Optimization with Uncertain Parameters: A Risk Protection Approach Using Automatic Differentiation

by

Hanxing Zhang

An essay
presented to the University of Waterloo
in fulfillment of the
essay requirement for the degree of
Master of Mathematics
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2009

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

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

Abstract

In this essay we propose a risk protection approach for minimizing a nonlinear unconstrained problem with uncertain parameters. By minimizing the average of second order approximations of the original objective function at p sample values of the parameters, we avoid unsatisfactory results due to unfortunate specification of the unknown parameters. We demonstrate that the computational cost of the risk protection approach is acceptable: we give both a theoretical analysis and provide results of computational experiments. In order to obtain accurate derivatives and reduce the computational cost in minimization, the reverse mode of automatic differentiation in ADMAT 2.0 is adopted to calculate gradients and Hessians. We also use ADMAT 2.0 to investigate some interesting sensitivity applications in computational finance such as "the Greeks", project and real option evaluation, and CVaR minimization.

Acknowledgements

First and foremost I owe my deepest gratitude to my supervisor Dr. Thomas Coleman, who has supported me throughout my Master's study with his knowledge, guidance, and patience. I attribute the level of my Master's degree to his encouragement and effort. Without him, this essay would not have been completed or written. One simply could not wish for a better or friendlier supervisor.

Secondly, I would like to show my gratitude to the readers of my essay, Dr. Stephen Vavasis and Dr. Michael Best. Thanks a lot for reading my essay so carefully and providing me with various useful suggestions that helped me make the essay better. The courses I took from Steve and Mike during my Master's study also offered me inspiration in my essay work, which I really appreciate.

Moreover, I am grateful to Dr. Wei Xu for all the time and effort he has spent answering the questions and solving the problems I encountered when using ADMAT 2.0.

I also would like to thank Jane Russwurm, my writing tutor. Her proofreading made this essay more accurate and attractive. In addition, her patience and encouragement meant so much to me.

It is a pleasure to thank Dr. Mun and Mr. Wong as well, who offered me a great eight- month internship opportunity to learn more about advanced statistical models and financial models during my master's study.

I greatly appreciate the support and help I received from my friends through my Master's study at University of Waterloo as well.

I cannot end without thanking my family, on whose constant encouragement and love I have relied throughout my life.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this essay we investigate unconstrained nonlinear optimization with uncertain parameters. Because the parameters are unknown, arbitrary or random settings of the parameters could lead to very undesirable results. To overcome this problem, we propose a risk protection approach.

Mathematically, let $f(x, \mu)$ be a scalar-valued function of n independent variables x and m parameters μ. In general, assume n is much greater than m. Assume that the dependence of f on x and μ is twice continuously differentiable.

Suppose μ is a random variable with a continuous distribution. Let $\mu_1, \cdots, \mu_p$ be p sample values of μ chosen randomly from its distribution. Then the expected value of f at x, $E(f_\mu(x))$, can be approximated by the average of function value at different sample values of μ

$$g(x) \equiv \frac{1}{p} \sum_{i=1}^{p} f(x, \mu_i) \tag{1.1}$$

Minimizing g with respect to x is an approach to finding the minimizer of the expected value function of f. Gradients are needed in many algorithms for nonlinear programming. To minimize g, we have to evaluate the gradient of $f(x, \mu)$ at each μ_i, $i = 1, \cdots, p$. However, p can be fairly large to ensure a good approximation to the expected value; thus, the cost to minimize g can be expensive.

The risk protection approach uses second order approximation of $f(x, \mu)$ at $\bar{\mu}$, where $\bar{\mu}$ is the mean of the sample values $\mu_1, \cdots, \mu_p$, and then takes the average of the approximation values at different sample values of μ. Define

$$\tilde{f}(x, \mu_i) \equiv \tilde{f}_i(x) = f(x, \bar{\mu}) + \nabla_\mu f(x, \bar{\mu})^T(\mu_i - \bar{\mu}) + \frac{1}{2}(\mu_i - \bar{\mu})^T \nabla_\mu^2 f(x, \bar{\mu})(\mu_i - \bar{\mu}) \tag{1.2}$$

and

$$\tilde{g}(x) \equiv \frac{1}{p} \sum_{i=1}^{p} \tilde{f}_i(x) \tag{1.3}$$

To evaluate $\tilde{g}(x)$, we only need to evaluate $f(x, \bar{\mu})$, $\nabla_\mu f(x, \bar{\mu})$ and $\nabla^2_\mu f(x, \bar{\mu})$ once and the rest of work consists of matrix multiplications. In the next section, we will analyse the cost of evaluating the function values, gradients, and Hessian of $g(x)$ and $\tilde{g}(x)$ carefully. We will show that evaluating the function value, gradients, and Hessians of $\tilde{g}(x)$, compared to $g(x)$, is less costly.

To compute exact values of derivatives and solve the minimization problems with acceptable computational cost, the automatic differentiation technique is used in this essay to calculate gradients and Hessians. More specifically, ADMAT 2.0—an Automatic Differentiation Toolbox for MATLAB developed by Dr. Coleman and Dr. Xu [4, 5, 7] is used. It is based on previous work by Dr. Coleman and Dr. Verma [3]. Both forward and reverse mode of automatic differentiation are included in ADMAT 2.0. The reverse mode of automatic differentiation is used to compute the gradient in the following analysis and experiment because the forward mode is more expensive. Not surprisingly, the reverse mode uses more memory because all the intermediate variables are saved. The automatic differentiation technique, forward mode and reverse mode, will be introduced in Appendix.

One reason that we are interested in this risk protection approach is that there are many sensitivity problems in computational finance. One example is the sensitivity of financial derivatives such as options to a change in underlying parameters on which the value of an instrument is dependent, sometimes known as "the Greeks". Matlab example functions include: 'blsprice', 'binprice', 'blkprice'. Thus, we can use ADMAT 2.0 to evaluate "the Greeks" very conveniently.

The sensitivity problem in project valuation is also interesting. The discounted cash flow analysis is a traditional way to price a project. However, it is subjective to choose a proper discount rate because the discount rate depends on the type of risk involved. Another way to valuate a project is using real option analysis. A real option gives the right but not the obligation to undertake some business decisions (such as choosing, contracting, or abandoning, a capital investment). The binomial tree method is an approach to price a real option and it takes the volatility as an input parameter. Estimation of the volatility of a real option is believed to be the greatest challenge in practice. Therefore, it will be interesting to investigate these sensitivity problems in project valuation.

Another application of approximating the expected value, $E(f_\mu(x))$, is CVaR minimization. In CVaR minimization, generally we assign an expected return to each asset. As a result, the number of parameters is equal to the number of variables. In our problem, we

are mostly interested in $n \gg m$. CVaR is a constrained problem while our original problem is an unconstrained one. We will explain how our risk protection approach can be used in CVaR minimization problem in Section 3.

The essay is organized in the following way. Section 2 introduces the motivation for the risk protection approach and shows the advantages of using the risk protection approach by both theoretic results and computational results. Section 3 shows the applications of ADMAT 2.0 and the risk protection approach in some problems in computational finance. Section 4 is the conclusion. We put introduction to automatic differentiation technique and BFGS method and trust-region method algorithms in Appendix so that the essay will be self-contained.

Chapter 2

Risk Protection Approach

2.1 Motivation of Risk Protection Approach

Let $f(x, \mu)$ be a scalar-valued function of n variables x and m parameters μ. Assume that the dependence of f on x and μ is twice continuously differentiable. Suppose μ is a random variable that has a continuous distribution. Suppose we wish to minimize the expected function of f—$E[f(x, \mu)]$, where the expectation is over the random choice of μ and the minimization is over x.

Instead of calculating the expectation by integration, i.e.,

$$E[f(x, \mu)] = \int_{-\infty}^{\infty} f(x, \mu) g(\mu) d\mu$$

where $g(\mu)$ is the probability density function for μ. we can select numerous sample values from the distribution of μ and approximate $E[f(x, \mu)]$ by

$$E[f(x, \mu)] = \frac{1}{p} \sum_{i=1}^{p} f(x, \mu_i)$$

where $\mu_1, \cdots, \mu_p$ are sample values of μ. It is obvious that the approximation will be accurate when p goes to infinity, i.e.,

$$\forall \epsilon > 0, \quad \Pr(|E[f(x, \mu)] - \frac{1}{p} \sum_{i=1}^{p} f(x, \mu_i)| > \epsilon) \to 0 \quad as \quad p \to \infty$$

Define $g(x)$ as

$$g(x) \equiv \frac{1}{p} \sum_{i=1}^{p} f(x, \mu_i) \tag{2.1}$$

Minimizing $g(x)$ is an approach to finding the minimizer of $E[f(x,\mu)]$.

The problem with minimizing $g(x)$ is that the cost of evaluating the gradients of $g(x)$ is expensive when p is large because the gradients of $f(x,\mu_i)$ will be evaluated at different μ_i. The risk protection approach we propose tries to minimize another function $\tilde{g}(x)$. Let $\bar{\mu}$ be the mean of the sample values $\mu_1, \cdots, \mu_p$. We can estimate $f(x,\mu_i)$ using a Taylor series around $\bar{\mu}$. Define

$$\tilde{f}(x,\mu_i) \equiv \tilde{f}_i(x) = f(x,\bar{\mu}) + \nabla_\mu f(x,\bar{\mu})^T(\mu_i - \bar{\mu}) + \frac{1}{2}(\mu_i - \bar{\mu})^T \nabla_\mu^2 f(x,\bar{\mu})(\mu_i - \bar{\mu}) \quad (2.2)$$

and

$$\tilde{g}(x) \equiv \frac{1}{p}\sum_{i=1}^{p} \tilde{f}_i(x) \quad (2.3)$$

When minimizing $\tilde{g}(x)$, the cost to evaluate the gradients and Hessians of $\tilde{g}(x)$ will be significantly less than the cost of $g(x)$. The detailed analysis can be found in the next section. Moreover, $\tilde{g}(x)$ is a satisfiable approximation to $E[f(x,\mu)]$ as well as $g(x)$.

The following examples show that when p is fairly large, (2.1) and (2.3) are both suitable approximations for the expected value $E[f(x,\mu)]$. In our examples, μ has a normal distribution or uniform distribution. However, μ can be assumed to follow any distribution as long as the sample values of μ are chosen randomly according to the same distribution.

Suppose μ is uniformly distributed in $[0,3]$, $f(x,\mu) = x\mu^3$. The expected value $E[f(x,\mu)]$ at $x = 2$ is $\int_0^3 2\mu^3 \frac{1}{3} d\mu = 13.5$. The approximations obtained by (2.1) are shown in Table 2.1 .

Table 2.1: Approximations to the Expected Value, Example 1

	$p = 100$	$p = 1000$	$p = 10000$	$p = 100000$
$g(x)$	14.9356	13.6035	13.8441	13.5350
$\tilde{g}(x)$	14.1250	13.1950	13.8455	13.5449

Suppose μ is uniformly distributed in $[0,10]$, $f(x,\mu) = x\mu^3$. The expected value $E[f(x,\mu)]$ at $x = 2$ is $\int_0^{10} 2\mu^3 \frac{1}{10} d\mu = 500$. The approximations by (2.1) are shown in Table 2.2.

Table 2.2: Approximations to the Expected Value, Example 2

	$p = 100$	$p = 1000$	$p = 10000$	$p = 100000$
$g(x)$	440.8820	510.3992	506.7020	501.5022
$\tilde{g}(x)$	529.8732	480.0121	510.4116	501.8660

Suppose μ is normally distributed in $N(2,1)$ and $f(x,\mu) = x\mu^3$. The expected value $E[f(x,\mu)]$ at $x = 2$ is $\int_{-\infty}^{\infty} 2\mu^3 \frac{1}{\sqrt{2\pi}} \exp(-\frac{(x-2)^2}{2}) d\mu = 28$. The approximations by (2.1) are shown in Table 2.3.

Table 2.3: Approximations to the Expected Value, Example 3

	$p = 100$	$p = 1000$	$p = 10000$	$p = 100000$
$g(x)$	23.5147	27.6601	27.2750	27.8768
$\tilde{g}(x)$	31.1890	27.5708	27.9571	27.9557

Suppose μ is normally distributed in $N(2,2)$, $f(x,\mu) = x\mu^3$. The expected value $E[f(x,\mu)]$ at $x = 2$ is $\int_{-\infty}^{\infty} 2\mu^3 \frac{1}{2\sqrt{2\pi}} \exp(-\frac{(x-2)^2}{2\times 2^2}) d\mu = 64$. The approximations by (2.1) are shown in Table 2.4.

Table 2.4: Approximations to the Expected Value, Example 4

	$p = 100$	$p = 1000$	$p = 10000$	$p = 100000$
$g(x)$	73.6073	69.7909	63.9035	63.6521
$\tilde{g}(x)$	74.6637	68.9387	63.1950	64.0597

These experiments clearly illustrate that more sample values are needed to achieve accurate approximations as the variance increases. When we have some distribution with mild variance, we don't need p to be quite large to achieve desirable approximation.

It is worthy to mention that a random guess from the distribution $f(x,\mu_{guess})$ or the function value of f at the mean of sample values, i.e., $f(x,\bar{\mu})$, are not good approximations for $E[f(x,\mu)]$ in most cases. Use the same function $f(x,\mu) = x\mu^3$ where $\mu \sim N(2,1)$ from previous example. The expected value $E[f(x,\mu)]$ at $x = 2$ is $\int_{-\infty}^{\infty} 2\mu^3 \frac{1}{\sqrt{2\pi}} \exp(-\frac{(x-2)^2}{2}) d\mu = 28$. We generate μ_{guess} randomly from $N(2,1)$ and 100 sample values from the same distribution to calculate the sample mean $\bar{\mu}$. We repeat the experiments 5 times and display the results in Table 2.5.

It is obvious from Table 2.5 that $g(x)$ or $\tilde{g}(x)$ are much more desirable approximations for $E[f(x,\mu)]$ than $f(x,\mu_{guess})$ or $f(x,\bar{\mu})$.

2.2 Compare the Cost of Minimizing $g(x)$ or $\tilde{g}(x)$

To do the minimization in Matlab, we use 'fminunc' in the Matlab optimization toolbox. Function 'fminunc' uses the BFGS method for medium-scale problem by default and the trust region method for large-scale problem. Both of these algorithms require gradients of

Table 2.5: Approximations to the Expected Value, Example 5

	Experiment 1	Experiment 2	Experiment 3	Experiment 4	Experiment 5
μ_{guess}	0.8299	3.3001	4.9863	0.6160	0.2025
$f(x, \mu_{guess})$	1.1432	71.8791	247.9436	0.4675	0.0166
$\bar{\mu}$	2.1206	1.9937	1.8286	2.0154	2.0791
$f(x, \bar{\mu})$	19.0723	15.8481	12.2297	16.3735	17.9755
$g(x)$	32.1296	26.0700	24.1327	31.0705	33.5842
$\tilde{g}(x)$	32.2989	25.9643	24.1192	30.8658	32.6712

the objective function. The trust region method may require the Hessians of the objective function as well. We adopt the automatic differentiation technique to get the gradients and Hessians of the objective function by using ADMAT 2.0.

Let $\omega(f)$ be the cost to evaluate f. The forward mode needs no more than $4n\omega(f)$ to compute the gradients of f with respect to x while the reverse mode needs no more than $4\omega(f)$ in theory. The forward mode needs $O(n^2\omega(f))$ to compute the Hessians of f with respect to x while the reverse mode needs no more than $(10n + 4)\omega(f)$. Not surprisingly, the reverse mode uses more memory because all the intermediate variables are saved. The forward mode and reverse mode of automatic differentiation and their complexities will be introduced in Section 3. Now, we analyze the costs to evaluate function value, gradient, and Hessian of $g(x)$ and $\tilde{g}(x)$ respectively.

Recall the definitions of $g(x)$ and $\tilde{g}(x)$.

$$g(x) \equiv \frac{1}{p}\sum_{i=1}^{p} f(x, \mu_i) \tag{2.4}$$

$$\tilde{f}(x, \mu_i) \equiv \tilde{f}_i(x) = f(x, \bar{\mu}) + \nabla_\mu f(x, \bar{\mu})^T(\mu_i - \bar{\mu}) + \frac{1}{2}(\mu_i - \bar{\mu})^T\nabla^2_\mu f(x, \bar{\mu})(\mu_i - \bar{\mu}) \tag{2.5}$$

$$\tilde{g}(x) \equiv \frac{1}{p}\sum_{i=1}^{p} \tilde{f}_i(x) \tag{2.6}$$

The cost of evaluating g is $p \times \omega(f)$. The cost of evaluating $\tilde{g}$ by reverse mode is $5 \times \omega(f)$ [to evaluate f and the gradient with respect to μ at $x, \bar{\mu}$] + $p \times m$[multiplication to finish first-order term] + $(10m + 4) \times \omega(f)$[to evaluate the Hessian with respect to μ] + $p \times m^2$[multiplication of second-order term]=$\omega(f)(10m + 9) + pm + pm^2$.

The gradient of $\tilde{g}$ is as follows.

$$\nabla_x \tilde{g} = \nabla_x g(x, \bar{\mu}) + \frac{1}{p}\sum_{i=1}^{p} \nabla_x (\nabla_\mu f(x, \bar{\mu})^T (\mu_i - \bar{\mu}) + \frac{1}{2}(\mu_i - \bar{\mu})^T \nabla_\mu^2 f(x, \bar{\mu})(\mu_i - \bar{\mu})) \quad (2.7)$$

$$= \nabla_x g(x, \bar{\mu}) + \frac{1}{p}\sum_{i=1}^{p} \nabla_x (\frac{1}{2}(\mu_i - \bar{\mu})^T \nabla_\mu^2 f(x, \bar{\mu})(\mu_i - \bar{\mu})) \quad (2.8)$$

The cost of evaluating the gradient of g is $p \times 4 \times \omega(f)$. The cost of evaluating the gradient of $\tilde{g}$ is $4\times\omega(f)$ [to evaluate the gradient with respect to x at $x, \bar{\mu}$] + $4\times(10m+4)\times$ $\omega(f)$[to evaluate the Hessian with respect to μ] + $p \times m^2$ [multiplication of second-order term]=$\omega(f)(20 + 40m) + pm^2$.

The Hessian of $\tilde{g}$ is as follows.

$$\nabla_x^2 \tilde{g} = \nabla_x^2 g(x, \bar{\mu}) + \frac{1}{p}\sum_{i=1}^{p} \nabla_x^2 (\nabla_\mu f(x, \bar{\mu})^T (\mu_i - \bar{\mu}) + \frac{1}{2}(\mu_i - \bar{\mu})^T \nabla_\mu^2 f(x, \bar{\mu})(\mu_i - \bar{\mu})) \quad (2.9)$$

$$= \nabla_x^2 g(x, \bar{\mu}) + \frac{1}{p}\sum_{i=1}^{p} \nabla_x^2 (\frac{1}{2}(\mu_i - \bar{\mu})^T \nabla_\mu^2 f(x, \bar{\mu})(\mu_i - \bar{\mu})) \quad (2.10)$$

The cost of evaluating the Hessian of g is $p \times (10n + 4) \times \omega(f)$. The cost of evaluating the Hessian of $\tilde{g}$ is $(10n + 4) \times \omega(f)$ [to evaluate the Hessian with respect to x at $x, \bar{\mu}$] + $(10n+4)\times(10m+4)\times\omega(f)$[to evaluate the Hessian] + $p\times m^2$[multiplication of second-order term]= $\omega(f)[(10n + 4)(10m + 5)] + pm^2$.

The following table shows the computation cost to evaluate g, gradient of g, Hessian of g and those of $\tilde{g}$.

Table 2.6: Computational Cost for g and $\tilde{g}$

	computational cost for g	computational cost for $\tilde{g}$
evaluating function value	$\omega(f)p$	$\omega(f)(10m + 9) + pm + pm^2$
evaluating gradients	$\omega(f)4p$	$\omega(f)(20 + 40m) + pm^2$
evaluating Hessians	$\omega(f)p(10n + 4)$	$\omega(f)[(10n + 4)(10m + 5)] + pm^2$

Thus, it costs much less to evaluate the derivatives of $\tilde{g}(x)$ than those of $g(x)$ when $\omega(f) \gg m^2$ and $p \gg 10m + 9$. These two assumptions are realistic because the cost of evaluating the function itself and the number of sample values in the risk protection approach are generally much larger the number of unknown parameters. Figure 2.1 shows the time needed to evaluate the function value, gradient, and Hessian of $g(x)$ and $\tilde{g}(x)$

while Figure 2.2 shows log of the time needed. In the experiment setup, $\omega(f) \gg 2n^3$; n increases from 5 to 100; m is equal to 2; p is equal to 100. When n is greater than 60, the time spent in evaluating the derivatives of $g(x)$ is unsatisfactory. As for $\tilde{g}(x)$, the gradients and Hessians of $f(x, \mu)$ with respect to μ are obtained in two ways. One is symbolic differentiation; the other is finite differencing. The time needed for $\tilde{g}(x)$ when $n = 100$ is still acceptable.

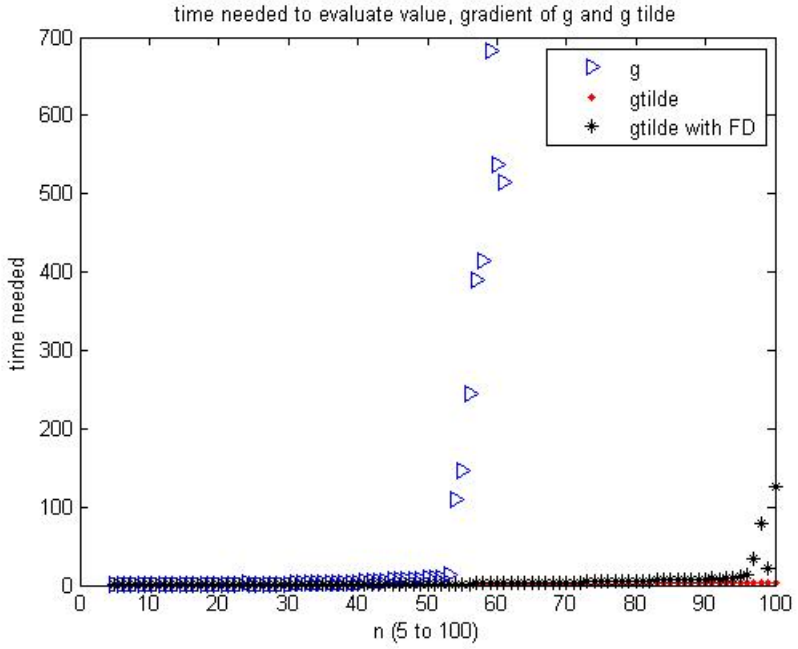

Figure 2.1: Time Needed to Evaluate Derivatives of g and $\tilde{g}$

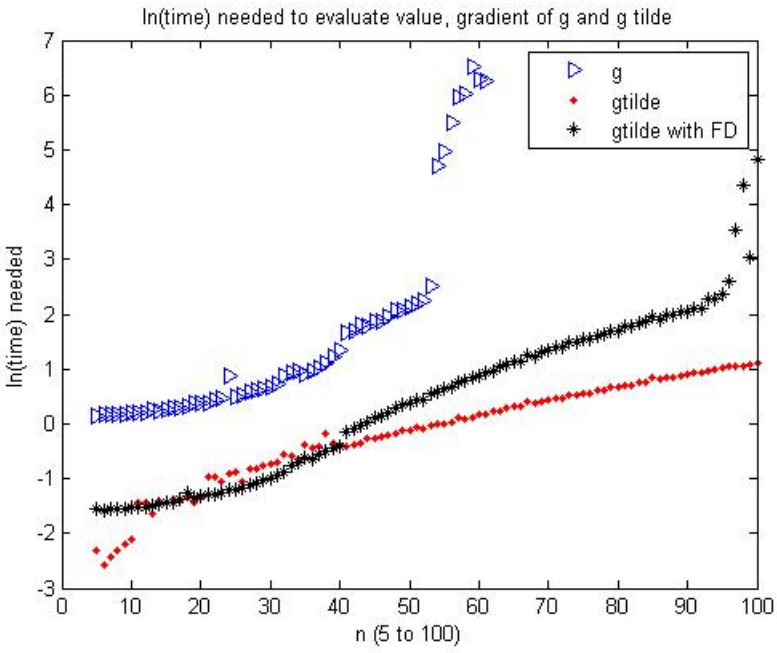

Figure 2.2: ln(Time) Needed to Evaluate Derivatives of g and $\tilde{g}$

Chapter 3

Applications of Risk Protection Approach

3.1 Calculating Greeks of Options

In mathematical finance, the Greeks such as Δ, Γ, ρ, Θ and Vega are used to model the behavior of options in correlation to changes in the value of the underlying financial instruments, interest rates, time and volatility. Let V denote the value of the portfolio of derivatives on a single underlying asset, S denote the price of the underlying asset, r denote the interest rate, $\tau = T - t$ denote the time to maturity, and σ denote the volatility. Mathematically, we have $\Delta = \frac{\partial V}{\partial S}$, $\Gamma = \frac{\partial \Delta}{\partial S} = \frac{\partial^2 V}{\partial S^2}$, $\rho = \frac{\partial V}{\partial r}$, $\Theta = -\frac{\partial V}{\partial \tau}$, and $\nu = \frac{\partial V}{\partial \sigma}$.

Under Black-Scholes framework, closed forms of the Greeks exist by using ϕ, standard normal probability density function, and Φ, standard normal cumulative distribution function. If we define the call option price by

$$C(S,0) = SN(d_1) - Ke^{-rT}N(d_2)$$

and the put option price by

$$P(S,0) = Ke^{-rT}N(-d_2) - SN(-d_1)$$

in the objective function, where $d_1 = \frac{\ln(S/K)+(r+\sigma^2/2)T)}{\sigma\sqrt{T}}$ and $d_2 = d_1 - \sigma\sqrt{T}$, then we can use the 'feval' function in ADMAT 2.0 to calculate the Greeks accurately without knowing the closed form formulas of the Greeks. Experiments show that the Greeks of European call options and put options are the same as the results using the functions in the financial toolbox in Matlab.

3.2 Calculating Sensitivity of Project Value Using Discount Cash Flow Method

Project valuation is essential when the investing pool has a few potential projects. The discounted cash flow analysis is a traditional way to price a project.

Steps of using discounted cash flow analysis include [12]:

1. Estimate the investment cost.
2. Estimate the annual revenues and annual costs and calculate the annual net cash flow for the expected project life cycle.
3. Calculate the present values of each net cash flow.
4. Calculate the net present value of the project.

An example of discount cash flow method is as Table 3.1 shows.

Table 3.1: Discounted Cash Flow Calculations for a Project

	Year						
	0	**1**	**2**	**3**	**4**	**5**	**6**
Investment cost	-$12.00						
Annual revenue		$3.00	$6.00	$10.00	$10.00	$10.00	$6.00
Annual cost		$2.00	$2.50	$3.00	$3.00	$3.00	$2.50
Annual net cash flow		$1.00	$3.50	$7.00	$7.00	$7.00	$3.50
Discount rate	0.20	0.20	0.20	0.20	0.20	0.20	0.20
Discount factor	1.00	0.83	0.69	0.58	0.48	0.40	0.33
PV of annual cash flow	-$12.00	$0.83	$2.43	$4.05	$3.38	$2.81	$1.17
NPV	$2.68						

It is not easy to determine a suitable discount rate in discounted cash flow analysis. Firstly, the discount rate is related to the magnitude of risk. Secondly, we need to distinguish between private risk and market risk associated with the project. If the cash flow stream is influenced by private risk, it is commonly agreed that the investor will not pay a risk premium thus a risk free rate should be used as the discount rate. On the other hand, if market risk is associated, the discount rate should be adjusted, i.e., a proportional risk premium should be added to the risk free rate. The problem is that most of the approaches

to determining the risk-adjusted discount rate are subjective. Thus, the sensitivity of the net present value of a project with respect to the discount rate is worth investigating.

Let the output of the objective function f be the net present value and the variable x be the discount rate. Using ADMAT 2.0, we can calculate the sensitivity $\frac{\partial f}{\partial x}|_{x=0.2} = -43.70$.

3.3 Calculating Sensitivity of Real Option Value Using Binomial Tree Method

It is common that organizations upsize or downsize as the market demands change or technology develops. Option to contract is a real option created for organizations so that they can hedge themselves.

For example, a car builder A introduces a new production-line. However, a competitor B is likely to introduce a similar line at much cheaper cost in two years. Hence, A can buy an option to contract 50 percent of its current size at any time over the next 5 years and gain $300 million in savings. The net present value of the future cash flows is $500; the risk free rate is assumed to be 10% over the 5 years; the annual volatility is estimated to be about 35%. If we choose $\delta t = 1$ year, we can build a binomial tree to calculate the value of the real option.

Firstly, we calculate u, d, and risk neutral probability p by formulas: $u = e^{\sigma\sqrt{\delta t}}$, $d = 1/u$, and $p = \frac{e^{r\delta t}-d}{u-d}$. Then, we construct a binomial tree with asset values at each node as Figure 3.1 shows. Next, we determine the option values at each node of a binomial tree as Figure 3.2 shows. In year 5, the option value is the maximum between the continuation value, which is the same as the asset value, and the contract value, i.e., 0.5*asset value+$300. The option value at each node before the final year is calculated by the maximum between the contract value and the risk-adjusted asset value, i.e., (p*option value at upper node in the next year+(1-p)*option value at lower node in the next year)*discount factor. Carry on the calculations until the single node of year 0 is reached.

In the above example, the value is $563 million at year 0. Because the present value of the future cash flows is $500 million, the difference of $63 million is the value of the real option. Unlike financial options, it is not easy to determine an appropriate volatility of a real option because no historical data can be used. Thus, we are interested in the sensitivity of the real option value with respect to the volatility. Again, we use ADMAT 2.0 to do the analysis. Suppose the output of the objective function f is the real option value and the variable x is the volatility. We get $\frac{\partial f}{\partial x}|_{x=0.35} = 181.79$.

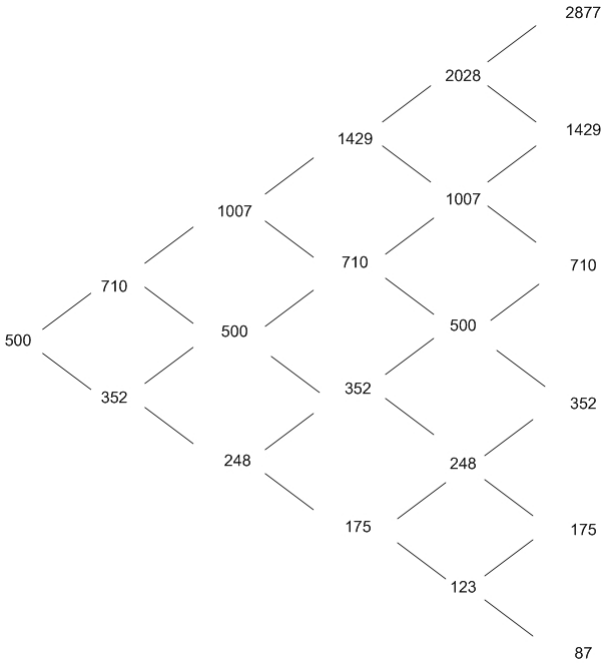

Figure 3.1: Binomial Tree of Asset Value

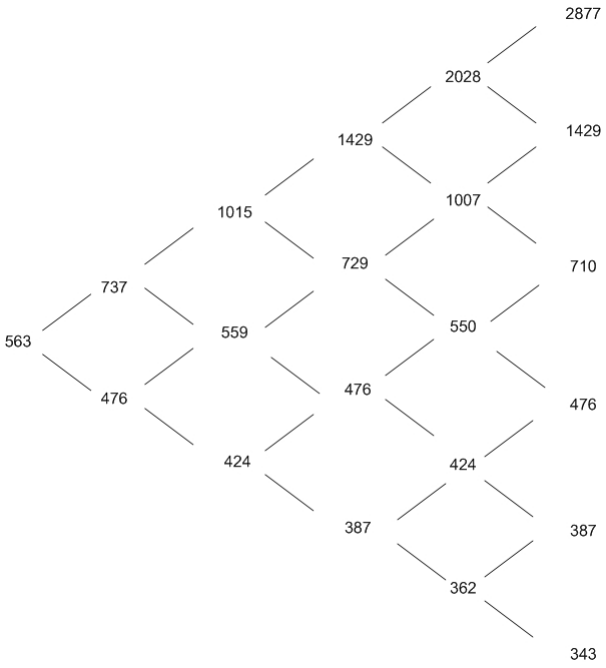

Figure 3.2: Binomial Tree of Option Value

3.4 CVaR Minimization

VaR, or Value at Risk, is a risk measure in financial mathematics. VaR is defined as the worst case loss that could be expected to be incurred from a given portfolio over a certain time period within a specified level of probability. Because VaR lacks subadditivity as a risk measure and neglects extreme loss information, another risk measure CVaR, or conditional VaR, which computes the expected loss given that the loss exceeds VaR, has been proposed.

Consider a portfolio of financial instruments with random returns. Let x denote the portfolio choice vector and μ denote the random returns. Assume that $\rho(\mu)$ is the probability density function of μ and $f(x, \mu)$ is the loss function. $\beta-$CVaR is defined as

$$\text{CVaR}_\beta(x) = \frac{1}{1-\beta} \int_{f(x,\mu) \geq \text{VaR}_\beta(x)} f(x, \mu)\rho(\mu)d\mu \tag{3.1}$$

Let $\mu_1, \cdots, \mu_p$ be sample values of μ chosen from the distribution of μ. (3.2) is an approximation for CVaR.

$$\text{CVaR}_\beta(x) = \frac{1}{p(1-\beta)} \sum_{f(x,\mu_i) \geq \text{VaR}_\beta(x)} f(x, \mu_i) \tag{3.2}$$

Define $F_\beta(x, \alpha)$ as

$$F_\beta(x, \alpha) = \alpha + \frac{1}{1-\beta} \int_{f(x,\mu) \geq \alpha} (f(x, \mu) - \alpha)\rho(\mu)d\mu \tag{3.3}$$

Literature shows that $F_\beta(x, \alpha)$ is convex in α and continuously differentiable at α. Moreover, the minimum value over α of $F_\beta(x, \alpha)$ is $\text{CVaR}_\beta(x)$ [8], i.e.,

$$\min_x \ \text{CVaR}_\beta(x) = \min_{x,\alpha} \ F_\beta(x, \alpha)$$

If $f(x, \mu)$ is convex with respect to x, $F_\beta(x, \alpha)$ is convex with respect to (x, α). Hence, a portfolio with minimum CVaR can be obtained.

Similarly, we can use sample values of μ to approximate $F_\beta(x, \alpha)$. We solve

$$\min_{x,\alpha} \ \alpha + \frac{1}{p(1-\beta)} \sum_{i=1}^{p} (f(x, \mu_i) - \alpha)^+ \tag{3.4}$$

If $f(x, \mu)$ is not linear or quadratic in μ, (3.4) can be approximated by (3.5), i.e., using the risk protection approach.

$$\min_{x,\alpha} \ \alpha + \frac{1}{p(1-\beta)} \sum_{i=1}^{p} (\tilde{f}(x, \mu_i) - \alpha)^+ \tag{3.5}$$

The objective function in (3.5) is continuous but not differentiable. We introduce the same smooth technique in [1] to (3.5) so that the objective function in (3.6) is differentiable.

$$\min_{x,\alpha} \ \alpha + \frac{1}{p(1-\beta)} \sum_{i=1}^{p} \rho_\epsilon(\tilde{f}(x,\mu_i) - \alpha)^+ \tag{3.6}$$

where $\rho_\epsilon(z)$ is defined as

$$\begin{cases} z & \text{if } z \geq \epsilon \\ \frac{z^2}{4\epsilon} + \frac{1}{2}z + \frac{1}{4}\epsilon & \text{if } -\epsilon \leq z \leq \epsilon \\ 0 & \text{otherwise} \end{cases}$$

When the loss function is defined as $f(x,\mu) = -\mu^T x$ and nonnegativity constraint and unit constraint such as $\{x_i \geq 0, \sum_i x_i = 1, i = 1, 2, \cdots, n\}$ are included, the problem

$$\min_{x,\alpha} \ \alpha + \frac{1}{p(1-\beta)} \sum_{i=1}^{p} \rho_\epsilon(\tilde{f}(x,\mu_i) - \alpha)^+$$

$$\text{s.t.} \quad x_i \geq 0, \ \sum_i x_i = 1, \ i = 1, 2, \cdots, n$$

is the standard CVaR minimization problem. In this case, $\tilde{f}(x,\mu_i)$ is the same as $f(x,\mu_i)$.

In our problem, $f(x,\mu)$ is not limited to being linear in μ. Thus, in the Matlab experiments, $f(x,\mu)$ has three settings: linear in μ, quadratic in μ and non-quadratic in μ; p is chosen to be 100; $\epsilon = 1$.

Chapter 4

Conclusion

In this essay, we proposed a risk protection approach for unconstrained nonlinear optimization problem with uncertain parameters. In the risk protection approach, we first generated sample values of μ through sampling, then we approximated each $f(x, \mu_i)$ using a Taylor series around the mean of sample values $\bar{\mu}$. In this way, gradients and Hessians of $\tilde{g}(x)$ were evaluated at $\bar{\mu}$ once instead of at each μ_i as in the case of $g(x)$. We showed that the risk protection approach is a method to minimize the expected value of function f, i.e., $E[f(x, \mu)]$, under desirable computational cost. At the same time, the reverse mode of automatic differentiation technique was used to ensure accurate derivative evaluations and satisfactory running time of minimization procedure. ADMAT 2.0 was used whenever automatic differentiation was needed in the experiments in this essay. Some interesting sensitivity problems in computational finance as well as the CVaR minimization problem, to which ADMAT 2.0 and the risk protection approach can be applied, were investigated with detailed examples. We introduced automatic differentiation technique, forward mode and reversed mode in Appendix. In addition, We briefly explained two nonlinear optimization algorithms in Matlab optimization toolbox, i.e., the BFGS method and the trust region method, in Appendix.

Bibliography

[1] S. Alexander, T.F. Coleman, and Y. Li. Minimizing CVaR and VaR for a portfolio of derivatives. *Journal of banking and finance*, 30(2):583–605, 2006. 16

[2] T.F. Coleman and Y. Li. On the convergence of interior-reflective Newton methods for nonlinear minimization subject to bounds. *Mathematical Programming*, 67(1):189–224, 1994. 29

[3] T.F. Coleman and A. Verma. ADMAT: An automatic differentiation toolbox for MATLAB. In *Proceedings of the SIAM Workshop on Object Oriented Methods for Inter-Operable Scientific and Engineering Computing, SIAM, Philadelphia, PA*, 1998. 2

[4] T.F. Coleman and W Xu. *ADMAT 2.0: An automatic differentiation toolbox for MATLAB.* `http://www.math.uwaterloo.ca/CandO_Dept/securedDownloadsWhitelist/`. 2

[5] T.F. Coleman and W Xu. Quick Start for ADMAT 2.0 Toolbox. Website. `http://www.math.uwaterloo.ca/CandO_Dept/securedDownloadsWhitelist/QuickStart.pdf`. 2

[6] Thomas F. Coleman and Yuying Li. An interior trust region approach for nonlinear minimization subject to bounds. *SIAM Journal on Optimization*, 6(2):418–445, 1996. 29

[7] Thomas F. Coleman and Wei Xu. Fast (structured) newton computations. *SIAM Journal on Scientific Computing*, 31(2):1175–1191, 2008. 2

[8] Gerard Cornuejols and Reha Tutuncu. *Optimization Methods in Finance (Mathematics, Finance and Risk)*. Cambridge University Press, 2007. 15

[9] A. Griewank. On automatic differentiation. pages 83–108. Citeseer, 1989. 21

[10] Andreas Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Number 19 in Frontiers in Appl. Math. SIAM, Philadelphia, PA, 2000. 21

[11] J. Herskovits. *Advances in structural optimization*. Kluwer Academic Pub, 1995. 26

[12] P. Kodukula and C. Papudesu. *Project Valuation Using Real Options: A Practitioner's Guide*. J. Ross Publishing, 2006. 12

[13] J. Sokolowski and J.P. Zolesio. *Introduction to shape optimization*. Springer-Verlag Berlin/New York, 1992. v, 21, 23, 24, 28

[14] Stephen A. Vavasis. *Lecture presented in Continuous Optimization*. University of Waterloo, ON, 2008. 28, 30

Appendix A

Methods to Get Derivatives

A.1 Symbolic Differentiation, Finite Differencing and Automatic Differentiation

Most algorithms for minimization problems require gradients and Hessians. Traditional ways of getting gradients and Hessians include differentiation by hand, symbolic differentiation, and finite differencing.

Taking derivatives by hand is very inefficient and error-prone. Symbolic differentiation can automatically produce an expression for the derivatives of a function. However, the approach is quite limited because it does not allow loops, branches, and subroutines in the computer program. Thus, when loops or branches occur in the objective function, human effort is needed to break up the complicated program to small pieces and reassemble all the results after differentiation.

Finite differencing uses approximations for the derivatives. For example, forward finite differencing uses the following formula to approximate the gradient,

$$\frac{\partial f(x_0)}{\partial x_i} \approx \frac{f(x_0 + \delta e^{(i)}) - f(x_0)}{\delta}$$

where $e^{(i)}$ is the i-th unit vector and $\delta > 0$. There are several disadvantages of finite differencing. Firstly, the accuracy of the approximation is hard to estimate. Secondly, δ has to be chosen suitably, which is not easy. In theory, δ needs to be small so that the higher order terms in the Taylor expansion of f can be ignored. However, if δ is too small, the cancellation error because of the subtraction of two similar values could be significant. Moreover, finite differencing is expensive, especially for higher order derivatives. Lastly, $x_0 + \delta e^{(i)}$ may be outside of the domain of f.

The research on automatic differentiation has been active since the last century. Automatic differentiation is a technique for automatically augmenting computer programs with derivative computations [9]. The idea is that each program can be broken into a sequence of elementary arithmetic operations and then the chain rule of calculus can be applied to these operations repeatedly. As a result, accurate derivatives of arbitrary order can be obtained automatically.

Besides the advantage of being accurate, automatic differentiation has the potential of less cost to get the gradient and the Hessian compared with finite differencing. If the objective function is defined as $f : R^n \to R$, the cost of computing the gradient of f with either finite differencing or symbolic differentiation grows linearly with the number of variables n. In contrast, the cost of evaluating the function value and the gradient of f by reverse mode in automatic differentiation is at most four times the cost of evaluating f [10], which is unrelated to n. The forward mode and reverse mode of automatic differentiation will be introduced next.

All work in this essay using automatic differentiation is done by ADMAT 2.0.

A.2 Forward Mode and Reverse Mode in Automatic Differentiation [13]

Let Φ be defined as $\Phi : R^n \to R^m$. Any program to evaluate Φ at $\xi \in R^n$ can be written as a sequence of K scalar assignment statements: There are K variables in total

Algorithm A.2.1 Evaluate Φ

for $i = 1$ to n **do**
  $x_i = \varphi_i(x_i) = \xi_i$
end for
for $i = n + 1$ to K **do**
  $x_i = \varphi(x_1, \cdots, x_{i-1})$
end for

in Algorithm A.2.1. Since Φ is defined as $\Phi : R^n \to R^m$, we call the first n variables *independent variables*. The last m variables are called *output variables* and the rest are called *temporary variables*. We require the function φ_i to be elementary functions, i.e., each φ_i is basic arithmetic operations ($\pm$, *, /) or a univariate transcendental function (exp, cos, etc).

Our goal is to differentiate the output variables with respect to the independent variables. We say a variable $x_i,\ i > n$ is *active* when either an independent variable or an

active variable is assigned to it. We apply chain rule to both sides of the equations in Algorithm A.2.1 and obtain the following results:

$$\frac{\partial x_i}{\partial x_j} = \begin{cases} \delta_{ij} + \sum_{k=1}^{i-1} \frac{\partial \varphi_i}{\partial x_k} \frac{\partial x_k}{\partial x_j}, & j \le i \le K \\ 0, & j > i \end{cases} \tag{A.1}$$

Define the following matrices

$$D\varphi := \left\{ \frac{\partial \varphi_i}{\partial x_j} \right\}_{i,j=1}^{K} = \begin{pmatrix} 0 & \cdots & & \\ \partial\varphi_2/\partial x_1 & 0 & \cdots & \\ \partial\varphi_3/\partial x_1 & \partial\varphi_3/\partial x_2 & 0 & \cdots \\ \vdots & & & \end{pmatrix}$$

$$Dx := \left\{ \frac{\partial x_i}{\partial x_j} \right\}_{i,j=1}^{K} = \begin{pmatrix} 1 & \cdots & & \\ \partial x_2/\partial x_1 & 1 & \cdots & \\ \partial x_3/\partial x_1 & \partial x_3/\partial x_2 & 1 & \cdots \\ \vdots & & & \end{pmatrix}$$

Then (A.1) can be written in matrix form as follows,

$$Dx = I + (D\varphi)(Dx)$$

This is equivalent to

$$(I - D\varphi)Dx = I \tag{A.2}$$
$$Dx(I - D\varphi) = I$$

and

$$(I - D\varphi)^T (Dx)^T = I \tag{A.3}$$

The system (A.2) is lower triangular. Thus, $\partial x_i/\partial x_j$ can be computed by using the forward substitutions

for $i = 2$ to K **do**
  for $j = 1$ to $i - 1$ **do**
    $$\frac{\partial x_i}{\partial x_j} = \sum_{k=j}^{i-1} \frac{\partial \varphi_i}{\partial x_k} \frac{\partial x_k}{\partial x_j}$$
  end for
end for

Because forward substitutions are used, the method is called forward mode of automatic differentiation. Similarly, the system (A.3) is upper triangular, so we can derive the reverse mode of automatic differentiation using backward substitutions

for $j = K-1$ to 1 **do**
  for $i = K$ to $j+1$ **do**

$$\frac{\partial x_i}{\partial x_j} = \sum_{k=j+1}^{i} \frac{\partial \varphi_k}{\partial x_j} \frac{\partial x_i}{\partial x_k}$$

  end for
end for

A.3 Examples of Automatic Differentiation

In this subsection, we give examples on forward mode and reverse mode of automatic differentiation respectively. We present the program to obtain the gradient of a real-valued function $f : R^n \to R$. Suppose $f = x_1/x_2 + \cos(x_1 + x_2)$.

The program to evaluate f can be written as follows

$$\begin{aligned} x_1 &= \xi_1 \\ x_2 &= \xi_2 \\ x_3 &= x_1/x_2 \\ x_4 &= x_1 + x_2 \\ x_5 &= \cos(x_4) \\ x_6 &= x_3 + x_5 \\ f &= x_6 \end{aligned}$$

Generally, if an original program of a real-valued function $f : R^n \to R$ is as Algorithm A.3.1 shows, the augmented program using the forward mode is as Algorithm A.3.2 shows [13].

Algorithm A.3.1 Evaluate f by using a sequence of basic operations

for $i = 1$ to n **do**
  $x_i = \varphi_i(x_i) = \xi_i$
end for
for $i = n+1$ to K **do**
  $x_i = \varphi_i(x_1, \cdots, x_{i-1})$
end for
$f = x_K$

For the above example, the forward mode computes the gradient in the following order

$$x_1 = \xi_1, \quad \nabla x_1 = (1, 0)^T$$

Algorithm A.3.2 Augmented program using the forward mode

for $i = 1$ to n **do**
  $x_i = \varphi_i(x_i) = \xi_i$
  $\nabla x_i = e^{(i)}$
end for
for $i = n+1$ to K **do**
  $x_i = \varphi_i(x_1, \cdots, x_{i-1})$
  $\nabla x_i = \sum_{j=1}^{i-1} \frac{\partial \varphi_i}{\partial x_j} \nabla x_j$
end for
$f = x_K$
$\nabla f = \nabla x_K$

$$
\begin{aligned}
&x_2 = \xi_2, \quad \nabla x_2 = (0,1)^T \\
&x_3 = \varphi_3(x_1, x_2) = x_1/x_2 \\
&\nabla x_3 = \frac{\partial \varphi_3}{\partial x_1}\nabla x_1 + \frac{\partial \varphi_3}{\partial x_2}\nabla x_2 = \frac{1}{x_2}\nabla x_1 + \frac{-x_1}{x_2^2}\nabla x_2 = (\frac{1}{x_2}, \frac{-x_1}{x_2^2})^T \\
&x_4 = \varphi_4(x_1, x_2) = x_1 + x_2 \\
&\nabla x_4 = \frac{\partial \varphi_4}{\partial x_1}\nabla x_1 + \frac{\partial \varphi_4}{\partial x_2}\nabla x_2 = \nabla x_1 + \nabla x_2 = (1,1)^T \\
&x_5 = \varphi_5(x_4) = \cos(x_4) \\
&\nabla x_5 = \frac{\partial \varphi_5}{\partial x_4}\nabla x_4 = -\sin(x_4)\nabla x_4 = (-\sin(x_4), -\sin(x_4))^T \\
&x_6 = \varphi_6(x_3, x_5) = x_3 + x_5 \\
&\nabla x_6 = \frac{\partial \varphi_6}{\partial x_3}\nabla x_3 + \frac{\partial \varphi_6}{\partial x_5}\nabla x_5 = \nabla x_3 + \nabla x_5 = (\frac{1}{x_2} - \sin(x_4), \frac{-x_1}{x_2^2} - \sin(x_4))^T \\
&\nabla f = \nabla x_6 = (\frac{1}{x_2} - \sin(x_4), \frac{-x_1}{x_2^2} - \sin(x_4))^T
\end{aligned}
$$

In the reverse mode, the augmented program is as Algorithm A.3.3 shows [13], For the same example, the reverse mode calculates the gradient as follows,

$$
\begin{aligned}
&x_3 = \varphi_3(x_1, x_2) = x_1/x_2 \\
&x_4 = \varphi_4(x_1, x_2) = x_1 + x_2 \\
&x_5 = \varphi_5(x_4) = \cos(x_4) \\
&x_6 = \varphi_6(x_3, x_5) = x_3 + x_5 \\
&f = x_6
\end{aligned}
$$

Algorithm A.3.3 Augmented program using the reverse mode

for $i = 1$ to n **do**
  $x_i = \xi_i$
end for
for $i = n + 1$ to K **do**
  $x_i = \varphi_i(x_1, \cdots, x_{i-1})$
end for
$f = x_K$
for $i = 1$ to $K - 1$ **do**
  $\bar{x}_i = 0$
end for
$\bar{x}_K = 1$
for $i = K$ to $n + 1$ **do**
  $\bar{x}_j = \bar{x}_j + \frac{\partial \varphi_i}{\partial x_j} \bar{x}_i \qquad \forall j \leq i$
end for
$\nabla f = \{\bar{x}_i\}_{i=1}^n$

$$
\begin{aligned}
&\bar{x}_i = 0, \quad i = 1, \cdots, 5 \\
&\bar{x}_6 = 1 \\
&\bar{x}_3 = \bar{x}_3 + \bar{x}_6 \frac{\partial \varphi_6}{\partial x_3} = 1 \\
&\bar{x}_5 = \bar{x}_5 + \bar{x}_6 \frac{\partial \varphi_6}{\partial x_5} = 1 \\
&\bar{x}_4 = \bar{x}_4 + \bar{x}_5 \frac{\partial \varphi_5}{\partial x_4} = -\sin(x_4) \\
&\bar{x}_1 = \bar{x}_1 + \bar{x}_4 \frac{\partial \varphi_4}{\partial x_1} = -\sin(x_4) \\
&\bar{x}_2 = \bar{x}_2 + \bar{x}_4 \frac{\partial \varphi_4}{\partial x_2} = -\sin(x_4) \\
&\bar{x}_1 = \bar{x}_1 + \bar{x}_3 \frac{\partial \varphi_3}{\partial x_1} = -\sin(x_4) + \frac{1}{x_2} \\
&\bar{x}_2 = \bar{x}_2 + \bar{x}_3 \frac{\partial \varphi_3}{\partial x_2} = -\sin(x_4) + \frac{-x_1}{x_2^2} \\
&\nabla f = (\bar{x}_1, \bar{x}_2)^T = (-\sin(x_4) + \frac{1}{x_2}, -\sin(x_4) + \frac{-x_1}{x_2^2})^T
\end{aligned}
$$

The gradients from the forward mode and reverse mode are the same. Because all the intermediate variables are saved in the reverse mode, the reverse mode requires extra storage.

A.4 Complexities of the Forward Mode and the Reverse Mode

The complexity results for Automatic Differentiation algorithms are given in Table A.1 [11].

Table A.1: Complexity Results for AD Algorithms

	forward	reverse
$L(f, \nabla f)$	$\leq 4nL(f)$	$\leq 4L(f)$
$L(f, \nabla f, H)$	$O(n^2 L(f))$	$\leq (10n+4)L(f)$
$S(f, \nabla f)$ $S(f, \nabla f, H)$	$O(S(f))$	$O(S(f)+L(f))$
$L(F, DF)$	$O(nL(f))$	$\leq (3m+1)L(f)$
$S(F, DF)$	$O(S(f))$	$O(S(f)+L(f))$

where the definitions are

$f : R^n \to R$
$F : R^n \to R^m$
∇f : the gradient of f
DF : the Jacobian of f
$L(\cdots)$: number of basic operations to compute $(\cdots)$
$S(\cdots)$: work space needed to compute $(\cdots)$

Appendix B

Minimization Algorithms for Nonlinear Unconstrained Problem

Suppose f is an at least twice continuously differentiable real-valued function defined as $f : R^n \to R$. We consider the unconstrained minimization problem, i.e.,

$$\min_{x \in R^n} f(x)$$

Let x_0 be the initial guess of the minimizer. Using Taylor expansion, the function value f can be approximated near x_0 by

$$f(x_0 + p) \approx f(x_0) + \nabla f(x_0)^T p + \frac{1}{2} p^T H(x_0) p, \quad p \in R^n \tag{B.1}$$

If the Hessian matrix is positive definite at x_0, then the minimizer of the right of (B.1) with respect to p is unique determined by $H(x_0)p = -\nabla f(x_0)$. A new approximation of f can be obtained at $x_1 = x_0 + p$.

The above method is called Newton's method. The advantage of Newton's method is the quadratic convergence speed in the last stages of iteration under some conditions. However, the classic Newton's method also has several drawbacks. The first disadvantage is that the Hessian at x_k may not be positive definite. Thus, the search direction $p_k = -[H(x_k)]^{-1}\nabla f(x_k)$ may not be a descent direction. In addition, we need to solve an $n \times n$ system of linear equations to obtain the Hessian and solving the linear system requires $O(n^3)$ operations using Cholesky decomposition or Gaussian elimination. Moreover, Newton's method has only local convergence, i.e., the algorithm converges to the true local minimizer only if the initial point x_0 is close to the true local minimizer.

In Matlab, the 'fminunc' function in the optimization toolbox uses two other minimization algorithms for nonlinear unconstrained optimization: the BFGS method by default for

medium-scale problems and the trust region method for large-scale problems. The BFGS method is one of the quasi-Newton methods. It uses some positive definite matrix to replace the Hessian at each iteration. Thus, each search direction p_k is a descent direction. Moreover, the cost of obtaining the search direction in the BFGS method can be reduced to $O(n^2)$. The trust region method fixes the local convergence problem with the Newton's method.

B.1 BFGS Method

Instead of using Hessians in the iterations, the BFGS method uses some positive definite matrix to approximate the Hessian at each iteration. The framework of the BFGS method is as Algorithm B.1.1 shows.

Algorithm B.1.1 Framework of BFGS Method [14]

1: **for** $k = 0, 1, 2, \cdots$ **do**

2: $p_k = -B_k^{-1}\nabla f(x_k)$

3: select α_k via line search

4: $x_{k+1} = x_k + \alpha_k p_k$

5: Use formula to obtain B_{k+1} from B_k

6: **end for**

Two conditions must be met in BFGS method. One is called the secant condition; the other is the Wolfe conditions in the line search. If we use a sequence of $\{B_k\}$ to approximate the Hessians, the secant condition is as follows

$$B_{k+1}(x_{k+1} - x_k) = \nabla f(x_k + 1) - \nabla f(x_k) \tag{B.2}$$

The Wolfe conditions in line search are required so that B_{k+1} is positive definite provided that B_k is positive definite. The Wolfe conditions require that

$$(\nabla f(x + p) - \nabla f(x))^T (x_{k+1} - x_k) > 0$$

The complete algorithm for the BFGS method is as Algorithm B.1.2 shows [13]. Step 2 requires $O(n^3)$ operations to obtain the search direction p_k. If we update $H_{k+1} = B_{k+1}^{-1}$ from $H_k = B_k^{-1}$ using Sherman-Morrison-Woodbury formula, we can obtain $p_k = -H_k\nabla f(x_k)$ in $O(n^2)$ operations.

Algorithm B.1.2 Algorithm of BFGS Method

1: Choose initial guess x_0, a tolerance parameter $\epsilon > 0$. Set $B_0 = I$ and $k = 0$

2: Calculate the search direction p_k by solving $p_k = -B_k^{-1}\nabla f(x_k)$

3: select α_k via line search

4: Update $x_{k+1} = x_k + \alpha_k p_k$

5: **if** $\|\nabla f(x_{k+1})\| \leq \epsilon$ **then**

6: stop

7: **end if**

8: Update the Hessian approximation by $B_{k+1} = B_k - \frac{B_k s_k (B_k s_k)^T}{s_k^T B_k s_k} + \frac{y_k (y_k)^T}{(y_k)^T s_k}$, where $s_k = x_{k+1} - x_k$, $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$

B.2 Trust Region Method

In the trust region method, we define the Quadratic Model as follows

$$m(x_k + p) = f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T \nabla^2 f(x_k) p \tag{B.3}$$

The idea is that we assume that $m(x_k + p)$ is valid only for p such that $\|p\| \leq \Delta$. We obtain $x_{k+1} = x_k + p_k$ by solving the Trust Region Subproblem defined as follows

$$p_k = \arg\min m(x_k + p), \qquad s.t. \qquad \|p\| \leq \Delta \tag{B.4}$$

On each iteration, the trust region method finds the minimizer of the Quadratic Model in a ball of radius Δ_k about x_k. Depending on whether the model faithfully reflects f, set

$$\Delta_{k+1} = \begin{cases} \Delta_k \\ \frac{1}{4}\Delta_k \\ 2\Delta_k \end{cases}$$

The overall algorithm of the trust region method is as Algorithm B.2.1 shows.

The trust region method has global convergence. Furthermore, the sequences using this algorithm will converge to x^* satisfying second order necessary condition, i.e., $\nabla^2 f(x^*)$ is positive semi-definite. The trust region method in 'fminunc' function is more complicated by incorporating other techniques [6, 2].

Algorithm B.2.1 Algorithm of Trust Region Method [14]

Given x_0, Δ

for $k = 0, 1, 2, \cdots$ **do**

Compute search direction p_k by solving Trust Region Subproblem (B.4)

$\rho_k = \frac{f(x_k) - f(x_k + p_k)}{m(x_k) - m(x_k + p_k)}$

if $\rho_k < \frac{1}{4}$ **then**

$\Delta_{k+1} = \frac{1}{4}\Delta_k$

else if $\rho_k > \frac{3}{4}$ and $\|p_k\| = \Delta_k$ **then**

$\Delta_{k+1} = 2\Delta_k$

else

$\Delta_{k+1} = \Delta_k$

end if

if $\rho_k > \eta$ **then**

$x_{k+1} = x_k + p_k$

else

$x_{k+1} = x_k$

end if

end for