h \cdot (\hat{J}^E_X) = \sum_{i=1}^{p} H^i_s \tag{3.20}$$

In segment-wise structured AD, the space requirement in each path is:

$$\sigma_s \sim |\nabla^2 h|_{jji} + \max_j \{\omega(\tilde{F}_j), j = 1, 2, \dots, T, \omega(\bar{f}^i)\}. \tag{3.21}$$

whereas, the space usage of straightforward reverse-mode AD is:

$$\sigma_{s-unsAD} \sim |\nabla^2 h|_{jji} + \{\sum_{j=1}^{T} \omega(\tilde{F}_j) + \omega(\bar{f}^i)\}. \tag{3.22}$$

Compare (3.21) with (3.22), it is clear that

$$\{\sum_{i=1}^{p}\{\sum_{j=1}^{T}\omega(\tilde{F}_j)+\omega(\bar{f}^i)\}\} \gg \max_j\{\omega(\tilde{F}_j), j=1,2,\ldots,T,\omega(\bar{f}^i)\},$$

especially for large number of MC paths.

For the final Hessian computation of the evaluated function, the total space requirement is the summation of the space usage of each paths sub-Hessian computation:

$$\sigma_{s-strAD-all\ paths} \sim \sum_{i=1}^{p}|\nabla^2 h|_{jji}+\max_j\{\omega(\tilde{F}_j), j=1,2,\ldots,T,\omega(\bar{f}^i)\}; \tag{3.23}$$

$$\sigma_{s-unsAD-all\ paths} \sim \sum_{i=1}^{p}|\nabla^2 h|_{jji}+\{\sum_{i=1}^{p}\{\sum_{j=1}^{T}\omega(\tilde{F}_j),\omega(\bar{f}^i)\}\}. \tag{3.24}$$

where $\sigma_{s-strAD-all\ paths}$ represents the total space usage for structured AD, and $\sigma_{s-unsAD-all\ paths}$ represents for straightforward AD.

Note in (3.23) and (3.24), $\sigma_{s-unsAD-all\ paths} \gg \sigma_{s-strAD-all\ paths}$, especially for large p and T. Instead of saving the whole computational graph, storage of the necessary information only contributes a lot to space saving. Therefore, the structured AD technique can greatly improve the space utilization efficiency, in the segment-wise Hessian computation.

In chapter 4, the experimental results show, more directly, how the difference of computational space utilization between structured AD and straightforward AD.

Chapter 4

Experimental Results

4.1 A Financial application

In many financial applications, the deterministic function can be presented in the structured form (2.1), which allows using the structured AD technique to compute Hessian matrices. In this chapter, we consider a simple option pricing model, the Heston [7], as our objective model. Then, we respectively apply the structured AD and straightforward AD to evaluate the Hessian of the Heston model in a Monte Carlo setting.

The Heston model can provide a good explanation of the volatility for an underlying asset . In the discrete stochastic process, the Heston model can be defined as:

$$\Delta S_j^i = \mu S_{j-1}^i \Delta t + \sqrt{v_{j-1}^i} S_{j-1}^i \Delta W_{j-1}^{S^i} \tag{4.1}$$

$$\Delta v_j^i = \kappa(\theta - v_{j-1}^i)\Delta t + \xi \sqrt{v_{j-1}^i} \Delta W_{j-1}^{v^i} \tag{4.2}$$

where $\Delta W_{j-1}^{S^i}$ and $\Delta W_{j-1}^{v^i}$ represent the Wiener process, corresponding to the random innovation Z_i. The variables in the Heston model are: μ is the risk-free interest rate; θ is the long-run average variance; κ is the rate at which v reverts to θ; ξ is the volatility of v.

When applying the structured AD with Monte Carlo simulation in the Heston model, i represents i^{th} path, and j represents the segments in each Monte Carlo path. In particular, for the path-wise computation, the stochastic processes from path 1 to p are examples of the evaluation of sub-functions $U(x, Z_i)$ in (2.13), where $i = 1 : p$. In the segment-wise computation: we further divide each path into T segments, and in each specific path i, the stochastic processes from segment 1 to T are examples of the evaluation of sub-functions $\tilde{F}_j$ in (3.2).

Under risk neutral pricing theory [7], the expectation of options price is defined as:

$$P = E[V(S)] \tag{4.3}$$

where $V(S)$ is the payoff function of the underlying asset S at maturity. The underlying asset value S is presented as the combination of initial parameters x and random variables Z:

$$S = g(x, Z) \tag{4.4}$$

where x is a vector variable of initial parameters, including risk free interest rate, initial value, volatility and some other deterministic parameters for the evolution of S. Z represents the random innovations. Thus, the price P has an integral form as:

$$P = \int V(g(x, Z))\rho(Z)dZ \tag{4.5}$$

where $\rho(Z)$ is the probability density function of Z and independent with the initial variable x. In particular, we can interchange the order of integration and differentiation in the integral form of P:

$$\frac{\partial P}{\partial x} = \frac{\partial}{\partial x}\int V(g(x, Z))\rho(Z)dZ = \int \frac{\partial V(g(x, Z))}{\partial x}\rho(Z)dZ \tag{4.6}$$

Next, we use the Monte Carlo simulation generating p paths to approximate the integral value. The approximate expectation formula in Monte Carlo simulation is:

$$\widehat{P} = \frac{1}{p}\sum_{i=1}^{p} V(g(x, Z_i)) = \frac{1}{p}\sum_{i=1}^{p} \hat{p}_i \tag{4.7}$$

$$\frac{\partial \widehat{P}}{\partial x} = \frac{1}{p}\sum_{i=1}^{p} \frac{\partial V(g(x, Z_i))}{\partial x} = \frac{1}{p}\sum_{i=1}^{p} \frac{\partial \hat{p}_i}{\partial x} \tag{4.8}$$

where the random innovation $Z_i \in R, (i = 1 : p)$ corresponding to the simulation paths in Figure 2.1. We focus on the path-wise evaluation of the Hessian in this chapter. Since all Monte Carlo paths are independent with each other, the structured AD can be tailored into the path-wise computation, and we can use Algorithm 1 and 2 to efficiently compute the Hessian.

There are two payoff functions of the option, one is for Call option as (1), and another is for Put option as (2). They are usually presented as:

$$[S_T - K]^+ (1) or [K - S_T]^+ (2) \tag{4.9}$$

We choose the European call option to do the path-wise computation. Combined with the payoff function, the path-wise evaluated estimator can be expressed as:

$$\frac{1}{p} \sum_{i=1}^{p} \frac{\partial V(g(x, Z_i))}{\partial x} 1_{(S_{T_i} > K)} \tag{4.10}$$

where $1_{(S_{T_i} > K)}$ is the indicator function which reflects whether the value of underlying asset S is bigger than strike price K at maturity time T. If $S_{T_i} < K$,we do not execute the call option, and the value of $\frac{\partial V(g(x,Z_i))}{\partial x} 1_{(S_{T_i} < K)}$ is zero. Thus, we do not need to compute $\frac{\partial V(g(x,Z_i))}{\partial x}$, and we can directly set the value of $\frac{\partial V(g(x,Z_i))}{\partial x} 1_{(S_{T_i} < K)}$ to zero. In particular, if using the structured reverse-mode AD, we can set the $\frac{\partial V(g(x,Z_i))}{\partial x} 1_{(S_{T_i} < K)}$ to zero before executing the reverse order evaluation of derivatives. Compared with the forward-mode AD, the reverse-mode AD can significantly save computational time and space with the ellipsis of a huge amount of derivative calculation. Hence, we choose reverse-mode in the structured AD technique for the path-wise Hessian computation.

Last, we compare the performance of the structured AD and straightforward AD in the Hessian computation. Theoretically, if the payoff function is in the structured form, the gradient and Hessian matrices are sparse, which can significantly reduce the computational complexity. However, if we do not consider the structure of payoff functions, straightforwardly using reverse-mode AD, we need to generate numerous zero elements in the gradient and Hessian matrices, which consumes massive CPU memory. In most cases, such a huge CPU memory requirement makes the Hessian computation very expensive, thereby making computers break without any outputs.

4.2 Comparison results

For consistency, all experiments are performed on the UW server with 262GB memory, while running Matlab R2015a under the Windows Sever 2008R2 system. The AD toolbox for the reverse-mode AD is ADMAT 2.0 [2].

Structured AD and straightforward AD are compared in three different versions. Firstly, we keep the Monte Carlo path number and segment number as constant, only increasing the asset number; secondly, only increasing the Monte Carlo path number; thirdly, only increasing the segment number. Finally, we compare the difference between the structured AD and straightforward AD in both time and space utilization.

Before performing the experiment, we set the following initial parameter values of x in the Heston model as:

Initial parameters of x in Heston Model		
S_0	abs(randn(NAssets,1))	Initial stock price
μ	0.005	Risk-free interest rate
v_0	$ones(NAsset, 1) * (\mu/2)^2$	Initial volatility
κ	0.1	Rate of reverting
ξ	$\kappa/4$	volatility of volatility v
K	mean(S_0)	Strike price
ρ	0.5	correlation of Wiener Processes

Table 4.1: Initial parameters of x in Heston Model

There are another three variables in each comparison form: NAsset represents the number of assets in portfolios; NMC Instance represents the Monte Carlo path number in simulations; NSegment represents the segment number in each Monte Carlo path. In the path-wise experiment, the segment number is the same in each Monte Carlo path.

4.3 Experimental results for only increasing Asset number

As the assets increase in numbers, both structured and straightforward AD require more space (tape) to complete the evaluation. When the MC path number is 1000 and segment number is 50, straightforward AD requires more than 500 times the space needed by structured AD. The structured AD requires only around 10MB tape, but straightforward AD needs more than 5000 MB tape.

When we increase the MC path number to 2000, the segment number to 100, and the number of assets to 200, the tape demand of straightforward AD rises to more than 40,000 MB tape, whereas the structured AD only needs 80MB. Therefore, the tape demand of straightforward AD significantly increases with the increasing number of assets, whereas the tape of structured AD only increases very slightly.

			Structured AD		Straightforward AD			
NAsset	NMC Instance	Nsegments	Tape (MB)	Time(s)	Tape (MB)	Time(s)	Tape Ratio	Time Ratio
10	1000	50	9.4123	347.0920	1984.4800	285.3985	210.8381	0.8223
20	1000	50	10.8047	354.3805	2246.9507	278.5683	207.9603	0.7861
40	1000	50	9.8884	353.3727	3481.1677	305.9876	352.0428	0.8659
80	1000	50	17.8151	364.4671	5240.2803	313.9215	294.1481	0.8613
100	1000	50	14.7242	364.5445	5233.2764	316.8103	355.4196	0.8691
10	2000	100	5.3425	1335.7755	7894.6515	1225.4974	1477.6871	0.9174
20	2000	100	8.9813	1352.5386	9644.0038	1149.5256	1073.7764	0.8499
40	2000	100	19.6760	1384.3007	13848.2293	1444.6707	703.8121	1.0436
80	2000	100	35.4456	1403.0559	20845.4553	1433.4063	588.0956	1.0216
100	2000	100	29.2938	1432.3050	20816.1889	1283.5254	710.5985	0.8961
200	2000	100	82.7546	1496.9616	41837.1331	3280.1093	505.5562	2.1912

Table 4.2: Only increasing Asset number

4.4 Experimental results for only increasing MC-Path number

When the number of assets and segments are constant, the space requirement for structured AD always remains constant no matter how much the number of Monte Carlo paths increases. In addition, the tape requirement of structured AD is significantly smaller than that of straightforward AD. If the asset number is 40, the Monte Carlo path is 8000 and the segment number is 50, then structured AD requires only 9.88MB tape to complete the evaluation, whereas the straightforward AD requires 27,849MB (equal to 27.2GB). In one more example, when we increase the asset number to 80, the MC path up 4000, and the segment number to 100, then structured AD requires only 35.44MB to compute the Hessian, whereas straightforward AD needs 41690MB (equal to 40.7GB), more than 1170 times of space requirement. No matter how large of the MC path number, structured AD can always evaluate the Hessian, but the straightforward AD will break when it runs out of CPU memory.

In real-world Hessian evaluation, such huge memory demand cannot be satisfied by most local computers. In addition, generating over 10,000 Monte Carlo paths for high accuracy is common, which makes using the straightforward AD for the Hessian computation very expensive. Thus, the structured AD technique significantly raises the space-utilization

			Structured AD		Straightforward AD			
NAsset	NMC Instance	Nsegments	Tape (MB)	Time(s)	Tape (MB)	Time(s)	Tape Ratio	Time Ratio
10	200	50	2.6839	66.0299	396.9067	56.8078	147.8832	0.8603
10	500	50	2.6839	167.3581	992.2467	144.9778	369.7007	0.8662
10	1000	50	2.6839	347.0920	1984.4800	285.3985	739.3963	0.8222
10	1500	50	2.6839	519.9241	2976.7133	428.3622	1109.092	0.8238
10	2000	50	2.6839	687.1953	3968.9465	558.3885	1478.7878	0.8125
10	8000	50	2.6839	2695.4196	15875.7458	2674.7563	5915.1359	0.9923
10	20000	50	2.6839	6745.8712	39689.3445	6054.8171	14787.832	0.8975
20	500	50	4.5130	175.4615	1123.4823	145.5246	248.9411	0.8293
20	1000	50	4.5130	354.3805	2246.9507	278.5683	497.8791	0.7861
20	1500	50	4.5130	545.5506	3636.4145	435.2705	805.7564	0.7978
20	2000	50	4.5130	733.9949	4848.5479	597.9032	1074.3409	0.8145
40	1500	50	9.8884	528.5701	5221.7435	468.7873	528.06337	0.8869
40	2000	50	9.8884	699.3492	6962.3193	619.8291	704.1355	0.8863
40	4000	50	9.8884	1414.3551	13924.6224	1221.5780	1408.1663	0.8637
40	8000	50	9.8884	2825.6240	27849.2288	2398.7290	2816.3309	0.8489
80	2000	100	35.4456	1403.0559	20845.4553	1433.4063	588.0956	1.0216
80	4000	100	35.4456	2819.3328	41690.8914	2456.3808	1176.1907	0.8713
80	8000	100	35.4456	5528.6956	Nan	Nan	Nan	Nan
100	2000	252	73.5856	3504.9084	52279.1405	3196.1551	710.4528	0.9119
100	3000	252	73.5856	5276.8601	78418.7012	4701.3353	1065.6791	0.8909
100	4000	252	73.5856	7042.8322	Nan	Nan	Nan	Nan
100	6000	252	73.5856	10617.6426	Nan	Nan	Nan	Nan
100	10000	252	73.5856	17230.0683	Nan	Nan	Nan	Nan

Table 4.3: Only increasing MC-Path number

efficiency in the Hessian evaluation, especially for larger numbers of Monte Carlo paths.

The comparison results are more obvious in graphs.

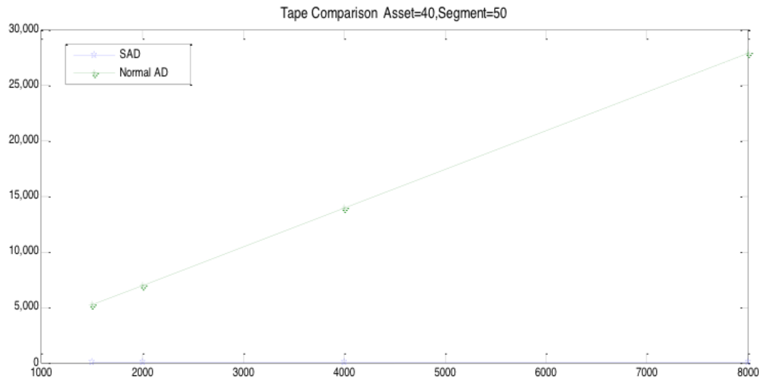

Figure 4.1: Tape Comparison Asset=40, Segments=50

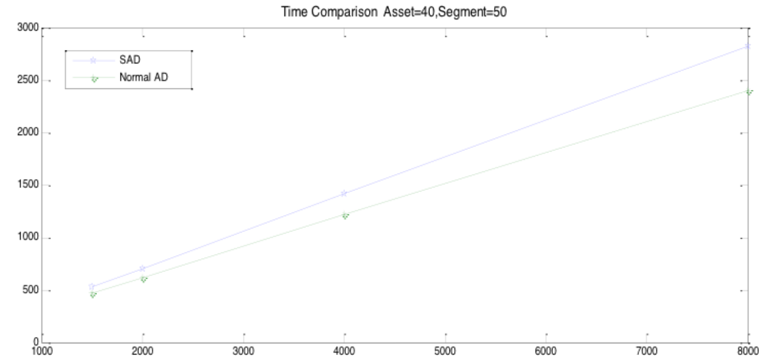

Figure 4.2: Time Comparison Asset=40, Segments=50

There is a significant difference in space utilization between the structured AD and straightforward AD in Figure 4.1. As for the time consuming aspect in Figure 4.4, the two AD techniques require a similar computational time for the Hessian evaluation.

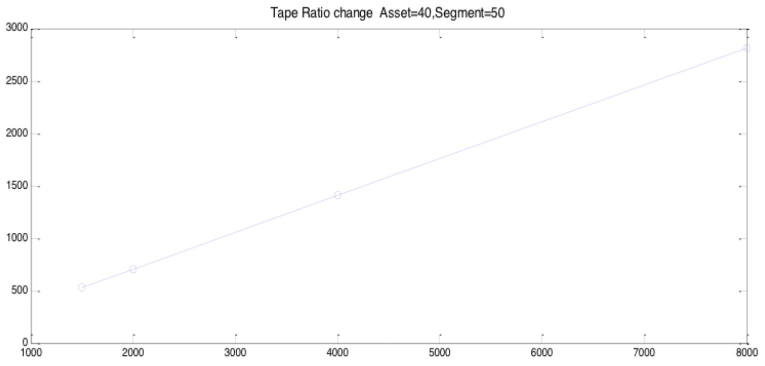

Figure 4.3: Tape Ratio Asset=40, Segments=50

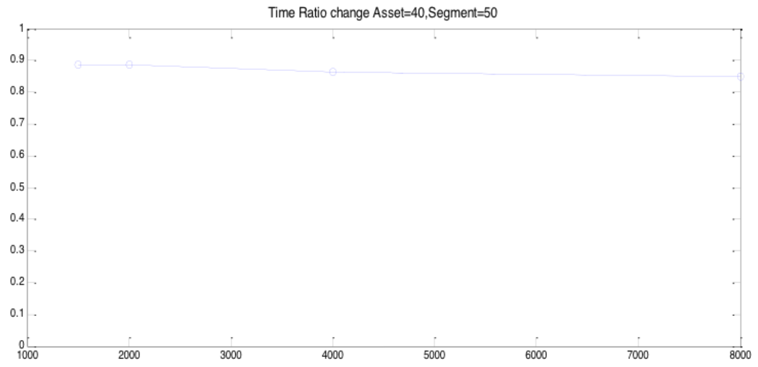

Figure 4.4: Time Ratio Asset=40, Segments=50

4.5 Experimental results for only increasing Segments number

			Structured AD		Straightforward AD			
NAsset	NMC Instance	Nsegments	Tape (MB)	Time(s)	Tape (MB)	Time(s)	Tape Ratio	Time Ratio
20	2000	50	21.5945	733.9949	4848.5479	597.9032	224.5261	0.8145
20	2000	100	21.5945	1414.9126	9644.0038	1149.5256	446.5937	0.8124
20	2000	252	22.5651	3341.1727	24222.1898	3197.1225	1073.4331	0.9568
40	2000	50	9.8884	699.3492	6962.3193	619.8291	704.0839	0.8863
40	2000	100	19.6760	1384.3007	13848.2293	1444.6707	703.8121	1.0436
40	2000	252	49.4302	3376.2734	34781.3958	3035.5861	703.6465	0.8991

Table 4.4: Only increasing segments number

When only the segment number increased, the tape increase conclusion is very similar to the former two cases. The structured AD technique also has a small space requirement for computation, around 20 MB, whereas the straightforward AD requires more than 20,000MB. In our experiment, the largest number of segments(time-step) is 252, since it represents the whole year time-steps in stock market.

To summarize, for computational space utilization, our structured AD has major advantage in path-wise Hessian computation, compared with the straightforward AD, especially for a large number of Monte Carlo paths.

Chapter 5

Conclusion

In quantitative finance, the evaluation of Hessian matrices is important for hedging and sensitivity analysis. However, twice differentiating the objective function can be difficult and costly. Many traditional methods are either too costly or inaccurate when computing the Hessian. Automatic differentiation is an alternative way to obtain the second derivatives. However, simply applying the straightforward AD is still relatively inefficient in Hessian evaluation. Thus, it is useful to develop a new efficient AD method to compute the Hessian.

The advent of the structured AD technique [14] provides a way forward to efficiently compute the Hessian. Many objective functions in practice can be presented in the structured form and this form usually exposes underlying sparsity. Our proposal is how to tailor the AD technique to the structured form, and use the 'hidden' sparse property to obtain the Hessian more efficiently. Compared with the straightforward AD, our structured AD can significantly improve the space utilization for structured problems arising in a Monte Carlo setting.

In this paper, we used the Heston model to develop numerical experiments. Our results show that the structured AD has a very high efficiency in computational space utilization. When the number of assets and time-segment number are constant, the structured AD only requires a small fixed computational space, regardless of how large number of the MC paths, whereas the space demand of straightforward AD soars for the large MC paths number. For example, when the number of assets is 10 and time-segment number is 50: for 2,000 MC paths, the structured AD requires 2.68MB tape, whereas the straightforward AD requires 3.88GB tape; for 20,000 MC paths, the structured AD still needs 2.68MB tape, however, the straightforward AD requires 38.76GB tape (14,788 times space more than that of the structured AD). In addition, we compared the final Hessian results between the two AD methods, and their absolute error is less than $1.0 * 10^{-14}$. Therefore, the new structured AD is a very efficient and accurate technique to compute the Hessian, and more applicable for local computers.

The Heston model is a stochastic volatility model for options pricing, and involves the evaluation of two stochastic differentiable equations of underlying assets S and the volatility. We applied the Heston model for Monte Carlo simulation to do the options pricing. For path-wise computation of the Hessian, the stochastic processes in the Heston model from path 1 to p are corresponding to ps Monte Carlo simulation paths, respectively, and all paths are independent with each other.

When the evaluated function is highly recursive, each Monte Carlo path involves a finite set of segments. In this case, we can use segment-wise computational algorithm to compute the Hessian. Theoretically, in structured AD technique, the segment-wise computation of the Hessian also has a very highly efficient space utilization, compared with the straightforward AD, although we did not do numerical experiments for it.

With the high efficiency and accuracy gains, the structured AD technique can be broadly used to compute the Hessian matrices for real-world applications. In addition, it is worth to do further research of segment-wise computation of the Hessian for more complicated Monte Carlo processes, such as the nested Monte Carlo cases. In conclusion, it is reasonable to believe that the appropriate application of structured AD technique of derivatives calculation will gain more efficiency in both time and space utilization.

References

[1] T. F. Coleman and G. F. Jonsson, *The efficient computation of structured gradients using automatic differentiation*, SIAM Journal on Scientific Computing, 20 (1999), pp. 1430–1437.

[2] T. F. Coleman and W. Xu, *Automatic differentiation in matlab using admat with applications*, 2016.

[3] M. C. Fu, *What you should know about simulation and derivatives*, Naval Research Logistics (NRL), 55 (2008), pp. 723–736.

[4] M. Giles and P. Glasserman, *Smoking adjoints: Fast monte carlo greeks*, Risk, 19 (2006), pp. 88–92.

[5] P. Glasserman, *Monte Carlo methods in financial engineering*, vol. 53, Springer Science &amp; Business Media, 2003.

[6] A. Griewank and A. Walther, *Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation*, ACM Transactions on Mathematical Software (TOMS), 26 (2000), pp. 19–45.

[7] S. L. Heston, *A closed-form solution for options with stochastic volatility with applications to bond and currency options*, Review of financial studies, 6 (1993), pp. 327–343.

[8] C. Homescu, *Adjoints and automatic (algorithmic) differentiation in computational finance*, Available at SSRN 1828503, (2011).

[9] M. Joshi and C. Yang, *Algorithmic hessians and the fast computation of cross-gamma risk*, IIE Transactions, 43 (2011), pp. 878–892.

[10] C. Käbe, J. H. Maruhn, and E. W. Sachs, *Adjoint-based monte carlo calibration of financial market models*, Finance and Stochastics, 13 (2009), pp. 351–379.

[11] A. Walther, *Computing sparse hessians with automatic differentiation*, ACM Transactions on Mathematical Software (TOMS), 34 (2008), p. 3.

[12] Wikipedia, *Chain rule.*

[13] W. Xu, X. Chen, and T. F. Coleman, *The efficient application of automatic differentiation for computing gradients in financial applications*, J. Comput. Finance, (2014).

[14] W. Xu, S. Embaye, and T. F. Coleman, *Efficient computation of derivatives and newton steps for minimization of structured functions using automatic differentiation*, (2016).