The Structured Automatic Differentiation Approach for Efficiently Computing Gradients from Monte Carlo Process

by

Zhongheng Yuan

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

© Zhongheng Yuan 2016

I hereby declare that I am the sole author of this report. This is a true copy of the report, including any required final revisions, as accepted by my examiners.

I understand that my report may be made electronically available to the public.

Abstract

The Monte Carlo process has been widely used in a variety of industrial and academic fields. Especially in the finance fields, efficiently computing the gradients of Monte Carlo process is often needed. The reverse-mode of Automatic Differentiation can be applied to compute the gradients in time as proportional to the time needed for computing the objective function. However, in practice, this efficiency could suffer from massive memory requirements. The subsequent exhaustion of fast memory and use of slow memory often significantly slows down the computation. Here, we illustrate that the evaluation of objective function in Monte Carlo process often exhibit special structure that can be used to reduce the memory requirement.

To overcome the excessive memory requirements, we will demonstrate a structured reverse-mode AD approach to compute the gradients of a Monte Carlo process. This approach takes advantage of its structure characteristics; therefore, it requires much less memory, and improve the efficiency on computing the gradients. Experiments based on the well-known Heston Model show that this method significantly reduces computing time.

Acknowledgements

I would like to thank my supervisor, Professor Thomas F. Coleman for his generous support and valuable guidance on my research project. I would like to thank Professor Henry Wolkowicz for taking the time to review this paper. I would also like to thank my colleage, Wanqi Li for his help on implementing my research expriments on ADMAT software. Lastly, I would like to thank all of my classmates, staff and faculty members in the Computational Mathematics Master Program. I had a great master's study experience with their help and encouragements along the way.

Dedication

This is dedicated to my loved family and friends.

Table of Contents

List of Tables viii

List of Figures ix

1 Introduction 1

1.1 Automatic Differentiation Background 2

1.2 Monte Carlo with Automatic Differentiation 3

1.3 Overview of Paper 4

2 Basics of Automatic Differentiation and the Structure Idea 5

2.1 Forward-mode and Reverse-mode 5

2.2 Pros and Cons of Reverse-mode 6

2.3 Structured AD Techniques for Computing Gradients 7

2.4 Time and Space for Structured AD 9

3 Monte Carlo Process for Portfolio Pricing 10

3.1 Evaluation of a Single Path 10

3.2 Evaluation for p Paths 11

3.3 Monte Carlo in a Flowchart View 12

3.4 Two Structured Functions 13

3.4.1 Composite Function 13

3.4.2 Generalized Partially Separable Function 14

4 Computing the Gradients of Monte Carlo process **16**

4.1 Evaluation of $\frac{dV}{dx}$ Based on Composite Function's Structure 16

4.2 Evaluation of $\frac{\widehat{dP}}{dx}$ Based on GPS Function's Structure 21

4.3 Minimizing The Memory Requirement 24

5 Numerical Results **26**

5.1 Experiment Design 26

5.2 Test I - SAD v.s DAD 27

5.3 Test II - SAD v.s SAD-PathOnly 30

6 Conclusion **32**

References **34**

List of Tables

5.1 Initial Inputs for Heston Model 27
5.2 Comparison of Memory and Time between Strcuture AD and Direct AD, NSegments=252 28
5.3 Comparison of Memory and Time between SAD and SAD-PathOnly, NAssets=10,000, NPaths=10 31

List of Figures

3.1 Monte Carlo Process for Portfolio Pricing 12

3.2 Computational steps in flowchart format for composite function. (see Figure 3 in [5]) 13

3.3 Computational steps in flowchart format for generalized partical separable function. (see Figure 4 in [5]) 15

4.1 Ideal Running Time comparison between Strcutured AD and Direct AD . . 21

5.1 Comparions of Memory and Time for SAD and DAD: Topleft and Topright graphs are the memory and time comparisons for NAssets=100 and NSegments=252 with different choices of NPaths; bottomleft and bottomright graphs are the memory and time comparisons for NPaths=400 and NSegments=252 with different choices of NAssets. 29

5.2 Comparions of Memory and Time for SAD and SAD-PathOnly: Left and Right graphs are the memory and time comparisons for NAssets=10,000 and NPaths=10 with different choices of NSegments. 31

Chapter 1

Introduction

In the field of scientific computing, we often need to calculate the partial derivatives. In some cases, such as applying Newtons Method on a differentiable scalar-valued function, the first partial derivatives are required. Another example is solving optimization problems for non-linear regression; in this case repeated calculations of derivatives, i.e. Jacobian matrices are required. Engineering applications normally depend on a variety of scientific computing methods. In financial engineering, it is an important task to accurately and efficiently compute partial derivatives.

One of the most popular applications for derivatives computation in financial engineering is the determination of "Greeks" for financial instruments to hedge. Derivatives are used in order to measure the sensitivity to stock price, interest rates, or volatility. For example, in Delta Hedging we need to compute the first derivative of option price with respect to stock price; in Gamma Hedging we need to compute both first and second partial derivatives. In these cases, Monte Carlo is often used to evaluate functions with stochastic terms. With the rapid development of computing power, using Monte Carlo process to perform a large amount of simulations is becoming increasingly popular. However, based on the complexity of financial models and financial instruments, the computation of derivatives in Monte Carlo settings can become an extremely expensive task, especially if the usual finite-difference approach is taken.

In general, we need to consider two aspects of a computational task: time and space. There are several approaches for the computation of partial derivatives, such as finite differencing, symbolic differentiation, hand coding of the derivative functions, and automatic differentiation (AD). Each of these approaches has its advantages and drawbacks. In the early stages, AD was criticized on time cost and large space requirements. However, with

recent technical advances AD has partially overcome these drawbacks and demonstrates superior advantages over other methods on derivatives computation. Andreas Griewank, Chris Bischof and their colleagues have contributed significantly into the recent development of AD as a very practical tool for derivatives computation [1, 9, 8]. In the field of scientific computing, AD is becoming an important method and attracting more interests [11]. Coleman and Xu, designers and developers of ADMAT[1], believe that AD can be the best available technology for calculating partial derivatives in scientific computing [6].

1.1 Automatic Differentiation Background

The earliest form of AD is the straightforward implementation of chain rule, also known as the forward-mode AD. On the other hand, reverse-mode AD was recognized by researchers [14, 13, 2] for its time efficiency in computing gradients. The downside of using reverse-mode, compared to forward-mode is the massive memory requirement to store the computational "tape". The "tape" refers to the space needed to save the entire computational graph, i.e. all intermediate variables of the computation to evaluate the function. After the evaluation of objective function, reverse-mode uses the "tape" to roll back from the end to the start to compute derivatives. When we are faced with a relatively large and complex problem for gradients computation, the drawback of massive memory requirement limits the usage of reverse-mode AD.

In order to overcome the memory disadvantage of reverse-mode AD, researchers have used a computer science technique called "checkpointing" [10] to develop a structured reverse-mode AD approach. Originally, for reverse-mode, we need to store the whole computational "tape" of a differential function. Instead, we can set a finite number of checkpoints in the computational graph and save necessary state information. After forward evaluating the objective function, the checkpoints based computational "tape" allows us to go backwards for computing the derivatives. This requires much less memory compare to the entire computational "tape" stored by using direct reverse-mode. The computational time cost remains proportional to the theoretical time required by reverse-mode AD without checkpoints. [2]

[1] An AD toolbox developed in MATLAB environment, see [6] for details.

[2] Though some computational costs might occur due to overhead costs, the total cost for this method is just a constant factor times the cost for evaluating the objective function.

1.2 Monte Carlo with Automatic Differentiation

A Monte Carlo process is often used for function evaluations in financial engineering. It simulates a large set of paths where each path starts with different randomized inputs. Moreover, each path may be broken into a number of timesteps, or segments. The massive amout of simulations, combined with possible complex financial models used, can result in a large numerical problem. The problem we consider here is: how to efficiently compute the gradient of a Monte Carlo function using AD.

In order to compute the gradients, one of the best tools to efficiently solve the problem is reverse-mode AD. In [4, 3, 7], the reverse-mode AD was applied to compute Greeks in a Monte Carlo setting. In 2009, Kaebe and Maruhn have used a reverse-mode AD method to improve the efficiency of Monte Carlo based calibration of financial market models [12], but they did not consider the potential drawback of massive memory requirement. Generally, for a large numerical problem, the massive memory requirements created from the computational "tape" cannot be ignored. For example, for a Monte Carlo process with 10^4 paths and 10^2 timesteps, if we assume the space requirement for each timestep's evaluation to be 10^4 bytes then the total space requirement for using reverse-mode AD is $10^4 * 10^2 * 10^4 = 10^{10}$ bytes ≈ 9.31 gigabytes. Thus the space requirement can become very expensive for Monte Carlo processes with many paths and timesteps. In practice, this may result in exhaustion of available fast memory and slows down the computational efficiency. In order to efficiently apply reverse-mode AD, works have shown that the application of structured reverse-mode AD can be the right tool to solve this kind of issue. Xu, Chen, and Coleman [15] have presented a structured reverse-mode AD approach to reduce the memory requirement for computing gradients in several financial applications, including Monte Carlo process. Their work was mainly focused on using the idea of "checkpointing" to take advantage on memory saving when the number of paths of Monte Carlo process gets large.

In practice, a Monte Carlo process can also involve a large number of segments (i.e. timesteps) within each path. This can eventually create excessive memory issues as well. In this essay, we focus on a Monte Carlo setting and demonstrate a structured reverse-mode AD approach applied both on the evaluation of Monte Carlos paths and the evaluation of each paths segments to efficiently calculate the gradients by further reducing the memory requirement. This structured reverse-mode AD based on the application of the "checkpointing" technique can transform this large numerical problem into a highly feasible one with taking advantages of the special structured characteristics of the Monte Carlo process. As a result, the significant savings on memory requirement provide us the feasibility of using localized memory only and maintaining the advantage of time efficiency from

reverse-mode AD.

1.3 Overview of Paper

The remainder of this this essay is organized as follows, Chapter 2 introduces basic concepts of forward-mode and reverse-mode AD methods with details on the analysis of intermediate variables and the formation of the extended Jacobian matrix. After the basics, we will illustrate the idea of using structured reverse-mode AD method to reduce the space requirement. In Chapter 3, we will review the evaluation and gradients calculations of a Monte Carlo process on a financial portfolio pricing, followed by Chapter 4 where we express the detailed analysis and structured reverse-mode AD algorithms on computing gradients for a Monte Carlo process. Numerical results are included in Chapter 5. We make time and space comparisons for applying different reverse-mode AD methods. Experiments are performed on computing gradients for the Monte Carlo process on Heston Model. The results illustrate that the space are significant reduced and the time efficiency are more stable on using the structured reverse-mode AD. Lastly, we give a general summary in Chapter 6.

Chapter 2

Basics of Automatic Differentiation and the Structure Idea

2.1 Forward-mode and Reverse-mode

The general idea behind AD is straightforward. Assume we have a differentiable mapping $z = F(x), F : R^n \rightarrow R^m$. If we want to evaluate F on a machine, it can always be expressed as an ordered sequence of intermediate atomic operations[1]. Since the atomic operations are in an ordered sequence, each of them is determined by inputs only from the original input x or previous computed intermediate variables. Thus the evaluations of these atomic operations and the calculation of their derivatives with respect to the inputs are easy to compute. Based on these atomic functions and the application of chain rule, we are able to obtain the extended Jacobian matrix J^E of the mapping F with respect to x and intermediate variables. Then we can compute the Jacobian matrix J of F with respect to x from J^E.

As introduced before, two kinds of AD method are applicable to obtain the Jacobian matrix for the objective mapping F, the forward-mode AD and the reverse-mode AD. Forward-mode AD is the simple idea of calculating the derivatives of each atomic operation with respect to their inputs while computing each atomic operation. We can apply the chain rule on the results of these derivatives and compute the Jacobian matrix of F.

Reverse-mode AD, in contrast, does not compute the derivatives while evaluating the

[1]We define an atomic operation as a simple function that takes unitary or binary inputs and only performs basic mathematic operations, such as +,-,*,/,sin,cos,...

atomic operations. Instead, it stores the intermediate variables from evaluating atomic operations on a computational "tape". After finishing evaluating the value of F, it rolls back from the end to the start of the "tape" and apply chain rule to obtain the Jacobian matrix with respect to the initial inputs.

2.2 Pros and Cons of Reverse-mode

In some cases, it is more efficient to compute the Jacobian matrix J when reverse-mode AD is used. Assume that for the differentiable mapping F, the time for evaluating F is $\omega(F)$ and the space requirement for evaluation is $\sigma(F)$. For forward-mode AD, the time for evaluating the Jacobian matrix J is proportional to $n \cdot \omega(F)$. In contrast, for reverse-mode AD, the time needed for computing J is $m \cdot \omega(F)$. The difference in running time tells us to simply apply forward-mode AD when $n \leq m$ and apply reverse-mode AD when $n > m$ if we do not consider other computational issues.

In this paper, we are mainly concerned on efficiently computing the gradients of a Monte Carlo process. It is clear that for gradients, we have a scalar-valued objective function $f : R^n \to R^1$, i.e. $m = 1$. Based on the previous discussions, if we denote the time for evaluation of f is $\omega(f)$, then the time needed for computing the gradients is proportional to $n \cdot \omega(f)$ for forward-mode AD and $1 \cdot \omega(f)$ for reverse-mode AD. Thus there is significant savings on computing time if we apply reverse-mode AD.

However, reverse-mode AD has its own drawbacks. Compared to forward-mode, reverse-mode AD trades off space requirement for time efficiency. The forward-mode AD evaluates the intermediate functions and their corresponding derivatives along the way. It only needs space that is required for evaluation of the objective function F and for storing the Jacobian matrix J, i.e. the space requirement for forward-mode is proportional to $\sigma(F)$. On the other hand, in order to perform reverse order calculations, the reverse-mode AD needs to store the matrix J^E. This is the entire computational steps involved in evaluation of F, thus the space requirement for reverse-mode is proportional to $\omega(F)$. In general, $\sigma(F) \ll \omega(F)$ and it can be a big challenge to apply reverse-mode AD on a real machine.

How would the massive space requirement lead to the unfeasibility of using reverse-mode? The answer relies on the saturation of fast memory. For example, if the number of intermediate atomic operations is extremely large, we need to store the computational "tape" on a machine's memory. If the space requirement goes over the capacity of fast memory, the machine needs to start accessing slow memory and this results in much longer running time for the computation. When this happens, the running time in theory, as

proportional to $m \cdot \omega(F)$, is no longer possible to achieve for applying direct reverse-mode AD. In order to reduce the space requirement, we can apply the structured idea and improve the feasibility of using reverse-mode AD. For clearer definitions in this paper, we will denote the AD method of using the strcutured idea as structured reverse-mode AD method and the direct application of reverse-mode AD as direct reverse-mode AD.

2.3 Structured AD Techniques for Computing Gradients

When faced with objective functions that exhibit structure in the following case, instead of using direct reverse-mode AD method, we will take advantage of sparsity of the structure. Since we are focusing on the structured approach on computing the gradients in this paper, we will demonstrate the strcutured idea in the case of computing gradients. Assume we have a scalar mapping $f(x) : \mathbb{R}^n \to \mathbb{R}^1$ and we can write its computation steps as the following:

$$\left.\begin{array}{ll} \text{Solve for } y_1 & : F_1^E(x, y_1) \equiv \tilde{F}_1(x) - M_1 \cdot y_1 = 0 \\ \text{Solve for } y_2 & : F_2^E(x, y_1, y_2) \equiv \tilde{F}_2(x, y_1) - M_2 \cdot y_2 = 0 \\ \vdots & \\ \text{Solve for } y_p & : F_p^E(x, y_1, y_2, \cdots, y_p) \equiv \tilde{F}_p(x, y_1, y_2, \cdots, y_{p-1}) - M_p \cdot y_p = 0 \\ \text{Solve for } z & : F_{p+1}^E(x, y_1, y_2, \cdots, y_{p+1}) \equiv \bar{f}(x, y_1, y_2, \cdots, y_p) - z = 0 \end{array}\right\} . \tag{2.1}$$

Here, we need all intermediate functions and the final scalar evaluation function to be differentiable. Intermediate variables $y_1, .., y_p$ are vectors of different lengths and matrices $M_1, ..., M_p$ are non-singular. Then we can write its extended Jacobian in the form as:

$$J^E = \begin{pmatrix} J_x^1 & -M_1 & & & & \\ J_x^1 & J_{y_1}^2 & -M_2 & & & \\ \vdots & \vdots & \ddots & \ddots & & \\ \vdots & \vdots & & \ddots & \ddots & \\ J_x^p & J_{y_1}^p & J_{y_2}^p & & J_{y_{p-1}}^p & -M_p \\ \nabla \bar{f}_x^T & \nabla \bar{f}_{y_1}^T & \nabla \bar{f}_{y_2}^T & \cdots & \cdots & \nabla \bar{f}_{y_p}^T \end{pmatrix} . \tag{2.2}$$

Furthermore, we can partition J^E as:

$$J^E = \left(\begin{array}{c|ccccc} J_x^1 & -M_1 & & & & \\ J_x^1 & J_{y_1}^2 & -M_2 & & & \\ \vdots & \vdots & \ddots & \ddots & & \\ \vdots & \vdots & & \ddots & \ddots & \\ J_x^p & J_{y_1}^p & J_{y_2}^p & & J_{y_{p-1}}^p & -M_p \\ \hline \nabla \bar{f}_x^T & \nabla \bar{f}_{y_1}^T & \nabla \bar{f}_{y_2}^T & \cdots & \cdots & \nabla \bar{f}_{y_p}^T \end{array}\right) = \left(\begin{array}{c|c} A & L \\ \hline \nabla \bar{f}_x^T & \nabla \bar{f}_y^T \end{array}\right). \quad (2.3)$$

From the extended Jacobian and based on Schur-complement computation, we can compute the gradients of objective function f, with respect to x as:

$$\nabla f^T(x) = \nabla \bar{f}_x^T - \nabla \bar{f}_y^T L^{-1} A. \quad (2.4)$$

In [15], the idea is to first evaluate $f(x)$ and save all intermediate variables $y_1, ..., y_p$ along the computation. Then we apply reverse-mode AD to $\bar{f}(x, y)$ to compute $\nabla \bar{f}_x^T$ and $\nabla \bar{f}_y^T$. Interestingly, we do not need to compute the matrix L directly. Instead, we can use reverse-mode AD step-by-step to finish the computation of the gradient.

To illustrate the approach in detail, [16] shows the method to compute $v^T = \nabla \bar{f}_y^T L^{-1} A$. Here, lets define $w^T = (w_1^T, ..., w_p^T)$ where $L^T w = \nabla \bar{f}_y$, or we can write it in the following form:

$$\begin{pmatrix} -M_1 & (J_{y_1}^2)^T & (J_{y_1}^3)^T & \cdots & (J_{y_1}^{p-1})^T & (J_{y_1}^p)^T \\ & -M_2 & (J_{y_2}^3)^T & \cdots & \vdots & (J_{y_2}^p)^T \\ & & \ddots & \cdots & \vdots & \vdots \\ & & & -M_{p-2} & (J_{y_{p-2}}^{p-1})^T & (J_{y_{p-2}}^p)^T \\ & & & & -M_{p-1} & (J_{y_{p-1}}^p)^T \\ & & & & & -M_p \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_{p-1} \\ w_p \end{pmatrix} = \begin{pmatrix} \nabla \bar{f}_{y_1} \\ \nabla \bar{f}_{y_2} \\ \nabla \bar{f}_{y_3} \\ \vdots \\ \nabla \bar{f}_{y_{p-1}} \\ \nabla \bar{f}_{y_p} \end{pmatrix} \quad (2.5)$$

Now we can compute v^T as :

$$v^T = \nabla \bar{f}^T B^{-1} A = w^T A = w_1^T J_x^1 + w_2^T J_x^2 + \cdots + w_{p-1}^T J_x^{p-1} + w_p^T J_x^p. \quad (2.6)$$

Hence, in order to compute the gradients of f in equation (2.4) , we sum all these above steps into the following structured reverse-mode AD algorithm:

Algorithm 1 *Structured Gradient Computation*

1. *Follow steps of* $i = 1, ..., p$ *only in* (2.1) *to evaluate values of* y_i.
2. *Follow step of* $i = p + 1$ *in* (2.1) *to evaluate* z *and compute* $\nabla \bar{f}_x^T$ *and* $\nabla \bar{f}_y^T$ *by using reverse-mode AD.*
3. *Use equation* (2.5) *and* (2.6) *to compute gradient:*
   (a) *Set* $v_i = 0, i = 1, ..., p, \nabla f = \nabla \bar{f}_x^T$,
   (b) *For* $j = p, p-1, ..., 1$
   - *Solve* $M_j w_j = \nabla \bar{f}_{y_j} - v_j$;
   - *Evaluate* $\bar{F}_j(x, y_1, ..., y_{j-1})$ *and use* w_j^T *on reverse-mode AD to compute* $w_j^T \cdot (J_x^j, J_{y_1}^j, ..., J_{y_{j-1}}^j)$.
   - *Set* $v_j^T = v_j^T + w_j^T J_{y_i}^j$ *for* $i = 1, ..., j-1$;
   - *Update* $\nabla f^T \leftarrow \nabla f^T + w_j^T J_x^j$;

2.4 Time and Space for Structured AD

By using Algorithm 1 for computing the gradient, we maintain a total running time being proportional to $\omega(f)$.[2] This means that the time cost for computing the gradient is similar to using direct reverse-mode AD method. However, our space requirement has significantly been reduced:

$$\sigma \leq max\{\omega(\tilde{F}_i), i = 1, ..., p, \omega(\bar{f})\}, \tag{2.7}$$

which is much smaller compared to $\omega(f)$ if we use direct reverse-mode AD method.[3]

This reduction of space requirements is the key to improve the efficiency with respect to gradients computation. Recall that in practice, we have limited fast memory. When direct reverse-mode AD algorithms require to store a computational "tape" that is excessive than our fast memory, this structured reverse-mode AD method is going to save the memory requirement significantly and keep the algorithm running in fast memory without entering into slow memory. Experiments results in Chapter 5 will show the details of the difference on applying this structured approach and direct reverse-mode AD method.

[2] In Algorithm 1, we need time proportional to $\sum_{i=1}^{p} \omega(\tilde{F}_i)$ to evaulate $y_i, i = 1, ..., p$ in Step 1. Similarly, Step 2 takes time proportional to $\omega(\bar{f})$. Lastly, in Step 3 we also need time proportional to $\sum_{i=1}^{p} \omega(\tilde{F}_i)$. Thus, our total work required to implement Algorithm 1 is proportional to $\omega(f)$ because $\sum_{i=1}^{p} \omega(\tilde{F}_i)\omega(\bar{f}) = \omega(f)$.

[3] In Algorithm 1, we need to apply reverse-mode AD in Step 2 and Step 3, which requires the storage requirement proportional to $\omega(\bar{f})$ for Step 2 and $\sigma \leq$ max $\{\omega(\tilde{F}_i), i = 1, ..., p\}$ for Step 3.

Chapter 3

Monte Carlo Process for Portfolio Pricing

With the recent advancement of computing power, Monte Carlo process is becoming extremely important in financial engineering. Particularly, it can be used to help estimate the price of a portfolio that consists a large set of financial instruments. In general, a Monte Carlo process for a financial portfolio pricing involves simulating different paths and different segments within each path. When the number of paths and number of segments used for a Monte Carlo process are relatively large, the computational cost is significantly increased. Though reverse-mode AD can be used for the the advantage on time savings on computational cost, we must overcome the massive memory requirement. In this Chapter, we will review the general procedure for pricing a financial portfolio by applying the Monte Carlo process and relate its evaluations to two special structured functions. In Chapter 4, we will then apply the structured reverse-mode AD based on these two special structured functions to tackle the problem on computing its gradients.

3.1 Evaluation of a Single Path

In order to estimate the price of the portfolio at a future date, assume we need to perform a Monte Carlo simulation that involves p paths and T segments within each path. We will start the analysis with the evaluation of a single path.

Assume we have a portfolio that consists of l financial instruments, where l is an integer. Then the a vector $S, S \in R^l$, can be used to represent the underlying assets' values at

maturity corresponding to these l instruments. Let's denote $V(S)$ as the portfolio's payoff of l financial instruments which is determined by the final simulated value vector S at maturity. Since S is simulated from the start point with some initial inputs to maturity with T segments, we can write S and V in the following formats:

$$S = g(x, Z). \tag{3.1}$$

$$V(S) = V(g(x, Z)). \tag{3.2}$$

We define g as the function that involves T segments to evaluate the value of S. $Z \in R^T$ is vector of the random innovations for T segments and x is the input vector of initial parameters, such as initial stock price, interest rate, initial volatility and so on. In Monte Carlo, one path corresponds to a specific choice of initial start values for Z.

3.2 Evaluation for p Paths

Now we can start to look at the computational steps for evaluation of the final price of the financial portfolio by using the results from all p paths. Assume the price of such a financial portfolio is P.

$$P = E(V(S)) = \int V(g(x, Z))\rho(Z)dZ. \tag{3.3}$$

This simply means that in theory our price should be equal to the expectation of the payoff and it can be written in the integral form as above. $\rho(Z)$ represents the probability density function of Z and it is not dependent on the initial parameter x. Next, we take the derivative of P with respect to x and apply basic calculus rules of interchanging the integration order:

$$\frac{dP}{dx} = \frac{d}{dx}\int \frac{dV(g(x, Z))}{dx}\rho(Z)dZ. \tag{3.4}$$

Note that, this is what we call path-wise derivatives which can be helpful for us to efficiently apply the structured Jacobian idea. We will express the details in Chapter 4. Now we will sum the analysis of this section by putting these formations back into Monte Carlo simulations. I.e, we will estimate the price P and the $\frac{dP}{dx}$ by using a Monte Carlo process with number of p paths and T segements in each path.

$$\hat{P} = \frac{1}{p}\sum_{i=1}^{p} V(g(x, Z_i)) \equiv \frac{1}{p}\sum_{i=1}^{p} \hat{P}_i. \tag{3.5}$$

$$\frac{\widehat{dP}}{dx} = \frac{1}{p}\sum_{i=1}^{p} \frac{dV(g(x, Z_i))}{dx}. \tag{3.6}$$

Where $\hat{P}$ and $\frac{\widehat{dP}}{dx}$ are estimated values for P and $\frac{dP}{dx}$ from Monte Carlo process. Note that $\hat{P}_i$ and $Z_i \in R^T$ are the estimated price and random innovation for path $i = 1, ..., p$.

3.3 Monte Carlo in a Flowchart View

Section 3.1 and 3.2 have expressed the details of Monte Carlo process on portfolio pricing. Here, we will demonstrate a flowchart view of a complete Monte Carlo process. We start with initial input x and for each path i, for $i = 1, \ldots, p$, the functions $[\tilde{g}_i(x, Z_i)]_1, \ldots, [\tilde{g}_i(x, Z_i)]_T$ are intermediate segments' evaluations of $g_i(x, Z_i)$. After T segments of evaluation on each path, the estimated portfolio prices $\hat{P}_i$, for $i = 1, \ldots, p$, are obtained. Thus the Monte Carlo process can be illustrated as:

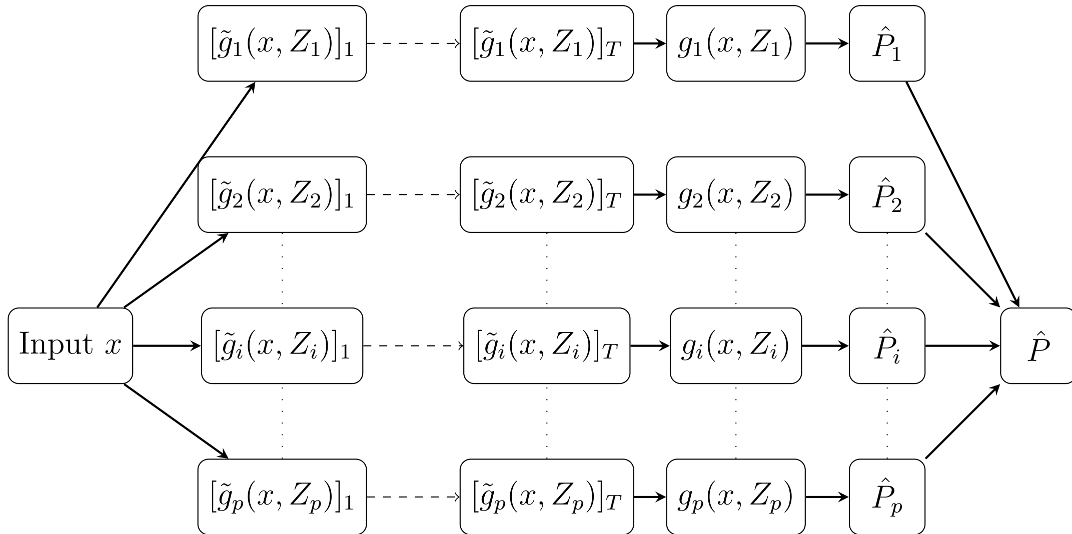

Figure 3.1: Monte Carlo Process for Portfolio Pricing

This flowchart can help us identify two speical functions that exhibit structure characteristics.

3.4 Two Structured Functions

In Section 2.3, we have expressed the general case of computing gradients by using the structured AD techinques. However, there are two special structured functions: the composite function and generalized partially separable (GPS) function. These two functions each involve an unique evaluation that is useful for us to simplify the application on the strcutured reverse-mode AD method. We will express the details of the application in Chapter 4. In the following subsections, we break down the complete evaluation of a Monte Carlo process into these two structured functions.

3.4.1 Composite Function

In general, for a differentiable mapping $F : R^n \to R^m$, we define F as a composite function if F is highly recursive, or:

$$F(x) = \bar{F}(\tilde{F}_T^E(\tilde{F}_{T-1}^E(\dots(\tilde{F}_1^E(x))\dots))), \tag{3.7}$$

where $\tilde{F}_i^E, i = 1, \dots, T$, and $\bar{F}$ are vector mappings that represent intermediate evaluations of F. Furthermore, if each intermediate function is not different from the others, i.e. $\tilde{F}_i^E$ and $\tilde{F}_j^E$ are identical mappings for $i, j = 1, \dots, T$, this composite function F is also a dynamic system. Figure (3.2) is a flowchart view of a composite function.

Composite Function

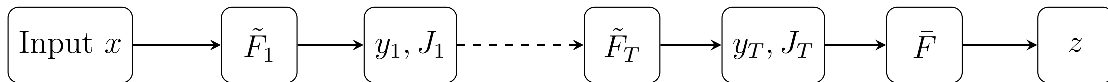

Figure 3.2: Computational steps in flowchart format for composite function. (see Figure 3 in [5])

From figure (3.2) and figure (3.1), it is clear that within each path of the Monte Carlo process, the evaluation of each path's price $\hat{P}_i$, for $i = 1, \ldots, T$, is a composite function. Within each path i, the intermediate mappings $[\tilde{g}_i(x, Z_i)]_1, \ldots, [\tilde{g}_i(x, Z_i)]_T$ are also identical to each other.

They key advantage of identifying the evaluation of T segments within each Monte Carlo's path as a composite function is that we are able to use the speical simplified calculation of derivatives. For a composite function F defined previously, the Jacobian matrix J is a matrix product of all intermediate evaluations:

$$J = \bar{J} \cdot \tilde{J}_T \cdot \tilde{J}_{T-1} \cdot \cdots \cdot \tilde{J}_1, \tag{3.8}$$

where $\bar{J}$ and $\tilde{J}_T, \ldots, \tilde{J}_1$ are corresponding Jacobians of $\bar{F}$ and $\tilde{F}_T, \ldots, \tilde{F}_1$. This simplified computation of Jacobian can be applied on our gradients computation of Monte Carlo process. We will express detailed analysis and form a complete algorithm in Chapter 4.

3.4.2 Generalized Partially Separable Function

Another special structured function is the generalized partially separable (GPS) function. In general, for a differentiable mapping $F : R^n \rightarrow R^m$, F is defined as a GPS function if:

$$\left.\begin{array}{l} \text{Solve for } y_i \ : \ \tilde{F}_i(x) - y_i = 0, \quad i = 1, \ldots, p \\ \text{Solve for } F(x) \ : \ \bar{F}(y_1, y_2, \cdots, y_p) - F(x) = 0 \end{array}\right\}, \tag{3.9}$$

where $\tilde{F}_i^E, i = 1, \ldots, p$, and $\bar{F}$ are vector mappings that represent intermediate evaluations of F. Figure (3.3) is a flowchart view of a generalized partial separable function.

From figure (3.3) and figure (3.1), it is obvious that the evaluation of the final portfolio price $\hat{P}$ based on p estimated prices $\hat{P}_1, \ldots, \hat{P}_p$ is a generalized partially separable function. The last step of mapping $\bar{F}(y_1, \ldots, y_p)$ is just a mean operation.

Though the GPS function exhibits a structure that is contrasting compared to the composite function, it creates a very sparse extended Jacobian matrix. It is useful to simplify the computation of the objective function F's Jacobian matrix J. In Chapter 4, we will express the detailed analysis of the application of this structure characteristics.

Generalized Partial Separable (GPS) Function

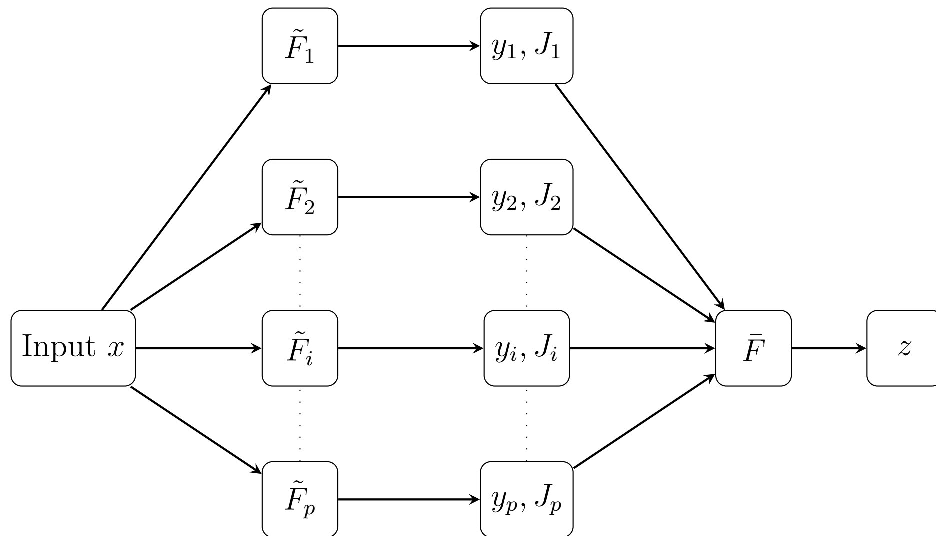

Figure 3.3: Computational steps in flowchart format for generalized partical separable function. (see Figure 4 in [5])

Chapter 4

Computing the Gradients of Monte Carlo process

This Chapter expresses the details for using structured reverse-mode AD approach to solve for the gradients of Monte Carlo process. This approach involves two sections:

• We focus on an individual path of the Monte Carlo method. For segments evaluation on each Monte Carlo path, we can use the strcuture characteristics of composite function to compute the gradients for this path, i.e. the value of $\frac{dV}{dx}$ in equation (3.6).

• We focus on the evaluation of the estimate of portfolio's price P, i.e. the mean operation of $\hat{P}_1, \ldots, \hat{P}_p$. We will use the results of $\frac{dV}{dx}$ from each path and apply the characteristics of the structure from the GPS function to compute the objective function f's gradients $\widehat{\frac{dP}{dx}}$.

4.1 Evaluation of $\frac{dV}{dx}$ Based on Composite Function's Structure

Recall that we assume for each Monte Carlo path, we take a total number of T segments. As discussed earlier, reverse-mode AD can be applied on gradients calculation due to the saving on computational time compared to forward-mode AD. However, applying direct-reverse mode AD require space to store the entire computational "tape" . In this case, the computation of the gradients for each path could become expensive in space requirement

when the number of segments T is large. In fact, cases with a large number of T are often observed in practice.

Here, we express a more efficient approach than the direct reverse-mode AD for gradient computation on each path, by taking advantage of the structure characteristics of Monte Carlos segments evaluation.

For each single path, the evaluation function for Monte Carlo can be defined as $f(x): R^n \to R^1$. The goal is to compute the gradients of f with respect to x. We can denote the intermediate variable evaluated at each segment i as $\tilde{F}_i$. We start with initial parameters x to evaluate the variable y_i at time step $i = 1$. After that, we go on to the next time step to calculate the variable based on the previous time step, until $i = T$. The final evaluation for calculating z is determined by x and all intermediate variables. Hence from the general form (2.1), we can form the following simplified computational graph for f:

$$\left.\begin{array}{l}\text{Solve for } y_1 \;:\; \tilde{F}_1(x) - y_1 = 0 \\ \text{Solve for } y_2 \;:\; \tilde{F}_2(y_1) - y_2 = 0 \\ \vdots \\ \text{Solve for } y_T \;:\; \tilde{F}_T(y_{T-1}) - y_T = 0 \\ \text{Solve for } z \;:\; \bar{f}(x, y_1, y_2, \cdots, y_T) - z = 0\end{array}\right\} . \tag{4.1}$$

This is a composite function that is highly recursive. Refer to figure (3.2) for the flowchart format of the computational steps for a composite function. For such a composite function, each intermediate variable y_i only depends on y_{i-1} for $i = 1, ..., T$.[1] Based on the extended Jacobian formation in (2.3), we can write down J^E of (4.1) as:

$$J^E = \left(\begin{array}{c|ccccc} J_x^1 & -I & & & & \\ 0 & J_{y_1}^2 & -I & & & \\ \vdots & 0 & \ddots & \ddots & & \\ \vdots & \vdots & \ddots & \ddots & \ddots & \\ 0 & 0 & \cdots & 0 & J_{y_{T-1}}^T & -I \\ \hline \nabla \bar{f}_x^T & \nabla \bar{f}_{y_1}^T & \nabla \bar{f}_{y_2}^T & \cdots & \cdots & \nabla \bar{f}_{y_T}^T \end{array}\right) = \left(\begin{array}{c|c} A & L \\ \hline \nabla \bar{f}_x^T & \nabla \bar{f}_y^T \end{array}\right) . \tag{4.2}$$

This extended Jacobian matrix demonstrates sparse features that can be used for faster gradient computation. We divide the matrix into four sub matrices $A, L, \nabla \bar{f}_x^T, \nabla \bar{f}_y^T$. It is

[1] Assume that we denote $y_0 = x$.

clear that calculations for are straightforward. From the extended Jacobian above, we can calculate the gradients of objective function f, with respect to x as:

$$\nabla f^T(x) = \nabla \bar{f}_x^T - \nabla \bar{f}_y^T L^{-1} A = \nabla \bar{f}_x^T - [\nabla \bar{f}_y^T L^{-1}]A. \tag{4.3}$$

Here, we need to compute $v^T = \nabla \bar{f}_y^T L^{-1} A$. We apply reverse mode techniques and define $w^T = (w_1^T, ..., w_T^T)$ where $L^T w = \nabla \bar{f}_y$, or write it in the following form:

$$\begin{pmatrix} -I & (J_{y_1}^2)^T & 0 & \cdots & \cdots & 0 \\ & -I & (J_{y_2}^3)^T & \ddots & & \vdots \\ & & \ddots & \ddots & \ddots & \vdots \\ & & & -I & (J_{y_{T-2}}^{T-1})^T & 0 \\ & & & & -I & (J_{y_{T-1}}^T)^T \\ & & & & & -I \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_{T-1} \\ w_T \end{pmatrix} = \begin{pmatrix} \nabla \bar{f}_{y_1} \\ \nabla \bar{f}_{y_2} \\ \nabla \bar{f}_{y_3} \\ \vdots \\ \nabla \bar{f}_{y_{T-1}} \\ \nabla \bar{f}_{y_T} \end{pmatrix} \tag{4.4}$$

This is the advantage of applying reverse mode AD algorithm since there is no need for us to actually compute the matrix L. Instead, we can directly compute the value of $(\nabla \bar{f}_y^T L^{-1})$, i.e. the value of w^T. In order to solve for w^T, we will need to solve (4.4).

However, the function $f(x) : R^n \to R^1$ is a composite function and we know that the evaluation functions at all the segments from $i = 1$ to $i = T$ are the same, i.e. it is a dynamic system. Thus we can use structured reverse-mode AD method to take advantage of this feature and make improvements on space requirement. During our evaluation, instead of saving the whole computational "tape" , we use the idea of checkpoingting and only store the intermediate variables, i.e. y_i for $i = 1, ..., T$. When we apply reverse-mode AD, at time step i, we will be able to regenerate the ith evaluation function: $\tilde{F}_i$ by the stored variables. This can be done because the ith evaluation function depends only on the previous segments variable y_{i-1}. After solving for w^T , then we can compute the gradients by the following simplified calculation:

$$\nabla f^T(x) = \nabla \bar{f}_x^T - (\nabla \bar{f}_y^T L^{-1})A = \nabla \bar{f}_x^T - w_1^T \cdot J_x^1. \tag{4.5}$$

Next, we can put everything together and express the following as the summary of our algorithm:

Algorithm 2 *Structured Gradient Computation for a Composite Function*

1. *Follow steps of $i = 1, ..., T$ only in (4.1) to evaluate values of y_i.*
2. *Follow step of $i = T + 1$ in (4.1) to evaluate z and compute $\nabla \bar{f}_x^T$ and $\nabla \bar{f}_y^T$ by using reverse-mode AD.*
3. *Use equation (4.4) and (4.5) to compute gradient:*
   (a) Set $v_i = 0, i = 1, ..., p, \nabla f = \nabla \bar{f}_x^T$,
   (b) For $j = T, T - 1, ..., 1$
   - *Solve $w_j = \nabla \bar{f}_{y_j} - v_j$;*
   - *Evaluate $\bar{F}_j(y_{j-1})$ and use w_j^T on reverse-mode AD to compute $w_j^T J_{y_{j-1}}^j$.*
   - *Set $v_j^T = v_j^T + w_j^T J_{y_i}^j$ for $i = j - 1$;*

   (c) Update $\nabla f^T \leftarrow \nabla f^T + w_1^T J_x^1$;

From the Algorithm 2, we see that we need to first evaluate the function $f(x)$ in Step 1 and Step 2, then go back wards from $i = T$ to the initial step $i = 1$. If we assume the time for evaluation of $f(x)$ is $\omega(f)$, then the computing time for using this structured reverse-mode AD method is close to $2 \cdot \omega(f)$, i.e. proportional to $\omega(f)$. Recall $f(x) : R^n \rightarrow R^1$, by using forward mode AD, we need computing time proportional to $n \cdot \omega(f)$. Hence compare to forward-mode AD, we have significant savings on computing time.

Also, our structured reverse-mode AD approach can further reduce the space requirement compared to direct reverse-mode AD. In general, from (2.7), the space requirement for using this structured approach is:

$$\sigma_s \leq max\{\omega(\tilde{F}_i), i = 1, ..., T, \omega(\bar{f})\}. \tag{4.6}$$

If we just apply direct reverse-mode for the algorithm, we will have:

$$\sigma_d = \sum_{i=1}^{T} \omega(\tilde{F}_i) + \omega(\bar{f}) = \omega(f). \tag{4.7}$$

This means that in direct reverse-mode AD, we need to store the whole computational graph, which is much larger than only storing the necessary information at checkpoints only. Due to the fact that each segment's evaluation is the same, this checkpointing idea of

only storing necessary information at each segment without saving the whole computational procedures provides us a significant advantage on space saving, normally we have:

$$\text{Direct } \sigma_d \gg \text{Structured } \sigma_s \tag{4.8}$$

Here, we also provide a simple numerical example to demonstrate how large the difference could be. Assume the time step T we use for this Monte Carlo path is 252 (a year), space requirements for each intermediate variables evaluation are: $\omega(\tilde{F}_i) = 1024, i = 1, ..., T$, and also assume $\omega(\bar{f}) = 1024$. Then applying structured reverse mode AD results in space requirement: $\sigma_s = max\{\omega(\tilde{F}_i), i = 1, ..., T\} = 1024$, which is significantly less than direct reverse mode AD: $\sigma_d = \sum_{i=1}^{T} \omega(F_i) + \omega(\bar{f}) \ = 1024 * 252 + 1024 = 259072$. This means that just for a single Monte Carlo path, the direct reverse-mode AD needs 253 times of the space requirements compared to our strcutured reverse-mode AD method!

As discussed in Chapter 2, the other advantage is that, this approach would improve on computational time. The larger the time step T is, the more efficiency we would gain. In practice, this saving on space would allow us to stay longer before we run out of fast memory. The direct reverse-mode would easily results in saturation on fast memory and starting to use slow memory for computation. Since fast memory is normally at least 10 times faster than slow memory, our structured reverse-mode AD approach would gain more advantage on efficiency when this happens. Figure (4.1) is an ideal graph to demonstrate this relationship between running time for two AD methods.[2]

Based on the graph demonstration, when $\omega(f)$ increases, structured reverse-mode AD would be much more efficient compare to direct reverse-mode AD.

[2]Note that SAD stands for structured AD and DAD stands for direct AD. In this ideal graph, the SAD's running time is more than DAD before DAD runs out of fast memory. However, the running time shoots off for both methods when fast memory runs out, but SAD stays longer before running out of fast memory

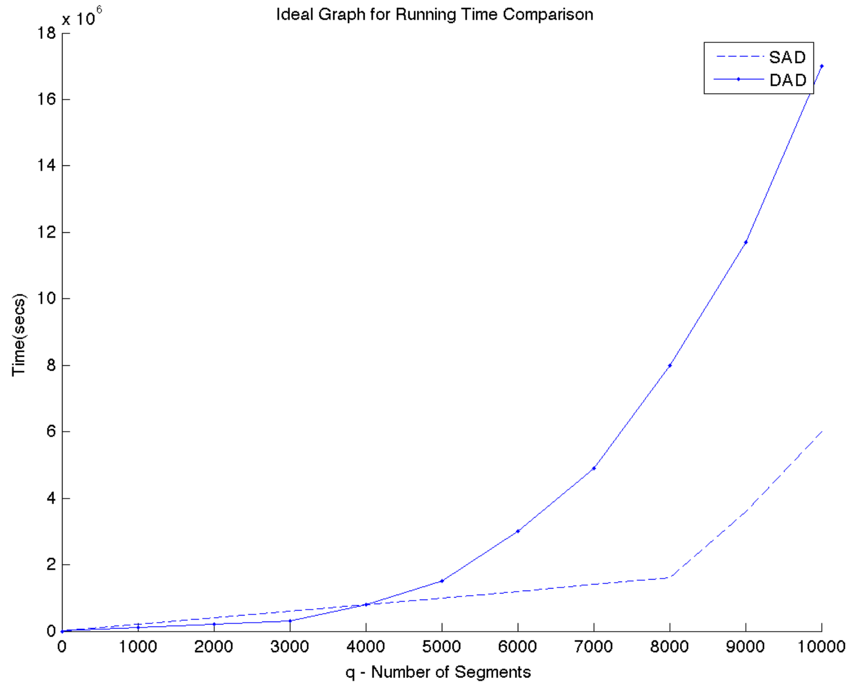

Figure 4.1: Ideal Running Time comparison between Strcutured AD and Direct AD

4.2 Evaluation of $\frac{d\widehat{P}}{dx}$ Based on GPS Function's Structure

In Monte Carlo simulation, after evaluating different paths, we need to generate the final output by performing the mean operation of the results from all paths, i.e. $\hat{P} = \frac{1}{p}\sum_{i=1}^{p}\hat{P}_i$.

From Section 4.1, we showed an efficient algorithm to compute the gradient $\frac{dV}{dx}$ for each path. In this Section, we will use the result of $\frac{dV}{dx}$ and express another similar structured reverse-mode AD algorithm for computing the gradients $\frac{d\widehat{P}}{dx}$ of the our Monte Carlo's objective function f. This would again, result in gaining on efficiency and reducing space compare to direct reverse-mode AD.

Assume that the Monte Carlos estimate computation is a mapping defined as: $f : R^n \rightarrow R^1$ and we have p different paths. We know that each path of Monte Carlo simulation is independent of other paths, and the final computation for step $p + 1$ is directly depended

on simulated values of all paths. These two features imply that each intermediate function F_i^E only depends on x and y_i for $i = 1, \ldots, p$, and the last function F_{p+1}^E depends on z and $y_1, \ldots, y_p$. Thus we can simplify the general computational graph (2.1) into the following:

$$\left.\begin{array}{lcl} \text{Solve for } y_1 & : & \tilde{F}_1(x) - y_1 = 0 \\ \text{Solve for } y_2 & : & \tilde{F}_2(x) - y_2 = 0 \\ \vdots & & \\ \text{Solve for } y_p & : & \tilde{F}_p(x) - y_p = 0 \\ \text{Solve for } z & : & \bar{f}(y_1, y_2, \cdots, y_p) - z = 0 \end{array}\right\} . \tag{4.9}$$

Refer to figure (3.3), it is clear that this evalaution is a generalized partially separable function. Next, we will explore the sparse features in its extended Jacobian matrix. From the general Jacobian matrix expression (2.3), the corresponding J^E for the computational graph above is:

$$J^E = \left(\begin{array}{c|ccccc} J_x^1 & -I & & & & \\ J_x^2 & 0 & -I & & & \\ \vdots & 0 & \ddots & \ddots & & \\ \vdots & \vdots & \ddots & \ddots & \ddots & \\ J_x^p & 0 & \cdots & 0 & 0 & -I \\ \hline 0 & \nabla \bar{f}_{y_1}^T & \nabla \bar{f}_{y_2}^T & \cdots & \cdots & \nabla \bar{f}_{y_p}^T \end{array}\right) = \left(\begin{array}{c|c} A & L \\ \hline \nabla \bar{f}_x^T & \nabla \bar{f}_y^T \end{array}\right) . \tag{4.10}$$

Similar to Section 4.1, the extended Jaciobian matrix here indicates sparse features as well. Due to the characteristics of Jacobian matrix of a GPS function, we have submatrix L being a negative identity matrix. Hence, by applying Schur-complement computation, we can obtain the gradient by:

$$\nabla f^T(x) = \nabla \bar{f}_x^T - (\nabla \bar{f}_y^T L^{-1})A = -(\nabla \bar{f}_y^T L^{-1})A \tag{4.11}$$

It is clear that the calculation of matrix $\nabla \bar{f}_y^T$ is straightforward by using the revese-mode AD after evaluating z. Also, we need to compute $v^T = \nabla \bar{f}_y^T L^{-1} A$. Similar to Section 4.1, we apply reverse-mode techniques and define $w^T = (w_1^T, ..., w_p^T)$ where $L^T w = \nabla \bar{f}_y$, or write it in the following form:

$$\begin{pmatrix} -I & 0 & 0 & \dots & \dots & 0 \\ & -I & 0 & \ddots & & \vdots \\ & & \ddots & \ddots & \ddots & \vdots \\ & & & -I & 0 & 0 \\ & & & & -I & 0 \\ & & & & & -I \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_{p-1} \\ w_p \end{pmatrix} = \begin{pmatrix} \nabla \bar{f}_{y_1} \\ \nabla \bar{f}_{y_2} \\ \nabla \bar{f}_{y_3} \\ \vdots \\ \nabla \bar{f}_{y_{p-1}} \\ \nabla \bar{f}_{y_p} \end{pmatrix} \tag{4.12}$$

The advantage of dealing with a GPS function is that we do not need to compute the matrix L at all. The value of $(\nabla \bar{f}_y^T L^{-1})$, i.e. the value of w^T can be solved directly from (4.12) in the following way:

$$w^T = -\nabla \bar{f}_y^T \tag{4.13}$$

Thus our gradient compuation from (4.11) is further simplified as:

$$\nabla f^T(x) = -(\nabla \bar{f}_y^T L^{-1})A = -w^T A = \nabla \bar{f}_{y_1}^T \cdot J_x^1 + \nabla \bar{f}_{y_1}^T \cdot J_x^1 + \cdots + \nabla \bar{f}_{y_1}^T \cdot J_x^p \tag{4.14}$$

Note that the values of J_x^i, for $i = 1, \dots, p$ are derivatives of the $\tilde{F}_i$ with respect to initial input x, this is exactly the results we get by applying Algorithm 2 on each path i. Though we can use direct remove-mode AD to calculate the above equation, the space requirement for computing $J_x^i, \dots, J_x^p$ could be very expensive with direct reverse-mode AD when the number of segments T for each path is large. Thus we will use Algorithm 2, the structured reverse-mode AD method to compute $J_x^i, \dots, J_x^p$.

Next, we can put all these steps in Section 4.2 together as Algorithm 3 for the gradients computation:

Algorithm 3 *Structured Gradient Computation for a GPS Function*

1. *Follow steps of $i = 1, ..., p$ only in (4.9) to evaluate values of y_i.*
2. *Follow step of $i = p + 1$ in (4.9) to evaluate z and compute $w_i^T = \nabla \bar{f}_{y_i}^T$, for $i = 1, \dots, p$ by using reverse-mode AD.*
3. *Use equation (4.12) and (4.13) to compute gradient:*
   (a) *Set $v_i = 0, i = 1, ..., p, \nabla f = 0$,*
   (b) *For $j = p, p - 1, ..., 1$*
      - *Evaluate $\bar{F}_j(x, y_j)$ and use Algorithm 2 to compute $v_j^T = w_j^T \cdot J_x^j$*

   (c) *Update $\nabla f^T \leftarrow v_1^T + v_2^T + \cdots + v_p^T$;*

Note that in Step 3(b) of Algorithm 3, we can use direct reverse-mode to compute v_j^T, for $j = p, \ldots, 1$, if T is relatively small or we do not worry about the space issues for computing $J_x^1, \ldots, J_x^p$. We will demonstrate detailed comparisons between using Algorithm 3 and this modified version in Chapter 5.

Next, lets take a look at the saving on time and space by applying Algorithm 3.

Assume the time to evaluate the function $f(x)$ is $\omega(f)$, then applying reverse-mode AD will result in the same time needed for the functions evaluation, which is $\omega(f)$. This is the same for direct reverse-mode AD and our structured reverse-mode AD.

Similar as Section 4.1, from (2.7), our structured approach in Algorithm 3 will result in required space as:

$$\sigma_s \leq max\{\omega(\tilde{F}_i), i = 1, ..., p, \omega(\bar{f})\}. \tag{4.15}$$

Compare to applying direct reverse-mode, we will have:

$$\sigma_d = \sum_{i=1}^{p} \omega(\tilde{F}_i) + \omega(\bar{f}) = \omega(f). \tag{4.16}$$

The sum for the space requirement of the intermediate steps is always larger than the max of them. When p gets larger, the max value will be much smaller than the sum. Hence, our structured AD approach is significantly reducing the amount of memory needed for storing the computational tape, especially for large p values, i.e.:

$$\text{Direct } \sigma_d \gg \text{Structured } \sigma_s \tag{4.17}$$

4.3 Minimizing The Memory Requirement

The application of Algorithm 2 in Step 3(b) of Algorithm 3 can help us put the space analysis of (4.6) into equation (4.15) and further reduces the space requirement in equation (4.15). Recall that our Monte Carlo process for a portfolio pricing is defined as a differentiable mapping $f : R^n \to R^1$. It involves p paths and T segments within each path. Then our full breakdown of the space requirement (4.15) is:

$$\sigma_s \leq max\{\omega(\tilde{F_i}), i = 1, ..., p, \omega(\bar{f})\}. \tag{4.18}$$

$$\omega(\tilde{F_i}) \leq max\{\omega(\tilde{F_{ij}}), j = 1, ..., T, \omega(\bar{f_i})\}, i = 1, ..., p. \tag{4.19}$$

Here, $\omega(\tilde{F_{ij}}), j = 1, ..., T$, and $\omega(\bar{f_i})$ are corresponding intermediate segments' evaluations for each path i. Putting (4.18) and (4.19) together, compare to direct reverse-mode AD method, this approach would reduce our space requirement in two dimensions, first on the level of GPS function evaluations on p paths and secondly on the compostie function evaluations on T segments. In principle, the ratio of space requirement between applying direct reverse-mode AD and this structured reverse-mode AD can be as large as $(p * T) : 1$, i.e. the larger p and T are, the more space we can save..

In practice, we are more confident to apply this structured reverse-mode AD method on computing gradients for a Monte Carlo process because we can keep the algorithm running on fast memory. In Chapter 5, we will show some numerical results on the gain on efficiency for applying this approach compared to direct reverse-mode AD.

Chapter 5

Numerical Results

In this section, we express the numerical comparisons for using different AD methods on gradients computation of a Monte Carlo process. In order to keep the experiments results consistent, all experiments are performed on a computer with RAM of 4G, AMD CPU 2.90Ghz, 512GB hard drive. We use Matlab 2015b under Windows 7 Professional and the AD toolbox ADMAT 2.0 [6].

5.1 Experiment Design

The experiments are based on the Heston Model for option pricing. This simple model includes evolutions on both the volatility and the price of an underlying asset, denoted correspondingly as v and S. The stochastic process can be written in the discretized form as:

$$\Delta S_i^j = \mu S_{i-1}^j \Delta t + \sqrt{v_{i-1}^j} S_{i-1}^j \Delta W_{i-1}^{S^j} \tag{5.1}$$

$$\Delta v_i^j = \kappa(\theta - v_{i-1}^j)\Delta t + \zeta \sqrt{v_{i-1}^j} \Delta W_{i-1}^{v^j} \tag{5.2}$$

Here, μ is the risk free rate and θ is the average variance in the long run. Equation (5.2) has the property that the volatility v will revert to the mean variance θ with a rate κ, and ζ is the volatility of volatility v. Lastly, the $W_{i-1}^{S^j}$ and $W_{i-1}^{v^j}$ are two Wiener processes

with correlation ρ. For simplicity, we choose European call option as our payoff function, i.e. $payoff = max(S - K, 0)$.

The initial parameters we chose as the input are included in the following table[1]:

Initial Inputs for Heston Model		
S_0	abs(randn(NAssets,1))	a vector of positive random initial stock prices
μ	0.005	risk free rate
v_0	0.05	a vector of intial volatility
θ	0.05	mean-reverting volatility
κ	0.1	rate of reverting
ζ	0.025	volatility of volatility
K	mean(S_0)	strike price as the average of all S_0 prices
ρ	0.5	correlation of Wiener Process

Table 5.1: Initial Inputs for Heston Model

Other than these shown above, we also have three variables: NAssets (number of assets in the portfolio), NPaths (number of simulations for Monte Carlo process), and NSegments (number of timesteps in each simulation). These are the chosen accordingly to set the complexity of gradients computation for the Monte Carlo process. The computation of reverse-mode AD from ADMAT 2.0 requires to store a computational tape. When the RAM of 4GB is running out on the computer, the computational tape has to be stored in the hard drive. The usage of hard drive in this case is the cause for an unefficiency computation. The purpose of our experiments is to show how this situation can affect the running time and make comparisons between different AD methods. We will express two comparison tests in the following sections.

5.2 Test I - SAD v.s DAD

First is the comparison between structured reverse-mode AD (denoted as SAD) method versus the direct reverse-mode AD (denoted as DAD) method where we keep the NSegments as 252 days (1 year) fixed and change the number of NAssets and NPaths. This way,

[1] In the initial inputs table, 'abs(randn(NAssets,1))' is a Matlab command to generate a vector of size $NAssets * 1$ with positive random numbers, 'mean(S_0)' is a Matlab command to take the mean of the vector S_0.

the computational "tape" would need much more memory requirement for DAD method than the SAD method when NAssets and NPaths are large. As discussed in Chapter 4, the application of structured reverse-mode AD method on the GPS and composite function structures, i.e. the evaluations on paths and segments, of the Monte Carlo process can significantly reduce our memory requirement. The running time should not be much different for two methods when we have enough fast memory to store the computational tape. When the fast memory is saturated for the DAD method, the running time would shoot up rapidly. In contrast, when this happens, the SAD method can still perform within the fast memory and keep its running time much lower than the DAD method. Hence, SAD would show a great advantage on time and space efficiency when the algorithm runs out of fast memory.

NAssets	Npaths	Gradient Size	SAD		AD		Ratio – AD/SAD	
			Memory(KB)	Time(S)	Memory(KB)	Time(S)	Memory	Time
100	50	100	11.43	78.48	118985.45	14.31	10408.43	0.18
100	100	100	11.43	157.70	237963.29	24.24	20816.20	0.15
100	200	100	11.43	314.66	475918.95	45.11	41631.73	0.14
100	400	100	11.43	630.06	951830.28	85.76	83262.79	0.14
100	800	100	11.43	1259.39	1903652.93	253.49	166524.91	0.20
100	1600	100	11.43	2519.50	3807298.25	2170.52	333049.15	0.86
100	3200	100	11.43	5038.28	5711050.25	17663.67	499582.73	3.51
50	400	50	8.31	604.86	676029.11	85.76	81384.18	0.14
100	400	100	11.43	630.06	951830.28	94.31	83262.79	0.15
200	400	200	17.68	663.45	1503432.62	129.04	85027.89	0.19
400	400	400	30.18	689.84	2606637.31	896.65	86365.00	1.30
800	400	800	55.18	736.07	4813046.68	3214.89	87221.88	4.37

Table 5.2: Comparison of Memory and Time between Strcuture AD and Direct AD, NSegments=252

In fact, the results we got are in agreement with our analysis. From the summary in table (5.2)[2], it is clear that when the memory requirement is much smaller than our RAMs size, DAD outperforms SAD in running time. This is due to the overhead cost of applying SAD method. The ratio is in the range of 0.14 to 0.20, which means DAD is on average 5 or 6 times faster than SAD. However, we can see that when the number of NAssets or NPaths become large enough, the relatively large size of memory demand for DAD slows

[2]Ratio in the tables represents the memory/time of DAD divided by memory/time of SAD.

down its performance and significantly drives up the running time. In the meantime, the running time for SAD keeps increasing in a linear level. In figure (5.1) we have the plot summaries to express the comparisons in memory and time.

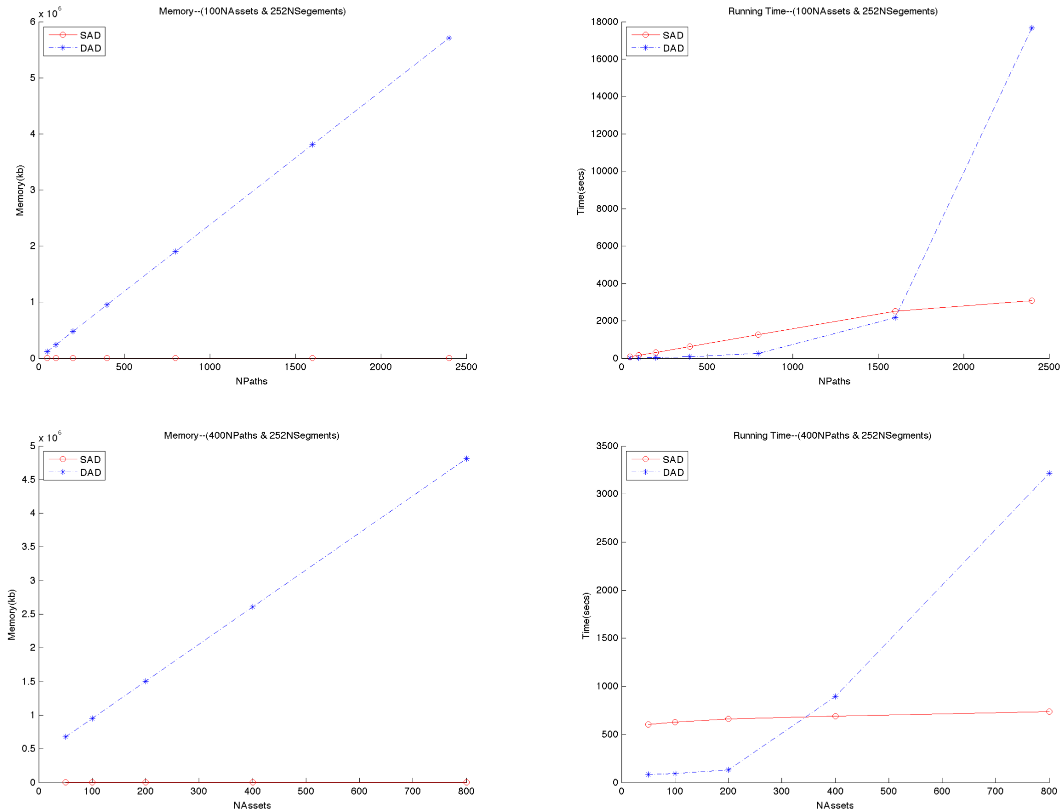

Figure 5.1: Comparions of Memory and Time for SAD and DAD: Topleft and Topright graphs are the memory and time comparisons for NAssets=100 and NSegments=252 with different choices of NPaths; bottomleft and bottomright graphs are the memory and time comparisons for NPaths=400 and NSegments=252 with different choices of NAssets.

These are experiments performed on a personal computer. In practice, we often are faced with a much larger number of simulations on a Monte Carlo process, which means the computational tape would be much larger than the results shown here. Hence, we can expect the time ratio to grow even much faster and expect the advantages of using SAD become much more significant in reality.

5.3 Test II - SAD v.s SAD-PathOnly

On the other hand, we conducted a second experiment for a comparison between two SAD methods. For simplicity, we make the following definitions. First, we denote SAD as the structured AD method we described earlier in this paper that applies structured AD idea on the evaluation of both paths and segments. Secondly, we denote SAD-PathOnly method as the structured AD method that only applies the second part of section four, i.e. SAD-PathOnly method only uses the structured AD idea on paths and applies direct AD on each paths evaluation. In this way, the SAD method applies the structured advantages on both the GPS and composite functions, i.e. the evaluations on paths and segments, while the SAD-PathOnly method only applies on the GPS functions. The main reason for conducting this comparison is that we have overhead cost in using SAD on both the evaluations of segments and paths. If the SAD-PathOnly method is efficient enough for the purpose of space saving, we might not need to apply the structured advantages on the segments level. It is only efficient to do so when we are faced a Monte Carlo process performed with a large number of segments. In that case, the computational tape needs to be reduced by using the full SAD method in order to keep the algorithm running within fast memory.

In theory, the SADs running time would outperform SAD-PathOnly method when the computational tape is relatively much larger for the latter method. In order to achieve this difference, we designed the experiment in the following way. The NAssets are fixed at 10,000 and NPaths are fixed at 10.[3] For different tests, we gradually increase the NSegments so that the computational tape will grow for the SAD-PathOnly method but not for the SAD method.

The table (5.3)[4] shows the results of this experiment. As expected, the memory for the SAD stays the same but increases for SAD-PathOnly method when we the NSegments get large. Again, due to some overhead costs, SAD underperforms SAD-PathOnly method when memory requirement is low. On average, SAD-PathOnly takes only 0.51 to 0.79 of the running time of SAD method if fast memory is not an issue. However, the running time for SAD-PathOnly method shoots up rapidly when we run out of fast memory while SAD method still keeps increasing in a relatively slow linear level. Figure (5.2) shows the details of the plot summary on space and time efficiency of these two methods.

[3]though it is be too small to be used in practice, we use it because of its suitability for the SAD-PathOnly method can run out of fast memory on the testing machine

[4]Ratio in the tables represents the memory/time of SAD-PathOnly divided by memory/time of SAD.

NSegments	Gradient Size	SSAD		SAD		Ratio – AD/SAD	
		Memory(KB)	Time(S)	Memory(KB)	Time(S)	Memory	Time
10	10000	630.18	1.72	5587.71	0.99	8.87	0.58
20	10000	630.18	2.18	11096.06	1.11	17.61	0.51
50	10000	630.18	5.54	27621.14	3.72	43.83	0.67
100	10000	630.18	10.62	55162.94	8.44	87.53	0.79
1000	10000	630.18	108.52	558460.77	344.10	886.19	3.17
2000	10000	630.18	214.71	1109316.55	1139.22	1760.31	5.31
3000	10000	630.18	323.81	1658271.07	2601.50	2631.42	8.03
4000	10000	630.18	431.75	2203423.10	5458.91	3496.49	12.64

Table 5.3: Comparison of Memory and Time between SAD and SAD-PathOnly, NAssets=10,000, NPaths=10

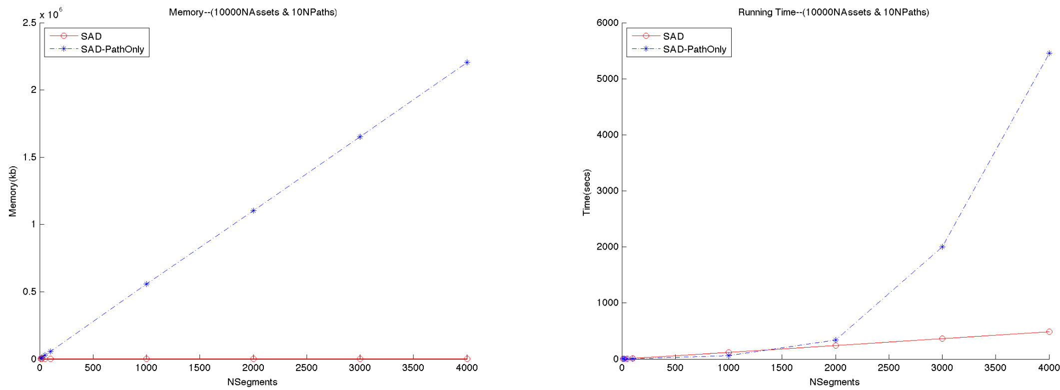

Figure 5.2: Comparions of Memory and Time for SAD and SAD-PathOnly: Left and Right graphs are the memory and time comparisons for NAssets=10,000 and NPaths=10 with different choices of NSegments.

Chapter 6

Conclusion

In a variety of fields that involve scientific computing, using Automatic Differentiation methods to efficiently solve for the derivatives can be a powerful tool. Reverse-mode AD in particular can be used if the goal is to solve for the gradients of a scalar mapping. The gradients can be solved in time proportional to that required to evaluate the objective function. In practice, this advantage has attracted lots of interests in using reverse-mode AD method. However, using reverse-mode AD requires to store the computational "tape" that can be significantly massive and cause negative impacts on the running time. For a Monte Carlo process, when the intermediate evaluations of the objective function become complex, eventually the memory requirement would be more than the fast memory's capacity. In this case, the access of second or slow memory can slow down the efficiency of reverse-mode AD methods dramatically.

With the idea of "checkpointing" , we are able to store only necessary information at certain states/checkpoints. The key of applying this idea is to recognize the characteristics of structure for the objective function. Through the analysis of the intermediate evaluations of a Monte Carlo process, we have expressed two important structures, the GPS function and the composite function. The simplified structured reverse-mode AD approach proposed can be applied based on these two structures. As a result, it reduces our computational "tape" compared to direct reverse-mode AD. The saving on memory requirement allows the algorithm to keep running on fast memory only and ensures the efficiency advantage for using reverse-mode AD method.

In practice, a Monte Carlo process can be more complex than the one considered in this paper. For example, a second level of Monte Carlo process might be involved within the original Monte Carlo process, often denoted as a nested Monte Carlo process. For such

a nested problem, it is still feasible to use the logic proposed in this paper to analyze the intermediate evaluations of the objective function. The special structures of GPS function and composite function can be applied again to further reduce the memory requirement. However, as illustrated in Chapter 5, due to the existence of the overhead cost for using structured reverse-mode AD, the best situation for using structured reverse-mode AD is when we are facing issues on running out of fast memory.

The numerical results expressed in this paper are performed based on a relatively simple model and do not take large numbers of paths or segments, i.e. the number of paths can be as large as 10,000 to 100,000. In practice, for complex or nested Monte Carlo processes that involves massive intermediate operations, it is normal to have an excessive memory requirement that is larger than the RAM size of a standard computer. Thus it is reasonable to believe that the application of structure reverse-mode AD can be widely used in Monte Carlo processes within the finance industry.

References

[1] M. Bartholomew-Biggs, S. Brown, B. Christianson, and L. Dixon. *Automatic differentiation of Algorithm*, volume 124. J. Comput. App. Math, 2000.

[2] D.G. Cacuci, C.F. Weber, E.M. Oblow, and J.H. Marable. *Sensitivity theory for general systems of nonlinear equations*, volume 88. Nuclear Sci. Engrg., 1980, 88110.

[3] Z. Chen and P. Glasserman. Fast pricing of basket default swaps. *Operations Research*, 56:286–303, 2008.

[4] Z. Chen and P. Glasserman. Sensitivity estimates for portfolio credit derivatives using monte carlo. *Finance and Stochastics*, 12:507–540, 2008.

[5] T. F. Coleman and A. Verma. *The efficient computation of sparse Jacobian matrices using automatic differentiation*, volume 19. SIAM J. Sci. Comput., 1998, 1210-1233.

[6] T.F. Coleman and W. Xu. *Automatic Differentiation in MATLAB using ADMAT (with Applications), SIAM*, 2016.

[7] M. Giles and P. Glasserman. Computation methods: Smoking adjoints: fast monte carlo greeks. *Risk*, 19:88–92, 2006.

[8] A. Griewank. *Some bounds on the complexity gradients.* Complexity in Nonlinear Optimization, P. Pardalos, Ed. World Scientific Publishing Co., Inc., River Edge, NJ, 1993.

[9] A. Griewank and G.F. Corliss. *Automatic Differetiation of Algorithms: Theory, Implementation and Applications.* 1991.

[10] A. Griewank and A. Walther. *Algorithm 799: Revolve: An Implementation of Checkpointing for the Reverse or Adjoint Mode of Computational Differentiation, ACM Trans*, volume 2. On Math. Soft., 2000, 19-45.

[11] A. Griewank and A. Walther. Evaluating derivatives: Principles, and techniques of algorithmic differentiation 2nd ed. *SIAM, Philadelphia, PA*, 2005.

[12] C. Kaebe, J. H. Maruhn, and E. W. Sachs. Adjoint based monte carlo calibration of financial market models. *Journal of Finance and Stochastics*, 13:351 379, 2009.

[13] S. Linnainmaa. *Taylor expansion of the accumulated rounding error*, volume 16. BIT, 1976, 146160.

[14] B. Speelpenning. *Compiling fast partial derivatives of functions given by algorithms.* PhD thesis, Department of Computer Science, University of Illionios at Urbana-Champain, Unrbana-Champaign, Ill., January 1980.

[15] W. Xu, X. Chen, and T. F. Coleman. The efficient application of automatic differentiation for computing gradients in financial applications. *Journal of Computational Finance*, 19(3), January 2016.

[16] W. Xu, S. Embaye, and T. F Coleman. Efficient computation of derivatives, and newton steps, for minimization of structured functions using automatic differentiation. Technical report, 2016.