Computation of Loss Distribution Based on the Structural Model for Credit Portfolios

by

Meng Han

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Computation of Loss Distribution Based on the Structural Model for Credit Portfolios

Meng Han

Doctor of Philosophy

Graduate Department of Computer Science

University of Toronto

2018

Credit risk analysis and management at the portfolio level is a challenging issue for financial institutions due to their portfolios' large size, heterogeneity and complex correlation structure. In this thesis, we propose several new asymptotic methods and exact methods to compute the distribution and VaR of a loan portfolio's loss in the CreditMatrics framework. For asymptotic methods, we give an approximation based on the Central Limit Theorem (CLT), which gives more accurate approximations to the conditional portfolio loss probabilities compared with existing approximations. For exact methods, we improve the efficiency by exploiting the sparsity that often arises in the obligors' conditional losses. A sparse convolution method and a sparse FFT method are proposed, which enjoy significant speedups compared with the straightforward convolution method. We also construct truncated versions of the sparse convolution method and the sparse FFT method to further improve their efficiency. To control the aliasing errors and roundoff errors incurred in the truncated sparse FFT method, an optimal exponential windowing approach is developed as well. For lumpy portfolios, we introduce hybrid methods which combine an asymptotic approximation with Monte Carlo simulation or one of exact methods to achieve a good balance between efficiency and accuracy.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisors, Prof. Ken Jackson and Dr. Alex Kreinin, for their continuous support of my Ph.D. study and research. Their patience, inspiration, motivation, and immense knowledge helped me to conquer numerous challenges along the journey. I could not imagine having better advisors and mentors for my Ph.D. study and research.

I would also like to thank the other members in my Ph.D. committee, Prof. Sebastian Jaimungal and Prof. Rudi Mathon, not only for their time and effort to read my thesis and to help me to improve it, but also for their challenging questions which encouraged me to widen and to deepen my research. Special thanks to Prof. David Saunders for serving as the External Examiner for my Final Oral Exam and for raising challenging questions and for making helpful recommendations. I would also like to thank Prof. Christina Christara for agreeing to be a member of my Final Oral Committee and for making constructive suggestions for improvements to my thesis.

I am also grateful to the following colleagues and friends in the Numerical Analysis Group in the Computer Science Department at the University of Toronto: Dr. Wanhe Zhang, Dr. Xiaofang Ma, Dr. Dongwoon Lee, Zhe (Robert) Wang, Zhenan (Eric) Fan, Yuehao (Josh) Qin, and Anton Braverman. Discussions with them were fruitful, inspiring and encouraging. I am also thankful to comments and suggestions I received from the audiences of my presentations at several scientific conferences.

I am very grateful for the generous financial support provided by Prof. Ken Jackson and Computer Science Department at the University of Toronto which helped to fund my Ph.D. study and research.

This journey would not have been possible without the support of my family. To my parents, thank you for your endless support and encouragement throughout my life. You taught me to pursue my goals with heart and soul, and to push myself out of my comfort zone. To my uncle, thank you for the excellent example you set, inspiring me to start and finish this journey. To my grandparents, thank you for your unconditional love since I was born. I wish I could spend more time with you. Rest in peace, Grandpa. I miss you. To my sons, Cody and Leo, thank you for your patience and understanding during Daddy's Ph.D. journey. Your smiles are always the source of my inner peace no matter what the crisis Daddy is facing. Finally, I do not know how to start to thank my wonderful wife, Bin. You have been with me through thick and thin with your love, understanding and support, and I think the biggest blessing in my life is having you as my wife. Thank you for everyting.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

In the current market environment, financial institutions are even more vulnerable than they have been in the past to borrowers' credit events, such as default or downgrading of the credit rating of instruments in their portfolios by rating agencies. Therefore, credit risk analysis and management at the portfolio level has become even more important. Since the implementation of Basel II, regulators have required financial institutions to reserve credit-risk capital to absorb unexpected credit losses. The determination of the amount of capital required involves a computation of the credit value-at-risk (credit VaR), which is calculated by either a standardized approach or an internal ratings based (IRB) approach. However, financial institutions do not have complete freedom to calculate the regulatory capital any way they want: they must use some models and parameters which are specified by the Basel Committee. To realize and manage their own risk, financial institutions tend to use their own internal model to compute credit VaR for economic capital. In addition to credit VaR, to understand the risk involved in some complex credit derivatives, such as collateralized debt obligations and basket credit derivatives, risk managers need to compute the whole loss distribution. Therefore, the accurate and efficient calculation of the credit loss probabilities for loan portfolios is an increasingly important problem.

For credit risk at the individual level, structural models are very popular in both industry and academia. These models rely on the company's capital structure, and a firm's default is triggered when its assets' value fall below its debt level. Merton's famous paper [44] gave birth to this class of models. Based on Merton's model, many industrial structural models have been developed, among which JP Morgan's CreditMetrics model [12, 22] and Moody KMV's KMV model [11, 12] are the most famous. The CreditMetrics model relies on rating agencies' credit ratings, and assumes that the obligors within the same rating class are homogeneous, while the KMV model computes the firm-specific default probability

with Merton's option pricing approach and Moody KMV's massive historic default database. At the portfolio level, modeling the correlations between obligors plays a central role. To estimate correlations more efficiently, both the CreditMetrics [22] and KMV [6, 12] models apply multi-factor decomposition to the systematic risk factors. The main ideas underlying these correlation models are similar, but the CreditMetrics model uses the correlation between equity returns as a proxy for calculating correlations of asset returns, while the KMV model applies a more sophisticated procedure to calibrate asset correlations using asset return data.

Under the CreditMetrics framework, the loss probability equals the expectation of the loss probability conditional on the systematic risk factors. The most appropriate way to calculate the expectation is by simulation-based methods, such as Monte Carlo (MC) simulation or Quasi Monte Carlo (QMC) simulation. There are at least three classes of approaches used to compute conditional probability. The first, and most commonly-used, is MC simulation. However, this leads to a two-level simulation for the computation of the unconditional probability, which makes this simulation approach extremely computationally intensive. To accelerate the computation, Glasserman et al. [18, 19] developed importance sampling techniques for both the inner-level and the outer-level MC simulations, but this method is still computationally intensive, due to the nature of MC simulation. Moreover, the randomness of the errors involved in the MC simulation is another disadvantage of this approach, especially for sensitivity analysis. The second class of approaches exploits the conditional independence structure to approximate the loss probabilities by the law of large numbers (LLN) [20, 21]. The advantage of the asymptotic approximations lies in their efficiency: they use much less computational time than MC simulation. On the other hand, they suffer from inaccuracy, to some extent, if the portfolio is not fine-grained enough. To improve the accuracy of the LLN approximations, granularity adjustments have been studied by Gordy [20, 21], Wilde [60], Martin and Wilde [40] for the single-factor setting, and by Pykhtin [51] for the multi-factor setting. The last class of approaches computes loss probability exactly, although these approaches are still subject to some roundoff errors and possibly aliasing errors. One subclass of the exact methods computes the conditional loss probability directly or indirectly by a discrete linear convolution [3, 13, 15, 41, 43]. The most straightforward approach is to compute the linear convolution directly, which in general leads to an algorithm with complexity $O\left(K^2 + NK\right)$, where K is the number of points in the distribution of portfolio losses and N is number of obligors in the portfolio. Another popular approach is to compute the linear convolution by fast Fourier transforms (FFTs), which reduces the complexity to $O\left(NK \log_2 K\right)$.

In this thesis, we propose several new methods to compute conditional loss probabilities. First, we introduce an asymptotic approximation based on the central limit theorem (CLT). We prove that both

the conditional and unconditional loss distributions converge to our CLT limiting distribution under conditions similar to those used by Gordy to prove the convergence of the LLN approximation. We analyze the error incurred by the CLT approximation by using Berry-Esseen-type bounds. Secondly, we improve the efficiency of the exact methods by exploiting the sparsity that often arises in the obligors' conditional losses. We develop a sparse convolution method which enjoys a speedup between $\Omega\left(\alpha^2\right)$ and $\Omega\left(\alpha^2 C^N\right)$ compared with the straightforward convolution method, where C is the number of rating classes in a credit rating system, and α is a spacing constant. To further accelerate the computation, we introduce a truncated sparse convolution method, which is subject to some additional truncation error to a user-specified level, but gives a significant speedup compared to the sparse convolution method. Moreover, we develop a sparse FFT method which gives a speedup of $\Omega\left(K/N\right)$ in the best case and $\Omega\left(K/\left(N\log_2 C\right)\right)$ in the worst case. We also construct a truncated sparse FFT method to further improve its efficiency, with an optimal exponential windowing approach used to balance aliasing errors and roundoff errors. Lastly, for "lumpy" portfolios, composed of a fairly homogeneous sub-portfolio and an inhomogeneous sub-portfolio having very large exposure to a few obligors, we propose a hybrid method combining an asymptotic method with an exact method or MC simulation to achieve a good balance between accuracy and efficiency to compute loss distribution and VaR.

The rest of this thesis is organized as follows. Chapter 2 discusses credit risk at the obligor's level and at the portfolio level, and introduces the general model setting. Chapter 3 outlines the structural models and the multi-factor correlation model, and describes how to calculate the portfolio loss probability using a two-level computation. In Chapter 4, we first review the LLN approximation and then we introduce the CLT approximation. Chapter 5 discusses the (truncated) sparse convolution method and the (truncated) sparse FFT method, and compares their performance to the traditional full convolution method and the full FFT method. In Chapter 6, we introduce the hybrid method. Finally we present our conclusions and discuss future work, including our new importance sampling approach, in Chapter 7.

Chapter 2

Preliminaries

2.1 Credit Risk

When investors, either institutional or individual, purchase a financial instrument, such as a bond, issued by a firm, an important point that they should consider is whether the firm will have the ability to pay interest during the term of the bond, and to repay the principle at maturity. If for some reason the firm defaults on a payment (coupon or principal), investors will suffer a financial loss. Although defaults are not frequent events, if they take place, they often result in significant losses. The risk of loss due to default is often called default risk, which is the most commonly considered credit risk.

To describe the credit worthiness of bonds, rating companies, such as Moody's, Standard & Poor's (S&P) and Fitch, provide credit ratings for firms. In addition, some institutional investors also use their own internal methods to rate firms. Though the credit rating is associated with the credit worthiness of a bond, almost all bonds issued by a firm have the same rating, so the credit rating can be considered as an attribute of a firm. Table 2.1 shows the credit ratings offered by different rating companies and the risk worthiness corresponding to each rating class.

If a firm's financial health deteriorates or the economy exhibits a downturn, a firm's credit rating might be downgraded to a lower rating class by rating companies, although the frequency of rating migrations is relatively low. Once a rating downgrade happens, the value of this firm's bonds and associated derivatives, such as CDS, will likely drop, which usually results in the debt holders' financial loss. The risk associated with rating migrations is called migration risk, which is another aspect of credit risk.

Quantifying the credit risk of a firm is not an easy task, since default or credit migration events occur

Table 2.1: Credit ratings offered by different rating companies

Credit Ratings			**Risk Worthiness**
Moody's	**S&P**	**Fitch**	
Aaa	AAA	AAA	Credit risk almost zero.
Aa1	AA+	AA+	Safe investment, low risk of failure.
Aa2	AA	AA	
Aa3	AA-	AA-	
A1	A+	A+	Safe investment, unless unforeseen events should occur in the economy at large or in that particular field of business.
A2	A	A	
A3	A-	A-	
Baa1	BBB+	BBB+	Moderately safe investment. Problems may arise if the economy deteriorates.
Baa2	BBB	BBB	
Baa3	BBB-	BBB-	
Ba1	BB+	BB+	Speculative investment. Problems likely to arise if the economy deteriorates. Usually difficult to predict future developments.
Ba2	BB	BB	
Ba3	BB-	BB-	
B1	B+	B+	Speculative investment. Deteriorating situation expected.
B2	B	B	
B3	B-	B-	
Caa1	CCC+	CCC	Bankruptcy or other serious business problems very likely.
Caa2	CCC		
Caa3	CCC-		
Ca	CC		
\	D	DDD	Bankruptcy or lasting inability to make payments almost certain.
		DD	
		D	

Table 2.2: Credit migration matrix (Source: S&P)

	AAA	**AA**	**A**	**BBB**	**BB**	**B**	**CCC/C**	**D**
AAA	95.60%	2.20%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
AA	0.60%	91.37%	3.21%	0.00%	0.00%	0.00%	0.00%	0.00%
A	0.00%	2.90%	86.26%	2.75%	0.22%	0.30%	0.07%	0.00%
BBB	0.00%	0.26%	3.81%	83.69%	2.70%	0.66%	0.07%	0.00%
BB	0.00%	0.00%	0.00%	6.72%	75.26%	6.44%	0.09%	0.19%
B	0.00%	0.00%	0.00%	0.08%	7.43%	75.40%	2.56%	0.24%
CCC/C	0.00%	0.00%	0.00%	0.00%	0.00%	20.00%	45.45%	14.55%
D	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%

unexpectedly. To help understand this randomness, a credit migration matrix, which gives probabilities of migration from one rating class to another in a given period (usually one year), is calculated by rating companies. For example, Table 2.2 shows S&P's 2007 global migration matrix for a one-year period. It shows, for example, that the probability that a BB firm will be downgraded to B within one year is 6.44%, while the probability that a BB firm will default on its bonds within one year is 0.19%.

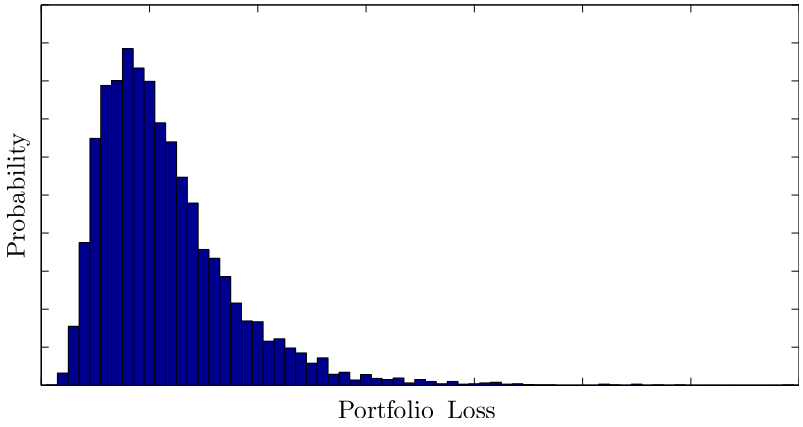
Such credit migration matrices are calculated based on default models and huge proprietary databases. Different rating companies use different default models, among which structural models are very popular in both industry and academia. The first structural model was proposed by Merton [44] in 1973, and many industrial structural models have been developed, such as JP Morgan's CreditMetrics model [12, 22] and Moody KMV's KMV model [11, 12]. In the next chapter, we discuss Merton's model and the CreditMetrics model.

2.2 Credit Risk in Credit Portfolios

At the portfolio level, credit risk is more complicated, since the distribution of credit portfolio losses is highly skewed and fat-tailed (as shown in Figure 2.1). Therefore, the two statistical measures mean and variance, which are frequently used to analyze market risk in equity portfolios, are not sufficient to help us understand the credit risk. Instead, quantile-based risk measures, such as VaR and CVaR, need to be used. Moreover, to understand the risk involved in some complex credit derivatives, such as collateralized debt obligations and basket credit derivatives, risk managers need to compute the whole loss distribution.

Handling credit quality correlations of obligors in the portfolio is another challenge. A credit portfolio may consist of thousands of obligors, and the default correlation is embedded at different levels. First, all

Figure 2.1: Loss distribution of a credit portfolio



obligors are more or less affected by the macro economic conditions. When the economy booms, obligors are generally more financially healthy, and they are more likely to fulfill their obligations. On the other hand, if the economy is in recession or even in depression, the chance that obligors will default is much higher. Second, many obligors belong to the same industry sector. If the entire industry sector develops significant financial difficulties, such as supply-demand problems or policy adjustment, all obligors in this industry sector will suffer simultaneously. Moreover, there might be a direct relationship between some obligors, such as a borrower-lender relationship. In this case, the default dependency between them may be very high. To model the correlations between different obligors, a multi-factor decomposition is often used. This is discussed in more detail in Section 3.2.

2.3 General Model Settings

In this thesis, we consider a portfolio consisting of N obligors. To avoid cumbersome notation, we assume that each obligor has only one loan. This assumption does not result in a loss of any generality, at least in the single period case. As mentioned above, in most cases, loans issued by the same obligor have the same credit rating. Moreover, if a credit event happens to any of the loans of this obligor, all loans of the obligor will experience the same credit migration. Therefore, we can consider all loans issued by the same obligor as being merged into one new loan. Also, we restrict our model to consider one-period $[t_0, t_1]$, where $t_1 - t_0 = 1$. This means that credit events can happen only at t_1. We build a portfolio at t_0, and keep the composition of the portfolio unchanged until t_1. If at t_1 no credit event happens, then there is no loss or gain due to credit changes. On the other hand, if a credit event happens at t_1, there will likely be an associated portfolio loss or gain.

Since this paper aims to study how to compute the loss distribution at the portfolio level, risk

attributes at the obligor level will be treated as input data to our model. One can consider this methodology as an analogy to an important principle in object-oriented programming (OOP): when viewing programs at the "high level", programmers concentrate on properties of objects and interactions between objects; the mechanisms and data structures inside objects being encapsulated are not considered until the programmer considers the "low level" implementation of the object.

These risk attributes are:

- $c = 0, \ldots, C-1$: Credit states, indicating risk ratings of obligors. $c = 0$ is associated with default, while $c = C-1$ represents the highest credit rating. If $C = 2$, then there are two credit states of an obligor: default ($c = 0$) or not default ($c = 1$).

- $P^c_{c(n)}$, $n = 0, \ldots, N-1, \quad c = 0, \ldots, C-1$: Probability that the obligor n will migrate from its current credit rating $c(n)$ at time t_0 to credit rating c at time t_1, which is obtained from credit migration matrices. Obligors with the same initial credit rating share the same probabilities of credit migration. In particular, $P^0_{c(n)}$ is the probability that obligor n will default at time t_1.

- EAD_n, $n = 0, \ldots, N-1$: Exposure-at-default of obligor n is the value of obligor n's loan at time t_1 given that this obligor's credit status is unchanged at time t_1;

- LGC^c_n, $n = 0, \ldots, N-1, \quad c = 0, \ldots, C-1$: Loss-given-credit-event of obligor n, which is a generalization of loss-given-default, indicating the percentage loss/gain of the loan value if obligor n's credit rating migrates from $c(n)$ at time t_0 to c at time t_1. Different obligors' LGCs may be different, even though their current credit rating may be the same. In particular, LGC^0_n represents the loss-given-default of the obligor n, which equals $1 - R_n$, where R_n is the recovery rate of obligor n. In our model, LGC^c_n are assumed to be deterministic: they can be estimated by the expectation of the percentage loss. Note that, if the credit rating of an obligor improves, then its loss-given-credit-event is negative. For convenience, from now on we use only "loss" instead of "loss/gain" with the understanding that a negative loss is a gain.

For obligor n, the loss associated with a credit event is a random variable:

$$\widetilde{\mathcal{L}}_n = \mathrm{EAD}_n \mathrm{LGC}^c_n \quad \text{with probability } P^c_{c(n)}.$$

Equivalently, $\widetilde{\mathcal{L}}_n$ can be expressed as

$$\widetilde{\mathcal{L}}_n = \mathrm{EAD}_n \sum_{c=0}^{C-1} \mathrm{LGC}^c_n \mathbb{I}^c_n, \tag{2.3.1}$$

where

$$\mathbb{I}_n^c = \begin{cases} 1, & \text{if obligor } n \text{ is in credit state } c \text{ at time } t_1, \\ 0, & \text{otherwise.} \end{cases}$$

Note that, for each obligor, $\mathbb{I}_n^{c_1}$ and $\mathbb{I}_n^{c_2}$ are not independent. Rather, they are negatively correlated with

$$\text{Corr}\left[\mathbb{I}_n^{c_1}, \mathbb{I}_n^{c_2}\right] = -\frac{1}{\sqrt{\left(\frac{1}{P_{c(n)}^{c_1}} - 1\right)\left(\frac{1}{P_{c(n)}^{c_2}} - 1\right)}},$$

since any obligor can end up in only one credit state. That is, if $\mathbb{I}_n^{c^*} = 1$, then $\mathbb{I}_n^c = 0, \forall c = 0, \ldots C-1$ for which $c \neq c^*$. $\text{Corr}\left[\mathbb{I}_n^{c_1}, \mathbb{I}_n^{c_2}\right]$ is within $[-1, 1]$ since $0 \leq P_{c(n)}^{c_1} + P_{c(n)}^{c_2} \leq 1$ implies

$$\left(\frac{1}{P_{c(n)}^{c_1}} - 1\right)\left(\frac{1}{P_{c(n)}^{c_2}} - 1\right) = \frac{1 - \left(P_{c(n)}^{c_1} + P_{c(n)}^{c_2}\right)}{P_{c(n)}^{c_1} P_{c(n)}^{c_2}} + 1 \geq 1.$$

Given each obligor's loss in (2.3.1), the portfolio loss is the sum of each loss:

$$\widetilde{\mathcal{L}} = \sum_{n=0}^{N-1} \widetilde{\mathcal{L}}_n.$$

Instead of modeling the dollar amount of the loss, it is often more useful to consider the rate of portfolio loss:

$$\begin{aligned} \mathcal{L} &= \frac{\sum_{n=0}^{N-1} \widetilde{\mathcal{L}}_n}{\sum_{n=0}^{N-1} \text{EAD}_n} \\ &= \sum_{n=0}^{N-1} \left[\left(\frac{\text{EAD}_n}{\sum_{k=0}^{N-1} \text{EAD}_k} \right) \sum_{c=0}^{C-1} \text{LGC}_n^c \mathbb{I}_n^c \right]. \end{aligned}$$

Note that coefficient $\text{EAD}_n / \sum_{k=0}^{N-1} \text{EAD}_k$ is the obligor's weight in the total portfolio exposure. These weights are the decision variables when we build a credit portfolio or perform risk analysis. Let $\omega_n = \text{EAD}_n / \sum_{k=0}^{N-1} \text{EAD}_k$, and $\mathcal{L}_n = \sum_{c=0}^{C-1} \text{LGC}_n^c \mathbb{I}_n^c$, then the portfolio loss rate can be written as

$$\mathcal{L} = \sum_{n=0}^{N-1} \omega_n \mathcal{L}_n. \tag{2.3.2}$$

Chapter 3

Multi-factor Structural Model

Structural models are used to analyze a firm's credit events based on assumptions about the firm's capital structure. This class of models can be traced back to Merton's paper [44], in which a firm is assumed to be financed only by equity and zero coupon bonds. By further assuming the dynamics of the firm's asset value to be a geometric Brownian motion, the value of the firm's equity and debt can be treated as options on the firm's assets. With this capital structure, a firm's default occurs when its asset value falls below its debt level, which can be modeled by a latent standard normal random variable falling below a certain threshold. Consequently, the default probability of a firm is equal to the probability that this latent variable falls below the threshold. Therefore, this class of models are also called latent variable models. Based on Merton's model, many industrial structural models have been developed, among which JP Morgan's CreditMetrics model is the most popular.

As mentioned in the previous chapter, when it comes to analyzing the credit risk of a portfolio consisting of loans issued by different firms (obligors), the marginal default probabilities computed from structural models are not enough to determine the joint loss probability of a portfolio, since the changes in credit status of obligors are not independent. Therefore, it is crucial to estimate the correlations of different obligors' asset returns for portfolio credit risk analysis. However, it is very inefficient to estimate the pairwise correlations directly. Consider a portfolio containing loans issued by N different obligors, then there are $N(N-1)/2$ parameters to estimate. It is common that a financial institution holds a very large portfolio containing thousands of obligors in order to diversify risk, thus the total number of correlations is staggering. Therefore, to estimate the correlations efficiently, a multi-factor decomposition is usually applied.

In this chapter, we first discuss Merton's model and the CreditMetrics model in Section 3.1, then

we introduce the multi-factor decomposition in Section 3.2. Applying the multi-factor structural model, we describe how to calculate the portfolio loss probability in a two-level computation and the challenges faced by traditional MC simulation for such a computation in Section 3.3.

3.1 Structural Models

3.1.1 Merton's Model

Merton's model assumes that a firm is financed only by equity, $S(t)$, and a single zero coupon bond, $D(t)$, with principle D and maturity T. Therefore, the asset value of a firm at any time $t \in [0, T]$ is

$$A(t) = S(t) + D(t).$$

Further, the model assumes that the asset value can be described by a geometric Brownian motion with drift μ_A and volatility σ_A

$$dA(t) = \mu_A A(t)dt + \sigma_A A(t)dB(t),$$

or equivalently

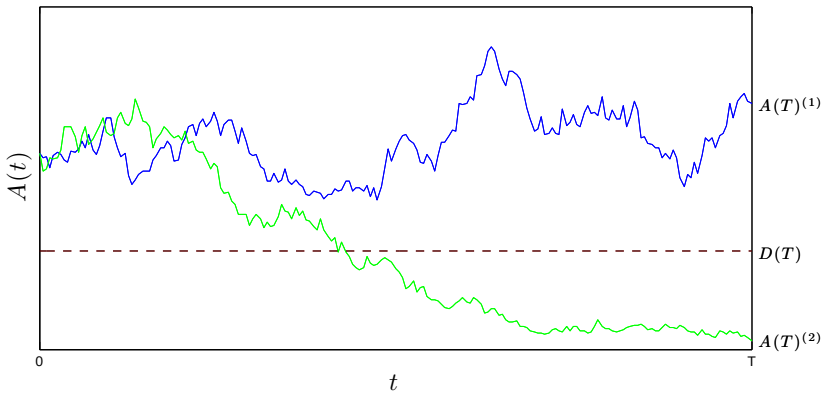
$$A(t) = A(0)e^{\left(\mu_A - \frac{1}{2}\sigma_A^2\right)t + \sigma_A B(t)}, \tag{3.1.1}$$

where $B(t)$ is a Brownian motion.

As shown in Fig. 3.1, two scenarios may happen at time T.

- Scenario 1: $A(T) \geq D$, the firm is able to pay the debt, and no default occurs (as shown by the blue line in Figure 3.1). In this scenario, debt holders of the firm get their principle back $D(T) = D$, while shareholders of the firm receive the residual $E(T) = A(T) - D$;
- Scenario 2: $A(T) < D$, the firm is not able to pay the debt, and default takes place (as shown by the green line in Figure 3.1). In this scenario, debt holders take over the firm, and receive the firm's total assets $D(T) = A(T)$, while shareholders of the firm receive nothing.

Figure 3.1: Merton's model: default vs. no default



Therefore, the payoff for the shareholders at the maturity T is $\max(A(T) - D, 0)$, which is the same as the payoff of a European call option on the firm's asset value $A(t)$ with maturity T and strike price D. Given the dynamics of $A(t)$ in (3.1.1) and assuming a constant risk-free interest rate r, the value of equity of the firm can be calculated by the Black-Scholes formula:

$$\begin{aligned} E(t) &= f_{BS}\left(A(t), \sigma_A, D, r, T\right) \\ &= A(t)\Phi\left(d_1\right) - De^{-r(T-t)}\Phi\left(d_2\right), \end{aligned} \tag{3.1.2}$$

where

$$\begin{aligned} d_1 &= \frac{\ln(A(t)/D) + \left(r + \sigma_A^2/2\right)(T-t)}{\sigma_A\sqrt{T-t}}, \\ d_2 &= d_1 - \sigma_A\sqrt{T-t}, \\ &= \frac{\ln(A(t)/D) + \left(r - \sigma_A^2/2\right)(T-t)}{\sigma_A\sqrt{T-t}} \end{aligned} \tag{3.1.3}$$

and $\Phi(x)$ is the cumulative distribution function (CDF) of a standard normal distribution.

Similarly, the payoff to the debt holders at maturity T is

$$\min(A(T), D) = D - \max(D - A(T), 0),$$

which equals the payoff of a risk-free bond with face value D and maturity T plus a short position in a European put option in the firm's asset value $A(t)$ with maturity T and strike price D. Therefore, according to the Black-Scholes formula, the value of the debt at time t is

$$D(t) = De^{-r(T-t)} - \left[De^{-r(T-t)}\Phi(-d_2) - A(t)\Phi(-d_1)\right].$$

We can also calculate a firm's default probability. As discussed above, in Merton's model, the event that a firm defaults is equivalent to the event $\{A(T) < D\}$, whence, if the dynamics of $A(t)$ are given by (3.1.1), then the default probability equals

$$\begin{aligned}\mathbb{P}\{A(T) < D\} &= \mathbb{P}\{\ln A(T) < \ln D\} \\ &= \mathbb{P}\left\{B(T) < \frac{\ln D - \ln A(0) - (\mu_A - \frac{1}{2}\sigma_A^2)T}{\sigma_A}\right\}.\end{aligned}$$

Define

$$\mathcal{Y} := B(T)/\sqrt{T}, \tag{3.1.4}$$

then $\mathcal{Y}$ is a standard normal random variable, since $B(T) \sim \mathcal{N}(0, T)$. Therefore,

$$\begin{aligned}\mathbb{P}\{A(T) < D\} &= \mathbb{P}\left\{\mathcal{Y} < \frac{\ln D - \ln A(0) - (\mu_A - \frac{1}{2}\sigma_A^2)T}{\sigma_A\sqrt{T}}\right\} \\ &= \Phi\left(\frac{\ln D - \ln A(0) - (\mu_A - \frac{1}{2}\sigma_A^2)T}{\sigma_A\sqrt{T}}\right) \\ &= \Phi\left(-\tilde{d}_2\right), \end{aligned} \tag{3.1.5}$$

where

$$\tilde{d}_2 = \frac{\ln(A(0)/D) + \left(\mu_A - \sigma_A^2/2\right)T}{\sigma_A\sqrt{T}}. \tag{3.1.6}$$

Note the difference between $\tilde{d}_2$ in (3.1.6) and d_2 in (3.1.3). In (3.1.3), we price the option in the "risk-neutral world", so the risk-free rate r is used. On the other hand, since the default probability in (3.1.5) is in the "real world", we use μ_A instead of r. Also notice that the random variable $\mathcal{Y}$ also has an economic interpretation. It is the normalized log-return of a firm's assets, and it can be considered to be the creditworthiness index of a firm. The creditworthiness is the only source of uncertainty in this model. When the creditworthiness increases, the obligor becomes more credit-worthy. On the other hand, if the creditworthiness decreases below a certain level, default takes place.

3.1.2 CreditMetrics Model

Although Merton's model is conceptually quite appealing, it suffers from a significant disadvantage in practice. It can handle only two credit states: default or no default. In practice, however, a firm's loan may have capital losses (gains) if the firm is downgraded (upgraded) by rating agencies. Merton's model ignores the migration risk. To address these issues, several industrial models have been developed to extend Merton's model. In this subsection, we review JP Morgan's CreditMetrics model.

The CreditMetrics model relies on a rating agency's credit rating system, as we discussed in the previous chapter, which specifies the credit rating of a firm and the probabilities that a firm migrates from one rating class to others in a given time interval. One strong assumption that the CreditMetrics model makes is that firms with the same credit rating have the same migration probabilities. This assumption is controversial. On the one hand, it simplifies the model, since financial institutions do not need to compute every firm's rating-migration probabilities. Instead, they can directly use rating-migration probabilities of corresponding rating classes offered by the rating agencies to approximate the firm's rating-migration probabilities, which significantly reduces the computation. However, this assumption ignores the heterogeneity of firms in the same rating class, and forces financial institutions to rely purely on the rating agencies' methodology and data. Crouhy et al. [12] mention that the rating agencies are slow to change their ratings. That is, some firms which should be downgraded or upgraded may remain in their current rating class for some time. Therefore, this assumption reduces the accuracy of rating-migration probabilities.

To incorporate migration risk into Merton's model, as shown in Figure 3.2 the CreditMetrics model defines a series of thresholds for each rating class in such a way that the rating-migration probabilities obtained from these thresholds match those in the credit migration matrix offered by a rating agency. Specifically, let $c = 0, \ldots, C-1$ represent the credit ratings, where $c = 0$ is associated with default, $c = C-1$ represents the highest credit rating, and denote the event that the credit rating of a firm with current rating c_1 migrates to rating c_2 at time T by $\mathrm{A}_{c_1}^{c_2}$. Define thresholds $H_{c_1}^{c_2}$, $c_1,\ c_2 = 0, \ldots, C-1$, such that

$$\mathbb{P}\left\{\mathrm{A}_{c_1}^{c_2}\right\} \quad = \quad \mathbb{P}\left\{H_{c_1}^{c_2-1} \leq \mathcal{Y} < H_{c_1}^{c_2}\right\} \text{ with } H_{c_1}^{-1} = -\infty, \tag{3.1.7}$$

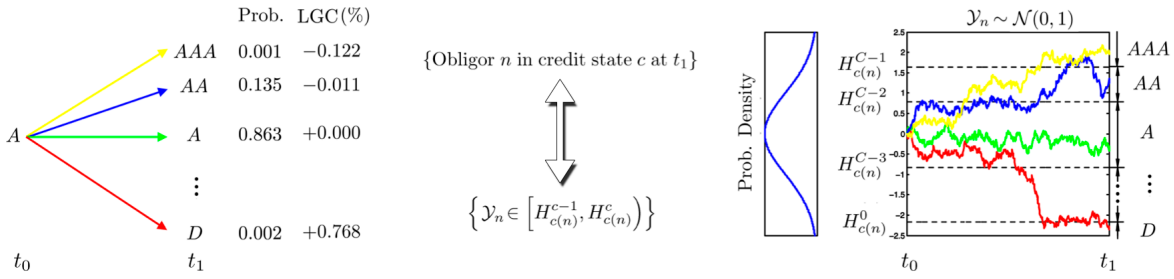
where $\mathcal{Y}$ is the creditworthiness index introduced in (3.1.4). If we let $P_{c_1}^{c_2}$ be the probability that the credit rating of a firm with current rating c_1 will migrate to rating c_2 at time T from the T-year credit-migration probability matrix in Table 2.2, then

$$\mathbb{P}\left\{H_{c_1}^{c_2-1} \leq \mathcal{Y} < H_{c_1}^{c_2}\right\} \quad = \quad P_{c_1}^{c_2}. \tag{3.1.8}$$

Since $\mathcal{Y} \sim \mathcal{N}(0,1)$, it follows that

$$H_{c_1}^{c_2} = \Phi^{-1}\left(\sum_{\zeta \leq c_2} P_{c_1}^{\zeta}\right).$$

Figure 3.2: CreditMetrics model



3.2 Multi-factor Decomposition

To incorporate correlations between obligors, Vasicek [58] suggested a decomposition of the individual credit worthiness index into an affine combination of the systematic risk factor $\mathcal{Z}$ and the idiosyncratic risk factor $\mathcal{E}_n$:

$$\mathcal{Y}_n = \beta_n \mathcal{Z} + \sqrt{1-\beta_n^2}\mathcal{E}_n \tag{3.2.1}$$

where $\mathcal{Z}$ and $\mathcal{E}_n$ are independent standard normal random variables.

The decomposition (3.2.1) is also known as the single-factor model. It divides the credit risk into two sources:

- Systematic risk $\mathcal{Z}$: This is a common risk to all obligors. It reflects the overall economic climate, and captures the impact of the macroeconomic environment on individuals.
- Idiosyncratic risk $\mathcal{E}_n$: This is obligor n's specific risk, which is neither dependent on the macroeconomic environment nor on other obligors' credit status.

The coefficient β_n is obligor-specific: it represents the sensitivity of obligor n to the macroeconomic environment. Also, to keep $\mathcal{Y}_n$ standard normal, the coefficients of the idiosyncratic risk factors are defined to be $\sqrt{1-\beta_n^2}$. In this model, β_n functions as a controller to adjust the proportions of risk coming from the systematic factor and from the idiosyncratic factor. Higher β_n implies more exposure to the systematic factor and less exposure to the idiosyncratic factor. Moreover, β_n is related to the correlation between obligors, since different obligors are correlated through the common systematic risk factor $\mathcal{Z}$:

$$\begin{aligned} \text{Corr}\left[\mathcal{Y}_i, \mathcal{Y}_j\right] &= \text{Corr}\left[\beta_i \mathcal{Z}, \beta_j \mathcal{Z}\right] \\ &= \beta_i \beta_j. \end{aligned} \tag{3.2.2}$$

The single-factor model is often used to model correlations, but it has several severe shortcomings.

As we mentioned in Section 2.2, obligors may belong to different industrial/regional sectors, and each sector has its own risk structure. Therefore, a model with a common systematic factor is too simplistic to capture the characteristics of losses in portfolios with an unequal distribution of risks from different sectors. As a remedy, assume that there are K sectors in the portfolio, and that obligor n belongs to one sector $k(n)$ only, then one can apply a finer decomposition to the creditworthiness index:

$$\begin{aligned} \mathcal{Y}_n &= \alpha_n \mathcal{X}_{k(n)} + \sqrt{1-\alpha_n^2}\mathcal{E}_n, \\ \mathcal{X}_k &= \sum_{s=0}^{S-1} \gamma_{k,s}\mathcal{Z}_s, \quad k = 0, \ldots, K-1, \end{aligned} \tag{3.2.3}$$

where $\sum_{s=0}^{S-1} \gamma_{k,s}^2 = 1$, and the components of the vector $[\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{E}}]^T = [\mathcal{Z}_1, \ldots, \mathcal{Z}_S, \mathcal{E}_1, \ldots, \mathcal{E}_N]^T$ are independent standard normal random variables.

In this multi-factor model, there are K sectorial credit drivers $\{\mathcal{X}_k\}$, created by taking affine combinations of S systematic risk factors $\{\mathcal{Z}_s\}$. Note that $\mathcal{X}_k \sim N(0,1)$ since it is an affine combination of independent standard normal random variables $\{\mathcal{Z}_s\}$ with weights $\gamma_{k,s}$ satisfying $\sum_{s=0}^{S-1} \gamma_{k,s}^2 = 1$. The sectorial drivers are characterized by the factor loadings, $\{\gamma_{\cdot,s}\}$, and they represent sector-specific sensitivities to systematic risk factors. Therefore, each sector has its unique credit driver, and different credit risk profiles across sectors can be handled.

Within the same sector k, an obligor's credit worthiness index depends only on the sector's risk driver, $\mathcal{X}_k$, and the obligor's idiosyncratic risk factor, $\mathcal{E}_n$. Thus, a multi-factor model creates a new sectorial level between the portfolio level and the obligor level. At the portfolio level, there are several systematic risk factors $\{\mathcal{Z}_s\}$, but at each sectorial level, it is just a single-factor model with different sector risk factors $\mathcal{X}_k$. Since each sector has its own sector-specific risk driver, sectorial credit drivers are a better choice than the common risk factor in the single-factor model.

The correlations embedded in the multi-factor model are more complicated than in the single-factor model. Unlike a single-factor model, a multi-factor model correlates obligors through multiple systematic factors $\{\mathcal{Z}_s\}$:

$$\begin{aligned} \text{Corr}\left[\mathcal{Y}_i, \mathcal{Y}_j\right] &= \text{Corr}\left[\alpha_i \mathcal{X}_{k(i)}, \alpha_j \mathcal{X}_{k(j)}\right] \\ &= \alpha_i \alpha_j \text{Corr}\left[\sum_{s=0}^{S-1} \gamma_{k(i),s}\mathcal{Z}_s, \sum_{s=0}^{S-1} \gamma_{k(j),s}\mathcal{Z}_s\right] \\ &= \alpha_i \alpha_j \sum_{s=0}^{S-1} \gamma_{k(i),s}\gamma_{k(j),s}. \end{aligned} \tag{3.2.4}$$

Comparing (3.2.2) and (3.2.4), we see that the multi-factor model can accommodate a more complex

correlation structure than the single-factor model.

Several industry multi-factor correlation models have been developed based on the multi-factor model (3.2.3), among which CreditMetrics Multi-factor Correlation Model [22] and KMV's Global Correlation Model$^{\text{TM}}$ [6, 12] are the most popular. The two methods differ mainly in their calibration. Since the asset value of a company is not observable in the market, the calibration is not straightforward. The CreditMetrics model uses the correlation between equity returns as a proxy for calculating correlations of asset returns, while the KMV model applies a more sophisticated iterative procedure to calibrate asset correlations using asset return data. More details on the calibration of multi-factor models can be found in [6, 12, 22, 24].

3.3 Computational Challenges for the Multi-factor Structural Model

If we apply the CreditMetrics model, then the individual loss $\mathcal{L}_n = \sum_{c=0}^{C-1} \text{LGC}_n^c \mathbb{I}_n^c$ in (2.3.2) changes to

$$\mathcal{L}_n = \sum_{c=0}^{C-1} \text{LGC}_n^c \mathbb{I}_{\left\{H_{c(n)}^{c-1} \le \mathcal{Y}_n < H_{c(n)}^{c}\right\}}, \tag{3.3.1}$$

where $H_{c(n)}^c = \Phi^{-1}\left(\sum_{\gamma \le c} P_{c(n)}^{\gamma}\right)$. The multi-factor decomposition in (3.2.3) can be written as

$$\mathcal{Y}_n = \boldsymbol{\beta}_n^T \boldsymbol{\mathcal{Z}} + \sqrt{1 - \boldsymbol{\beta}_n^T \boldsymbol{\beta}_n}\mathcal{E}_n, \tag{3.3.2}$$

where $\boldsymbol{\beta}_n = (\beta_{1n}, \ldots, \beta_{Sn})^T$ and $\beta_{sn} = \alpha_n \gamma_{k(n),s}$ is obligor n's sensitivity to the systematic risk factor s. Substituting (3.3.2) into (3.3.1) and re-arranging terms gives

$$\mathcal{L}_n = \sum_{c=0}^{C-1} \text{LGC}_n^c \mathbb{I}_{\left\{\frac{H_{c(n)}^{c-1} - \boldsymbol{\beta}_n^T \boldsymbol{\mathcal{Z}}}{\sqrt{1-\boldsymbol{\beta}_n^T \boldsymbol{\beta}_n}} \le \mathcal{E}_n < \frac{H_{c(n)}^{c} - \boldsymbol{\beta}_n^T \boldsymbol{\mathcal{Z}}}{\sqrt{1-\boldsymbol{\beta}_n^T \boldsymbol{\beta}_n}}\right\}},$$

and the portfolio loss rate (2.3.2) is extended to

$$\mathcal{L} \quad = \quad L^{(N)}(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{E}}) = \sum_{n=0}^{N-1} \widetilde{L}_n(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{E}}),$$

where

$$\begin{aligned}\widetilde{L}_n(\boldsymbol{\mathcal{Z}},\boldsymbol{\mathcal{E}}) &= \omega_n L_n(\boldsymbol{\mathcal{Z}},\boldsymbol{\mathcal{E}}),\\ L_n(\boldsymbol{\mathcal{Z}},\boldsymbol{\mathcal{E}}) &= \mathcal{L}_n = \sum_{c=0}^{C-1} \mathrm{LGC}_n^c \mathbb{I}_{\left\{\frac{H_{c(n)}^{c-1}-\boldsymbol{\beta}_n^T\boldsymbol{\mathcal{Z}}}{\sqrt{1-\boldsymbol{\beta}_n^T\boldsymbol{\beta}_n}} \le \mathcal{E}_n < \frac{H_{c(n)}^{c}-\boldsymbol{\beta}_n^T\boldsymbol{\mathcal{Z}}}{\sqrt{1-\boldsymbol{\beta}_n^T\boldsymbol{\beta}_n}}\right\}}, \end{aligned} \tag{3.3.3}$$

with $H_{c(n)}^{-1} = -\infty$.

Given a quantile l, the loss probability of a portfolio can be computed as

$$\begin{aligned}\mathbb{P}\left\{L^{(N)}(\boldsymbol{\mathcal{Z}},\boldsymbol{\mathcal{E}}) \le l\right\} &= \mathbb{E}\left[\mathbb{P}\left\{L^{(N)}(\boldsymbol{\mathcal{Z}},\boldsymbol{\mathcal{E}}) \le l \middle| \boldsymbol{\mathcal{Z}}\right\}\right]\\ &= \int_{\mathbb{R}^S} \mathbb{P}\left\{L^{(N)}(\boldsymbol{z},\boldsymbol{\mathcal{E}}) \le l\right\} d\Phi_S(\boldsymbol{z}). \end{aligned} \tag{3.3.4}$$

The computation of (3.3.4) can be divided into two levels: an outer-level integration and an inner-level conditional loss probability calculation. For the outer-level integration, if there are a few (e.g. one to three) systematic risk factors only, then we can use a quadrature method to approximate the integral. If the number of systematic risk factors, S, is more than a few but not many (e.g. four or five), then possibly sparse grid integration techniques may be effective. However, S is usually large. Hence, standard quadrature techniques (and possibly even sparse-grid methods) would require too much computation. In practice, Quasi Monte Carlo (QMC) simulation is usually applied to generate U realizations of the systematic risk factor $\boldsymbol{\mathcal{Z}}^{(u)}$, $u = 1, \ldots, U$, and the loss probability is approximated by

$$\begin{aligned}\mathbb{P}\left\{L^{(N)}(\boldsymbol{\mathcal{Z}},\boldsymbol{\mathcal{E}}) \le l\right\} &= \mathbb{E}\left[\mathbb{P}\left\{L^{(N)}(\boldsymbol{\mathcal{Z}},\boldsymbol{\mathcal{E}}) \le l \middle| \boldsymbol{\mathcal{Z}}\right\}\right]\\ &\approx \frac{1}{U}\sum_{u=1}^{U} \mathbb{P}\left\{L^{(N)}\left(\boldsymbol{\mathcal{Z}},\boldsymbol{\mathcal{E}}\right) \le l \middle| \boldsymbol{\mathcal{Z}} = \boldsymbol{\mathcal{Z}}^{(u)}\right\}. \end{aligned} \tag{3.3.5}$$

For the inner-level, we must compute the conditional probability

$$\mathbb{P}\left\{L^{(N)}\left(\boldsymbol{z},\boldsymbol{\mathcal{E}}\right) \le l\right\} = \mathbb{P}\left\{L^{(N)}\left(\boldsymbol{\mathcal{Z}},\boldsymbol{\mathcal{E}}\right) \le l \middle| \boldsymbol{\mathcal{Z}} = \boldsymbol{z}\right\}.$$

In practice, a MC simulation is often used. For each realization of $\boldsymbol{\mathcal{Z}} = \boldsymbol{\mathcal{Z}}^{(u)}$, one samples from the multivariate normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$ to obtain observations of the individual risk drivers $\boldsymbol{\mathcal{E}}^{(u,v)}$, $v =$

$1, \ldots, V$, then the conditional loss probability can be estimated by

$$\begin{aligned} \mathbb{P}\left\{L^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}, \boldsymbol{\mathcal{E}}\right) \leq l\right\} &= \mathbb{E}\left[\mathbb{I}_{\left\{L^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}, \boldsymbol{\mathcal{E}}\right) \leq l\right\}}\right] \\ &\approx \frac{1}{V}\sum_{v=1}^{V} \mathbb{I}_{\left\{L^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}, \boldsymbol{\mathcal{E}}^{(u,v)}\right) \leq l\right\}}, \end{aligned} \tag{3.3.6}$$

where

$$L^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}, \boldsymbol{\mathcal{E}}^{(u,v)}\right) = \sum_{n=0}^{N-1} \omega_n \left(\sum_{c=0}^{C-1} \mathrm{LGC}_n^c \mathbb{I}_{\left\{ \frac{H_{c(n)}^{c-1} - \boldsymbol{\beta}_n^T \boldsymbol{\mathcal{Z}}^{(u)}}{\sqrt{1-\boldsymbol{\beta}_n^T \boldsymbol{\beta}_n}} \leq \mathcal{E}_n^{(u.v)} < \frac{H_{c(n)}^{c} - \boldsymbol{\beta}_n^T \boldsymbol{\mathcal{Z}}^{(u)}}{\sqrt{1-\boldsymbol{\beta}_n^T \boldsymbol{\beta}_n}} \right\}} \right). \tag{3.3.7}$$

Substituting (3.3.6) into (3.3.5), the unconditional loss probability is estimated by the unbiased estimator

$$\mathbb{P}\left\{L^{(N)}(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{E}}) \leq l\right\} \approx \frac{1}{UV}\sum_{u=1}^{U}\sum_{v=1}^{V} \mathbb{I}_{\left\{L^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}, \boldsymbol{\mathcal{E}}^{(u,v)}\right) \leq l\right\}}. \tag{3.3.8}$$

The simulation-based approximation is based on MC simulation, which is very computationally intensive. Since this is a two-level simulation, we need to generate $U \cdot V$ scenarios. For each realization of the systematic risk factor $\boldsymbol{\mathcal{Z}} = \boldsymbol{\mathcal{Z}}^{(u)}$, $V \cdot N$ independent random numbers must be generated to approximate one conditional loss probability, whence $U \cdot (V \cdot N + S)$ independent random numbers are required to approximate one unconditional loss probability. Also, to satisfy the accuracy requirement, we must either let both U and V be very large or solve an optimization problem to choose appropriate U and V. (See [25] for a discussion of the sample-size allocating problem.) Meanwhile, given realizations of random numbers, the computational cost of calculating the portfolio loss $L^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}, \boldsymbol{\mathcal{E}}^{(u,v)}\right)$ increases with the portfolio size N. When the number of obligors in the portfolio is very large, MC simulation could be very computationally intensive. To accelerate the computation, Glasserman et al. [18, 19] developed an importance sampling technique for both the inner-level and outer-level simulations to capture rare large losses more efficiently. However, even with this acceleration, MC simulation is still very computationally intensive.

Chapter 4

Asymptotic Approximation

One intriguing feature of the multi-factor structural model is that, conditional on the sectorial (systematic) risk factors, the losses from the individual obligors' loans are independent since the obligors' individual risk factors $\mathcal{E}_n$ are independent. Together with the huge size of some portfolios, this conditional independence gives rise to a class of approximations to the conditional portfolio loss probabilities, based on the asymptotic behavior of the conditional losses of large portfolios. In this chapter, we first review the existing asymptotic approximation based on the law of large number (LLN) in Section 4.1, then we introduce a new asymptotic approximation based on the central limit theorem (CLT) in Section 4.2. For a special class of portfolios, which we call "lumpy portfolios", we develop a hybrid approximation which combines the asymptotic approximation (CLT or LLN) and the MC approximation in Section 6.1. Numerical results which compare these different approximations are presented in Section 4.3.

To begin, we make the following realistic assumptions on our model parameters:

1. $\exists\ \mathrm{LGC}_{max} < \infty$ such that $\forall\ n \in \{0, \ldots, N-1\}$, $\forall\ c \in \{0, \ldots, C-1\}$, $|\mathrm{LGC}_n^c| \leq \mathrm{LGC}_{max}$;

2. $\exists\ \mathrm{EAD}_{max} < \infty$ such that $\forall\ n \in \{0, \ldots, N-1\}$, $|\mathrm{EAD}_n| \leq \mathrm{EAD}_{max}$;

3. $\forall\ n \in \{0, \ldots, N-1\}$, $\left|\mathrm{LGC}_n^i - \mathrm{LGC}_n^j\right| \geq \xi > 0$ if $i \neq j$, and LGC_n^c satisfy

$$\begin{cases} \mathrm{LGC}_n^c > 0,\ c(n) > c, \\ \mathrm{LGC}_n^c = 0,\ c(n) = c, \\ \mathrm{LGC}_n^c < 0,\ c(n) < c; \end{cases}$$

4. $\forall\ n \in \{0, \ldots, N-1\}$, $c(n) \neq 0$;

5. $\exists\, P_{max} \in [0,1)$ such that $\forall\ n \in \{0,\dots,N-1\}$, $\forall\ c(n) \in \{1,\dots,C-1\}$, $\forall\ c \in \{0,\dots,C-1\}$, $0 \le P^c_{c(n)} \le P_{max}$;

6. $\exists\, \beta_{max} \in [0,1)$ such that $\forall\ n \in \{0,\dots,N-1\}$, $\boldsymbol{\beta}_n^T \boldsymbol{\beta}_n \le \beta_{max}$;

7. $\sum_{n=0}^{N-1} \mathrm{EAD}_n \neq 0$.

Assumptions 1 and 2 are required to prove some of the mathematical results we state later; they are always satisfied in practice. The third assumption guarantees that a change of credit rating will impact the value of a loan. More specifically, downgrading (upgrading) will decrease (increase) a loan's value. The fourth condition indicates that each obligor is not in the default state at time $t = t_0$. The fifth assumption ensures that any non-default obligor that is in credit state $c(n) \neq 0$ at time $t = t_0$ will not be in a certain credit state c at time $t = t_1$ almost surely (i.e. $P^c_{c(n)} \neq 1$ for any $c \in \{0, 1, \dots C-1\}$). If this were not the case, the loss to this obligor at t_1 would be deterministic, whence we would not need this probabilistic model to calculate it. The sixth condition guarantees the validity of the decomposition of the creditworthiness index $\mathcal{Y}_n$ in (3.3.2). Moreover, the assumption that $\beta_{max} < 1$ is reasonable since no obligor can be perfectly correlated to one systematic risk factor or a combination of several systematic risk factors. The last condition ensures that the weights $\omega_n = \mathrm{EAD}_n / \sum_{n=0}^{N-1} \mathrm{EAD}_n$ are meaningful.

4.1 The LLN Approximation

In 2003, Gordy wrote a seminal paper [20], in which he studied the asymptotic behavior of portfolio losses conditional on the systematic risk factors. He proved that, if the portfolio is infinitely fine-grained, that is, no obligor's exposure exceeds an arbitrarily small portion of the total portfolio exposure, then, conditional on the sectorial risk factors, the portfolio loss converges to its expectation. Specifically, Gordy proves that, under the following assumptions

1. $\exists\, \delta > 0$ such that $\sup_n \{|\omega_n|\} = O\left(N^{-(1/2+\delta)}\right)$,

2. $\sum_{n=0}^{N-1} \mathrm{EAD}_n \to \infty$ as $N \to \infty$,

the LLN can be applied, and that the conditional portfolio loss converges to its expectation as N tends to infinity:

$$L^{(N)}\left(\boldsymbol{z}, \boldsymbol{\mathcal{E}}\right) \xrightarrow{a.s.} \mathbb{E}\left[L^{(N)}(\boldsymbol{z}, \boldsymbol{\mathcal{E}})\right], \text{ as } N \to \infty.$$

Therefore, denoting $\mathbb{E}\left[L^{(N)}(\boldsymbol{z}, \boldsymbol{\mathcal{E}})\right]$ by $\mu^{(N)}(\boldsymbol{z})$, the conditional loss probability can be approximated by

$$\mathbb{P}\left\{L^{(N)}\left(\boldsymbol{z}, \boldsymbol{\mathcal{E}}\right) \le l\right\} \approx \mathbb{I}_{\left\{\mu^{(N)}(\boldsymbol{z}) \le l\right\}}, \tag{4.1.1}$$

and the unconditional loss probability can be approximated by

$$\mathbb{P}\left\{L^{(N)}(\boldsymbol{\mathcal{Z}},\boldsymbol{\mathcal{E}}) \le l\right\} \approx \frac{1}{U}\sum_{u=1}^{U} \mathbb{I}_{\left\{\mu^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)\le l\right\}}.$$

The beauty of the LLN approximation lies in its simplicity. Compared with the MC approximation, the computational work is substantially reduced. Only U scenarios and $U \cdot S$ random numbers need to be generated, and the inner-level simulation is simply replaced by the evaluation of a Heaviside step function. Numerical results show that the LLN approximation works well for fine-grained homogeneous portfolios. However, the difference between the LLN limiting distribution and the true distribution can be significant if the portfolio is either coarse-grained or heterogeneous (see Han [23]). To improve the LLN approximation, granularity adjustments for portfolio VaR and ES have been studied by Gordy [20, 21], Wilde [60], Martin and Wilde [40] for the single-factor setting, and by Pykhtin [51] for the multi-factor setting. However, a significant amount of extra computation is required by all of these granularity adjustments, especially for the multi-factor setting.

4.2 The CLT Approximation

To achieve better accuracy than the LLN approximation, without increasing the computational work significantly, several researchers [36, 52] have mentioned that the central limit theorem (CLT) can be applied to approximate the conditional loss, but the convergence of this asymptotic approximation has not been established. In this section, we prove the convergence based on the Linderberg's version of CLT. Also, the error incurred in this approximation is examined.

4.2.1 The Approximation and Convergence

Like Gordy [20], we utilize the fact that, conditional on the realization of the systematic risk factor $\boldsymbol{\mathcal{Z}} = \boldsymbol{z}$, the individual risk factors, $\mathcal{E}_n$, $n = 0, \ldots, N-1$, are mutually independent. Hence the conditional loss is the sum of N independent random variables $\widetilde{L}_n(\boldsymbol{z}, \boldsymbol{\mathcal{E}})$:

$$L^{(N)}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}) \quad = \quad \sum_{n=0}^{N-1} \widetilde{L}_n(\boldsymbol{z}, \boldsymbol{\mathcal{E}}).$$

If $\widetilde{L}_n(\boldsymbol{z}, \boldsymbol{\mathcal{E}})$ were identically distributed, the classic CLT would be enough to show that

$$\frac{L^{(N)}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}) - \mu^{(N)}(\boldsymbol{z})}{\sigma^{(N)}(\boldsymbol{z})} \xrightarrow{d} \mathcal{N}(0,1).$$

However, a difficulty here is that the $\widetilde{L}_n(\boldsymbol{z}, \boldsymbol{\mathcal{E}})$ are not identically distributed. Nevertheless, we are able to show that, under conditions similar to those used by Gordy [20], Linderberg's version of the CLT guarantees that the normalized conditional loss converges to a standard normal random variable in distribution. This result is stated more precisely in the following theorem.

Theorem 4.1. *Conditional on $\boldsymbol{\mathcal{Z}} = \boldsymbol{z} < \infty$, if $\exists\, \delta > 0$ such that $\sup_n \{|\omega_n|\} = O\left(N^{-(1/2+\delta)}\right)$, then the normalized conditional portfolio loss converges in distribution to a standard normal random variable:*

$$\frac{L^{(N)}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}) - \mu^{(N)}(\boldsymbol{z})}{\sigma^{(N)}(\boldsymbol{z})} \xrightarrow{d} \mathcal{N}(0,1),$$

as $N \to \infty$, where $\mu^{(N)}(\boldsymbol{z}) = \mathbb{E}\left[L^{(N)}(\boldsymbol{z}, \boldsymbol{\mathcal{E}})\right]$ and $\sigma^{(N)}(\boldsymbol{z}) = \sqrt{\mathbb{V}\left[L^{(N)}(\boldsymbol{z}, \boldsymbol{\mathcal{E}})\right]}$.

A proof of this theorem, based on Linderberg's version of the CLT, is given in Appendix A. As a consequence of this theorem, given any loss quantile l, the loss probability conditional on $\boldsymbol{\mathcal{Z}} = \boldsymbol{z}$ satisfies

$$\begin{aligned} & \mathbb{P}\left\{L^{(N)}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}) \leq l\right\} - \Phi\left(\frac{l - \mu^{(N)}(\boldsymbol{z})}{\sigma^{(N)}(\boldsymbol{z})}\right) \\ = \quad & \mathbb{P}\left\{\frac{L^{(N)}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}) - \mu^{(N)}(\boldsymbol{z})}{\sigma^{(N)}(\boldsymbol{z})} \leq \frac{l - \mu^{(N)}(\boldsymbol{z})}{\sigma^{(N)}(\boldsymbol{z})}\right\} - \Phi\left(\frac{l - \mu^{(N)}(\boldsymbol{z})}{\sigma^{(N)}(\boldsymbol{z})}\right) \\ \to \quad & 0 \end{aligned}$$

for all $\boldsymbol{z} \in \mathbb{D}^S$ as $N \to \infty$.

The condition $\sup_n \{|\omega_n|\} = O\left(N^{-(1/2+\delta)}\right)$ in Theorem 4.1 on the weights ω_n is the same as condition 1 for Gordy's LLN approximation. This condition can be considered as a granularity condition: it guarantees that the largest weight vanishes with a rate of $O\left(N^{-(1/2+\delta)}\right)$ as $N \to +\infty$. This condition also implies that the portfolio's Herfindahl-Hirschman Index (HHI), defined by $\text{HHI} = \sum_{n=0}^{N-1} \omega_n^2$, which is the most commonly used empirical measure for the name concentration of a portfolio, tends to zero with at least a rate of $O\left(N^{-2\delta}\right)$. In practice, this is not a very strong restriction for large financial institutions, since portfolios held by most large financial institutions are well-diversified.

Consequently, the following theorem shows that, as the condition in Theorem 4.1 is satisfied, then the unconditional loss probability converges to the CLT limiting unconditional probability.

Theorem 4.2. *If $\exists\, \delta > 0$ such that $\sup_n \{|\omega_n|\} = O\left(N^{-(1/2+\delta)}\right)$, then*

$$\mathbb{P}\left\{L^{(N)}(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{E}}) < l\right\} - \int_{\mathbb{R}^S} \Phi\left(\frac{l - \mu^{(N)}(\boldsymbol{z})}{\sigma^{(N)}(\boldsymbol{z})}\right) d\Phi_S(\boldsymbol{z}) \to 0$$

for all $l \in \mathbb{R}$ as $N \to \infty$, where $\Phi_S(\boldsymbol{z})$ is the cumulative probability function of the $S-$dimensional multivariate normal distribution.

A proof of this theorem, based on the Dominated Convergence Theorem, is given in Appendix B.

As a consequence of Theorem 4.2, given any loss quantile l, the loss probability can be approximated by

$$\mathbb{P}\left\{L^{(N)}(\boldsymbol{\mathcal{Z}},\boldsymbol{\mathcal{E}}) \le l\right\} \approx \int_{\mathbb{R}^S} \Phi\left(\frac{l-\mu^{(N)}(\boldsymbol{z})}{\sigma^{(N)}(\boldsymbol{z})}\right) d\Phi_S(\boldsymbol{z}), \tag{4.2.1}$$

and, if MC simulation is applied to estimate the integral above, then

$$\mathbb{P}\left\{L^{(N)}(\boldsymbol{\mathcal{Z}},\boldsymbol{\mathcal{E}}) \le l\right\} \approx \frac{1}{U}\sum_{u=1}^{U} \Phi\left(\frac{l-\mu^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)}{\sigma^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)}\right). \tag{4.2.2}$$

Theorems 4.1 and 4.2 reveal some interesting points about the asymptotic behavior of the portfolio loss. The LLN approximation assumes that all idiosyncratic risks are diversified away. However, the CLT approximation

$$\mathbb{P}\left\{ L^{(N)}(\boldsymbol{\mathcal{Z}},\boldsymbol{\mathcal{E}}) \le l \,\middle|\, \boldsymbol{\mathcal{Z}} = \boldsymbol{z}\right\} \approx \Phi\left(\frac{l-\mu^{(N)}(\boldsymbol{z})}{\sigma^{(N)}(\boldsymbol{z})}\right) \tag{4.2.3}$$

does not ignore all idiosyncratic risks. The conditional variance $\sigma^{(N)}(\boldsymbol{z})$ is matched exactly in the CLT limiting distribution, therefore, both systematic risk factors and idiosyncratic risk factors are taken into consideration, which leads to a more accurate approximation. On the other hand, the computational work required by the CLT approximation is comparable to that required by the LLN approximation: only U scenarios and $U \cdot S$ random numbers in total need to be generated. The only significant extra computational work required by the CLT approximation, compared to the LLN approximation, is to calculate the conditional variance $\left(\sigma^{(N)}(\boldsymbol{z})\right)^2$ and the normal CDF $\Phi\left(\frac{l-\mu^{(N)}(\boldsymbol{z})}{\sigma^{(N)}(\boldsymbol{z})}\right)$; both the LLN and CLT approximations require the computation of $\mu^{(N)}(\boldsymbol{z})$.

4.2.2 Error Analysis of the CLT Approximation

The CLT approximation (4.2.3) is a limiting probability, while in reality a financial institution's portfolio consists of finitely many obligors. Therefore, there is an asymptotic error associated with (4.2.1). More precisely,

$$\mathbb{P}\left\{L^{(N)}(\boldsymbol{\mathcal{Z}},\boldsymbol{\mathcal{E}}) \le l\right\} = \int_{\mathbb{R}^S} \Phi\left(\frac{l-\mu^{(N)}(\boldsymbol{z})}{\sigma^{(N)}(\boldsymbol{z})}\right) d\Phi_S(\boldsymbol{z}) + \Delta(l),$$

where

$$\begin{aligned}\Delta(l) &= \int_{\mathbb{R}^S} \delta(l, \boldsymbol{z})\, d\Phi_S(\boldsymbol{z}), \\ \delta(l, \boldsymbol{z}) &= \mathbb{P}\left\{L^{(N)}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}) \leq l\right\} - \Phi\left(\frac{l - \mu^{(N)}(\boldsymbol{z})}{\sigma^{(N)}(\boldsymbol{z})}\right). \end{aligned} \tag{4.2.4}$$

The key to analyze the difference, $\Delta(l)$, between the CLT limiting loss probability and the true loss probability is the difference between their conditional probabilities, $\delta(l, \boldsymbol{z})$. Many researchers have worked on bounds for the difference between the CLT limiting distribution function and the true distribution function for a sum of independent random variables. We briefly review some of that work.

Let $\mathcal{X}_1, \ldots, \mathcal{X}_N$ be independent random variables with zero means and finite variance. Define

$$\mathcal{S}^{(N)} = \sum_{n=0}^{N-1} \mathcal{X}_n,$$

and assume

$$\mathbb{V}\left[\mathcal{S}^{(N)}\right] = \sum_{n=0}^{N-1} \mathbb{E}\left[\mathcal{X}_n^2\right] = 1.$$

Let $F^{(N)}(x)$ be the CDF of $\mathcal{S}^{(N)}$. Then the CLT states that

$$F^{(N)}(x) \to \Phi(x), \ \ as\ N \to \infty.$$

Assume $\mathbb{E}\left[|\mathcal{X}_n|^3\right] < \infty$ for all $n = 1, 2, \ldots, N$, and let

$$\rho^{(N)} = \sum_{n=0}^{N-1} \mathbb{E}\left[|\mathcal{X}_n|^3\right].$$

Berry [4] and Esseen [14] showed that, if $\mathcal{X}_n$ are identically distributed, then there is a uniform bound:

$$\left|F^{(N)}(x) - \Phi(x)\right| \leq C_0 \rho^{(N)}, \tag{4.2.5}$$

where C_0 is some positive constant. In 1986, Siganov [55] showed that the constant C_0 can be taken to be 0.7655; this was improved to 0.7164 by Chen [9] in 2002. Without the assumption that $\mathcal{X}_n$ are identically distributed, (4.2.5) still holds, but the constant C_0 may be larger. In 1972, von Beek [59] showed that C_0 is at most 0.7975; Siganov [55] improved this to 0.7915 in 1986. Also, there is a non-uniform version

of (4.2.5),

$$\left|F^{(N)}(x) - \Phi(x)\right| \leq C_1 \frac{\rho^{(N)}}{1+|x|^3}, \tag{4.2.6}$$

which was first proved by Nagaev [46] in 1965 for identically distributed $\{\mathcal{X}_n\}$. This bound was generalized to the non-identical case by Bikelis [5] in 1966. Michel [45] showed that C_1 can be taken to be 30.84 for the identical case, while, for the non-identical case, Paditz [48, 49] first calculated C_1 to be at most 114.7 in 1977 and improved C_1 to 31.395 in 1989.

There are similar bounds under less restrictive assumptions than $\mathbb{E}\left[|\mathcal{X}_n|^3\right] < \infty$. However, this assumption always holds for our problem. Hence we do not review these more general bounds in this paper.

In our problem,

$$\mathcal{X}_n = \frac{\widetilde{L}_n(\boldsymbol{z}, \boldsymbol{\mathcal{E}}) - \widetilde{\mu}_n(\boldsymbol{z})}{\sigma^{(N)}(\boldsymbol{z})},$$

and

$$\mathcal{S}^{(N)} = \sum_{n=0}^{N-1} \mathcal{X}_n = \frac{L^{(N)}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}) - \mu^{(N)}(\boldsymbol{z})}{\sigma^{(N)}(\boldsymbol{z})},$$

where $\widetilde{\mu}_n(\boldsymbol{z}) = \omega_n \mathbb{E}\left[L_n\left(\boldsymbol{z}, \boldsymbol{\mathcal{E}}\right)\right]$. Therefore,

$$\left|\delta\left(l, \boldsymbol{z}\right)\right| \quad \leq \quad B\left(\frac{l - \mu^{(N)}(\boldsymbol{z})}{\sigma^{(N)}(\boldsymbol{z})}\right), \tag{4.2.7}$$

where $B(x)$ is the left side of (4.2.5) or (4.2.6). To make the bound tighter, one can even choose $B(x)$ to be the minimum of (4.2.5) and (4.2.6). Consequently, the difference between the CLT limiting loss probability and the true loss probability, $\Delta(l)$, satisfies:

$$\Delta(l) \leq \int_{\mathbb{R}^S} B\left(\frac{l - \mu^{(N)}(\boldsymbol{z})}{\sigma^{(N)}(\boldsymbol{z})}\right) d\Phi_S(\boldsymbol{z}). \tag{4.2.8}$$

However, if the simulation-based method is used to approximate the integral in (4.2.1), as in (4.2.2), the error structure is more complicated. First, there is a sampling error due to the simulation, which can be written as

$$\int_{\mathbb{R}^S} \mathbb{P}\left\{L^{(N)}\left(\boldsymbol{z}, \boldsymbol{\mathcal{E}}\right) \leq l\right\} d\Phi_S(\boldsymbol{z}) - \frac{1}{U}\sum_{u=1}^{U} \mathbb{P}\left\{L^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}, \boldsymbol{\mathcal{E}}\right) \leq l\right\}.$$

This type of error is usually monitored by computing the associated confidence interval. Secondly, there is an asymptotic error due to the CLT approximation, $\delta\left(l, \boldsymbol{z}\right)$, which is discussed above. The next theorem combines these two types of errors to construct a confidence interval for this two-level approach.

The proof of this theorem is in Appendix C.

Theorem 4.3. *In the approach in which the conditional probabilities are approximated by CLT and the outer-level expectation is computed by MC simulation, given a confidence level α, there is a confidence interval $[u_\alpha(\boldsymbol{\mathcal{X}}), v_\alpha(\boldsymbol{\mathcal{X}})]$ defined by*

$$\begin{aligned}
u_\alpha(\boldsymbol{\mathcal{X}}) &= \overline{\boldsymbol{\mathcal{X}}^-} - t_{U-1,1-\alpha/2}\sqrt{S^2_{\boldsymbol{\mathcal{X}}^-}/U}, \\
v_\alpha(\boldsymbol{\mathcal{X}}) &= \overline{\boldsymbol{\mathcal{X}}^+} + t_{U-1,1-\alpha/2}\sqrt{S^2_{\boldsymbol{\mathcal{X}}^+}/U}, \\
\boldsymbol{\mathcal{X}}^\pm &= \left[X^\pm\left(\boldsymbol{\mathcal{Z}}^{(1)}\right), \ldots, X^\pm\left(\boldsymbol{\mathcal{Z}}^{(U)}\right)\right]^T, \\
X^\pm\left(\boldsymbol{\mathcal{Z}}^{(u)}\right) &= \Phi\left(\frac{l-\mu^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)}{\sigma^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)}\right) \pm B\left(\frac{l-\mu^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)}{\sigma^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)}\right), \\
\overline{\boldsymbol{\mathcal{X}}^\pm} &= \frac{1}{U}\sum_{u=1}^{U} X^\pm\left(\boldsymbol{\mathcal{Z}}^{(u)}\right), \\
S^2_{\boldsymbol{\mathcal{X}}^\pm} &= \frac{1}{U-1}\sum_{u=1}^{U}\left(X^\pm\left(\boldsymbol{\mathcal{Z}}^{(u)}\right) - \overline{\boldsymbol{\mathcal{X}}^\pm}\right)^2,
\end{aligned}$$

such that the true loss probability $P(l) = \mathbb{P}\left\{L^{(N)}\left(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{E}}\right) < l\right\}$ satisfies

$$\mathbb{P}\left\{P(l) \in [u_\alpha(\boldsymbol{\mathcal{X}}), v_\alpha(\boldsymbol{\mathcal{X}})]\right\} \geq 1-\alpha.$$

4.3 Numerical Results and Comparisons

In this section, we present numerical results for MC simulation, the LLN and CLT asymptotic approximations. All methods are implemented in MATLAB R2010b and the computation was performed on a workstation with a 2.8GHz Intel Core 2 Duo CPU and 6GB 667 MHz DDR2 SDRAM. The methods are applied to a variety of portfolios to assess accuracy and efficiency. Since we do not know the closed-form solution for the portfolio loss probabilities, we use the two-level MC approximation with sample size $U = 5000$ and $V = 5000$ as a benchmark. Also, we consider a single systematic factor only (i.e. $S = 1$) because multi-factor models take too long to compute the benchmark on a standard sequential machine. We consider 4 credit rating classes ($C = 4$): A, B, C and D, corresponding to $c = 3,\ 2,\ 1$, and 0, respectively. The credit rating "A" is the highest rating, while the credit rating "D" represents default. The unconditional credit migrating probabilities, $P^{c_2}_{c_1}$, are shown in Table 4.1.

Table 4.1: Credit migration matrix

$c_1 \backslash c_2$	**A (3)**	**B (2)**	**C (1)**	**D (0)**
A (3)	0.9796	0.0202	0.0000	0.0002
B (2)	0.0208	0.9398	0.0125	0.0270
C (1)	0.0000	0.0649	0.6801	0.2550
D (0)	0.0000	0.0000	0.0000	1.0000

4.3.1 Sample Portfolios

We consider two classes of portfolios: non-lumpy portfolios and lumpy portfolios. Obligors in a non-lumpy portfolio have about the same EAD. We further divide this class of portfolios into two subclasses: homogeneous portfolios and heterogeneous portfolios. All obligors in a homogeneous portfolio have an "A" credit rating, share the same parameters EAD_n, LGC_n^c, and β_n, while, in a heterogeneous portfolio, the parameters, EAD_n, LGC_n^c, and β_n, are randomly generated. For each subclass, we build three portfolios with different numbers of obligors: namely, fine-grained ($N = 5000$), medium-grained (N=500) and coarse-grained (N=50). Details of each portfolio are presented in Table 4.2.

Table 4.2: Parameters for non-lumpy portfolios

Portfolio		**Label**	$\mathbf{EAD}_n$	$\mathbf{LGC}_n^c$	$\boldsymbol{\beta}_n$	$\mathbf{c}(n)$	N	**HHI of ω's**
Homo-geneous	Coarse-grained	Π_1	1	$\widehat{\text{LGC}}_n^c$	0.5	3	50	2.00×10^{-2}
	Medium-grained	Π_2					500	2.00×10^{-3}
	Fine-grained	Π_3					5000	2.00×10^{-4}
Hetero-geneous	Coarse-grained	Π_4	$\text{Unif}(0.5, 1.5)$	$\widetilde{\text{LGC}}_n^c$	$\text{Unif}(-0.9, 0.9)$	$\text{Unif}\{1, 2, 3\}$	50	2.20×10^{-2}
	Medium-grained	Π_5					500	2.18×10^{-3}
	Fine-grained	Π_6					5000	2.17×10^{-4}

The loss-given-credit-event, LGC_n^c, given in Table 4.2 are set to be

$$\overline{\mathrm{LGC}}_n^c = \begin{cases} 0.8, & c=0 \\ 0.5, & c=1 \\ 0.3, & c=2 \\ 0.0, & c=3 \end{cases}, \qquad \widetilde{\mathrm{LGC}}_n^c = \begin{cases} \mathrm{LGD}_n, & \text{if} \quad c=0 \\ \mathrm{Unif}\left(0, \widetilde{\mathrm{LGC}}_n^{c-1}\right), & \text{if} \quad 0<c<c(n) \\ 0, & \text{if} \quad 0<c=c(n) \\ \mathrm{Unif}\left(R_n, \widetilde{\mathrm{LGC}}_n^{c-1}\right), & \text{if} \quad c>c(n) \end{cases}, \tag{4.3.1}$$

for homogeneous portfolios and for heterogeneous portfolios respectively. In the latter case, obligor n's loss-given-default, LGD_n, is randomly generated with distribution $\mathrm{Unif}(0, 0.8)$, and R_n is a negative number representing the return of obligor n, which is randomly generated with distribution $\mathrm{Unif}(-0.5, 0)$. The choice of LGC_n^c in (4.3.1) for heterogeneous portfolios ensures that the following conditions hold:

- If an obligor defaults, the LGC of this obligor equals its LGD;
- If an obligor is downgraded, the LGC of this obligor is positive, but no greater than its LGD;
- If an obligor's credit state remains unchanged, the LGC of this obligor is zero;
- If an obligor is upgraded, the LGC of this obligor will be negative, but no less than its market return;
- Obligor n's LGC is decreasing with respect to c.

The last column in Table 4.2 shows the HHI index of each portfolio, where the HHI index equals $\sum_{n=1}^N \omega_n^2$. The HHI index is no less than $1/N$, and the closer to zero it is, the less concentrated the portfolio is. Meanwhile, for the purpose of comparison, we let positions in Π_4 be contained in Π_5, and positions in Π_5 be contained in Π_6, and set $\sum_{n=1}^N \mathrm{EAD}_n$ in Π_4, Π_5, Π_6 to be 50, 500, 5000, respectively.

4.3.2 The CLT Approximation and Confidence Intervals

First we check the CLT approximation and its confidence intervals constructed based on the discussion in Subsection 4.2.2. For confidence intervals for the CLT approximation, we choose the bound on the asymptotic errors to be

$$B(x) = \min\left(C_0\rho^{(N)}, C_1\frac{\rho^{(N)}}{1+|x|^3}\right), \tag{4.3.2}$$

since we know $\mathbb{E}\left[\left|\frac{\widetilde{L}_n(\boldsymbol{z},\boldsymbol{\mathcal{E}})-\widetilde{\mu}_n(\boldsymbol{z})}{\sigma^{(N)}(\boldsymbol{z})}\right|^3\right]$ is finite.

We test the CLT approximation on the non-lumpy portfolios Π_1 to Π_6. For each portfolio, we choose an interval $[l_{min}, l_{max}]$ and then compute 200 equally spaced quantiles satisfying $l_k = l_{min} + \frac{k-1}{199}(l_{max} - l_{min})$, $k = 1, \ldots, 200$. To check the performance of the approximation at both tails of the distribution, l_{min} and l_{max} are chosen to satisfy

$$\mathbb{P}\{\mathcal{L} \leq l_{min}\} < 10^{-4}, \quad \mathbb{P}\{\mathcal{L} \leq l_2\} \geq 10^{-4},$$

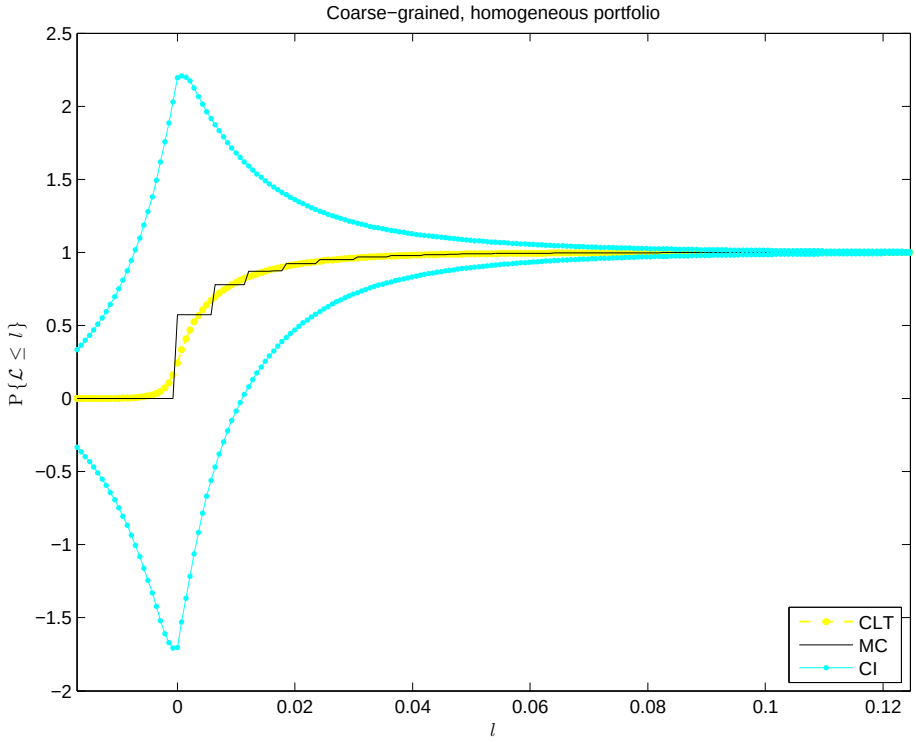
and

$$\mathbb{P}\{\mathcal{L} > l_{max}\} < 10^{-4}, \quad \mathbb{P}\{\mathcal{L} > l_{199}\} \geq 10^{-4}.$$

Since risk managers are more interested in the tail risk, for example, 95% value-at-risk or expected shortfall, we focus on the high probability tail by examining $\mathbb{P}\{\mathcal{L} > l\}$. Since the tail is very close to zero and very flat, we plot $\mathbb{P}\{\mathcal{L} > l\}$ on a semi-log scale. The graphs also show the 95% confidence intervals associated with the CLT approximation. Figures 4.1 - 4.6 present the results for portfolio Π_1 - Π_6, respectively.

Figure 4.1: Loss distribution generated by the CLT approximation: Portfolios Π_1
yellow: CLT, black: MC, cyan: CI

(a) $\mathbb{P}\{\mathcal{L} \leq l\}$: Portfolio Π_1



(b) $\mathbb{P}\{\mathcal{L} > l\}$: Portfolio Π_1

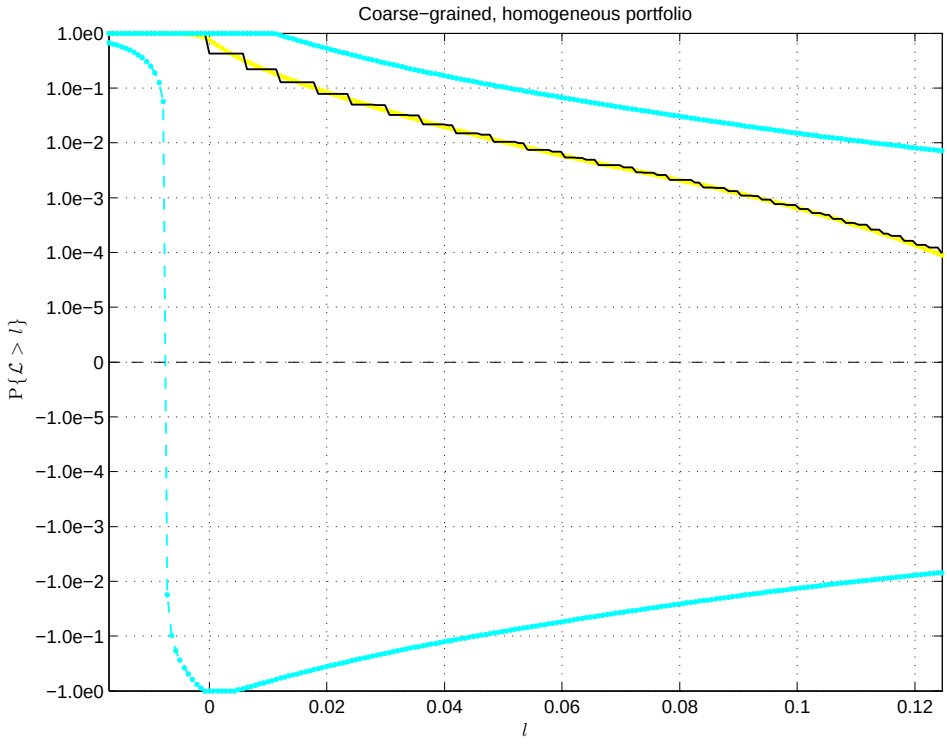
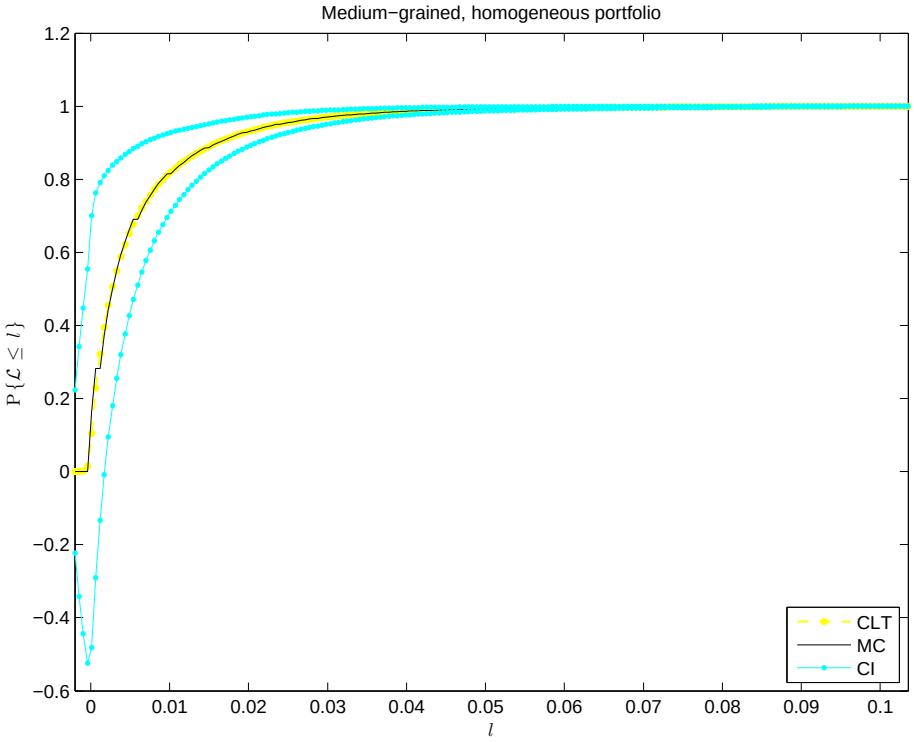


Figure 4.2: Loss distribution generated by the CLT approximation: Portfolios Π_2
yellow: CLT, black: MC, cyan: CI

(a) $\mathbb{P}\{\mathcal{L} \leq l\}$: Portfolio Π_2



(b) $\mathbb{P}\{\mathcal{L} > l\}$: Portfolio Π_2

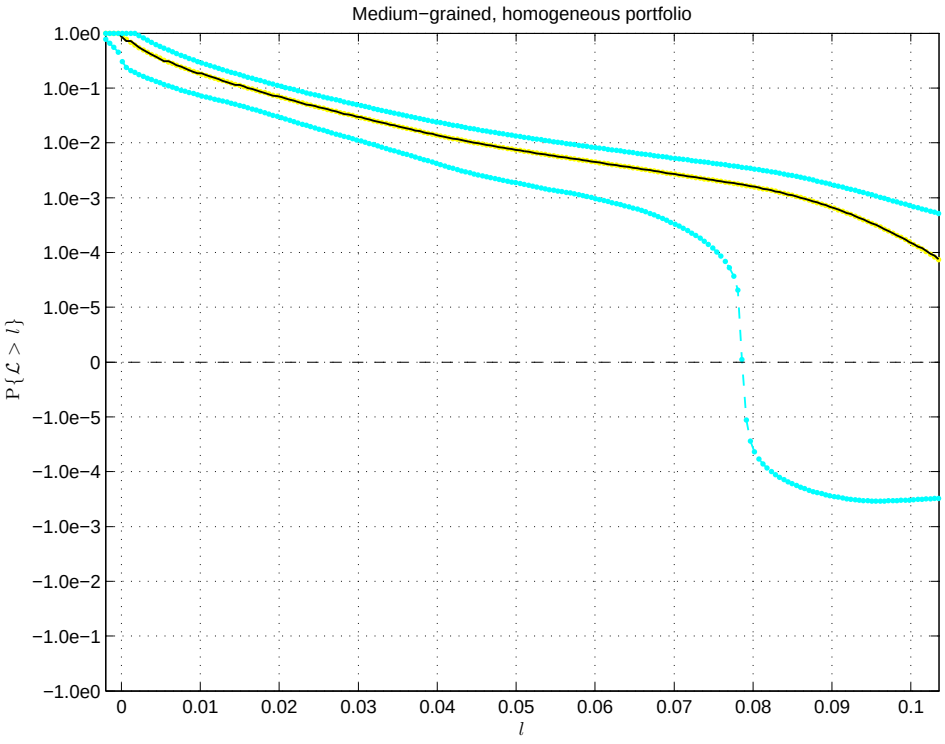
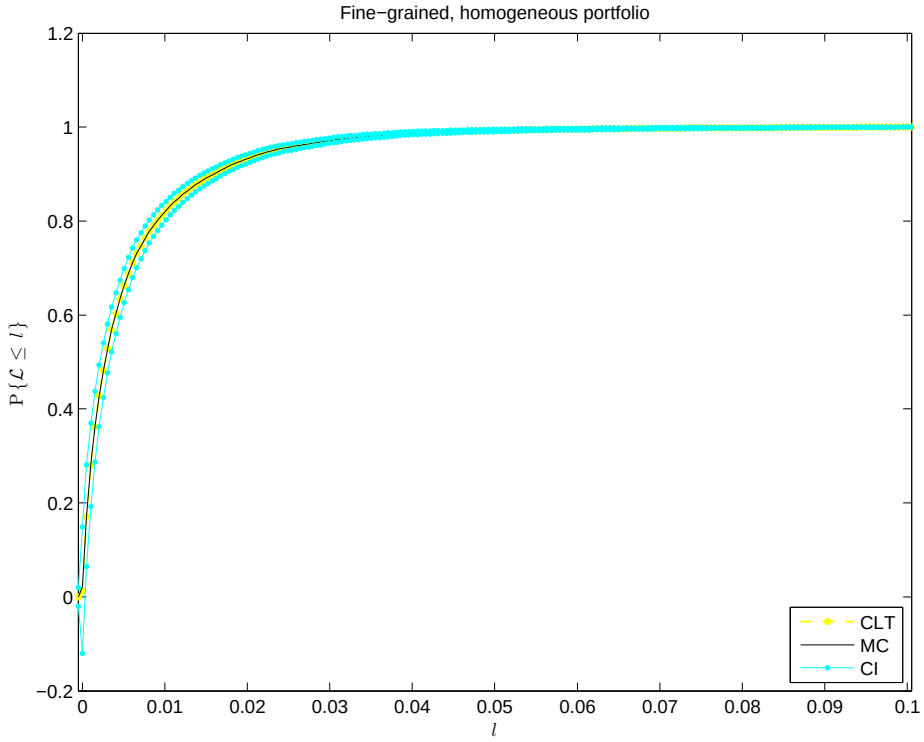


Figure 4.3: Loss distribution generated by the CLT approximation: Portfolios Π_3
yellow: CLT, black: MC, cyan: CI

(a) $\mathbb{P}\{\mathcal{L} \leq l\}$: Portfolio Π_3



(b) $\mathbb{P}\{\mathcal{L} > l\}$: Portfolio Π_3

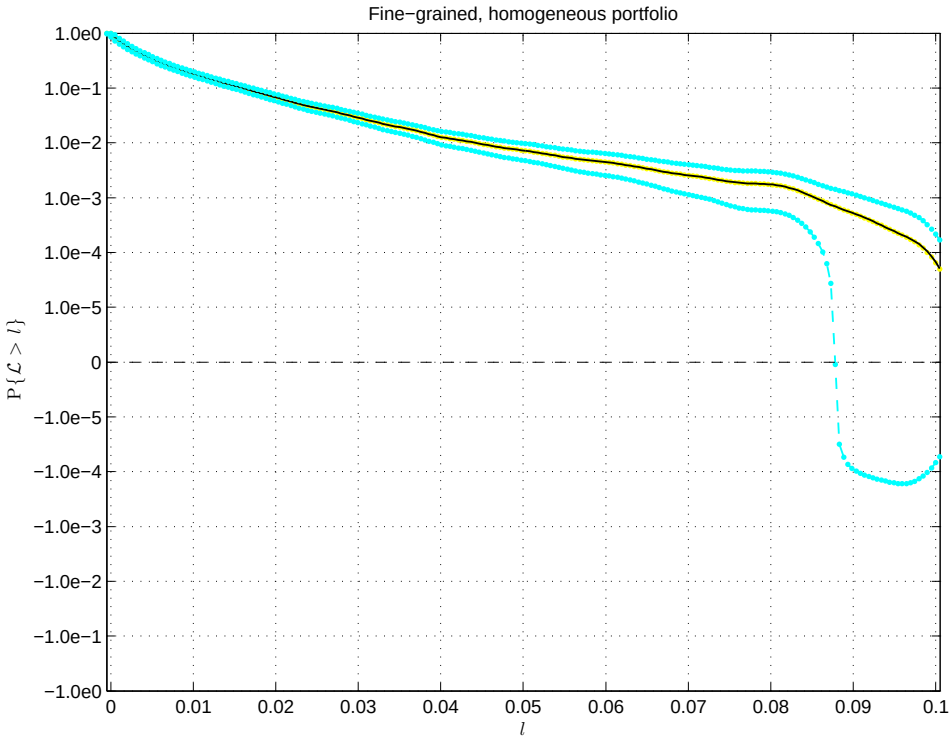
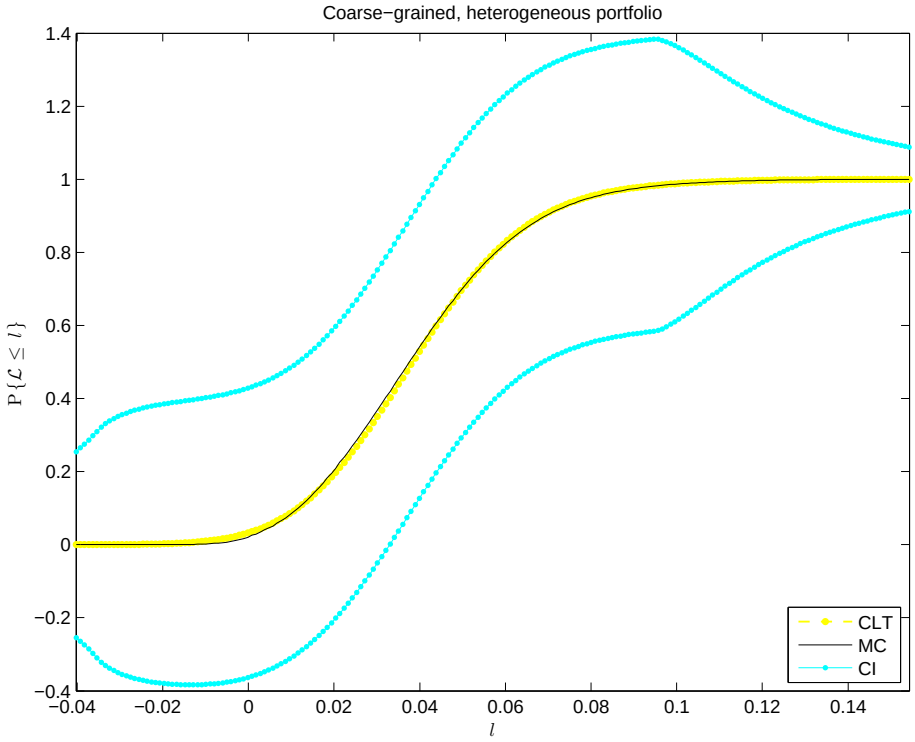


Figure 4.4: Loss distribution generated by the CLT approximation: Portfolios Π_4
yellow: CLT, black: MC, cyan: CI

(a) $\mathbb{P}\{\mathcal{L} \leq l\}$: Portfolio Π_4



(b) $\mathbb{P}\{\mathcal{L} > l\}$: Portfolio Π_4

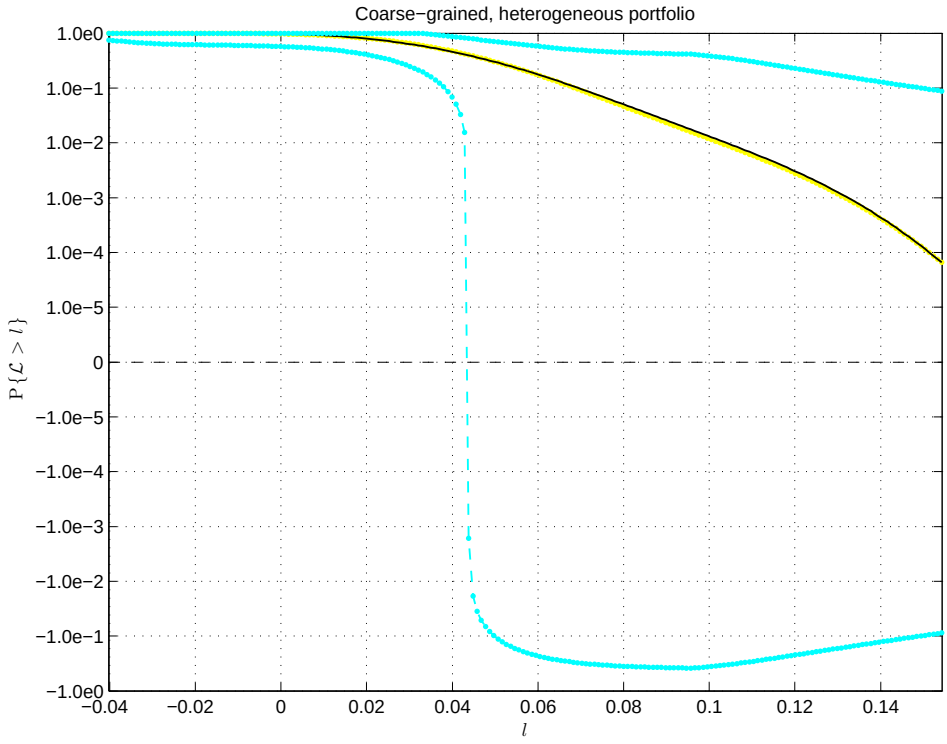
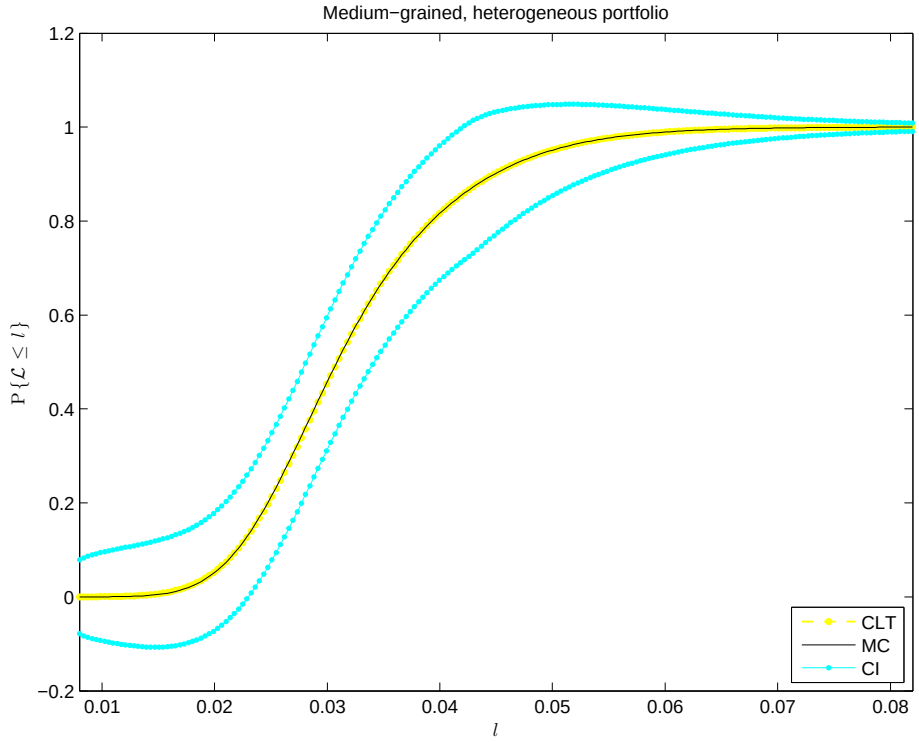


Figure 4.5: Loss distribution generated by the CLT approximation: Portfolios Π_5
yellow: CLT, black: MC, cyan: CI

(a) $\mathbb{P}\{\mathcal{L} \leq l\}$: Portfolio Π_5



(b) $\mathbb{P}\{\mathcal{L} > l\}$: Portfolio Π_5

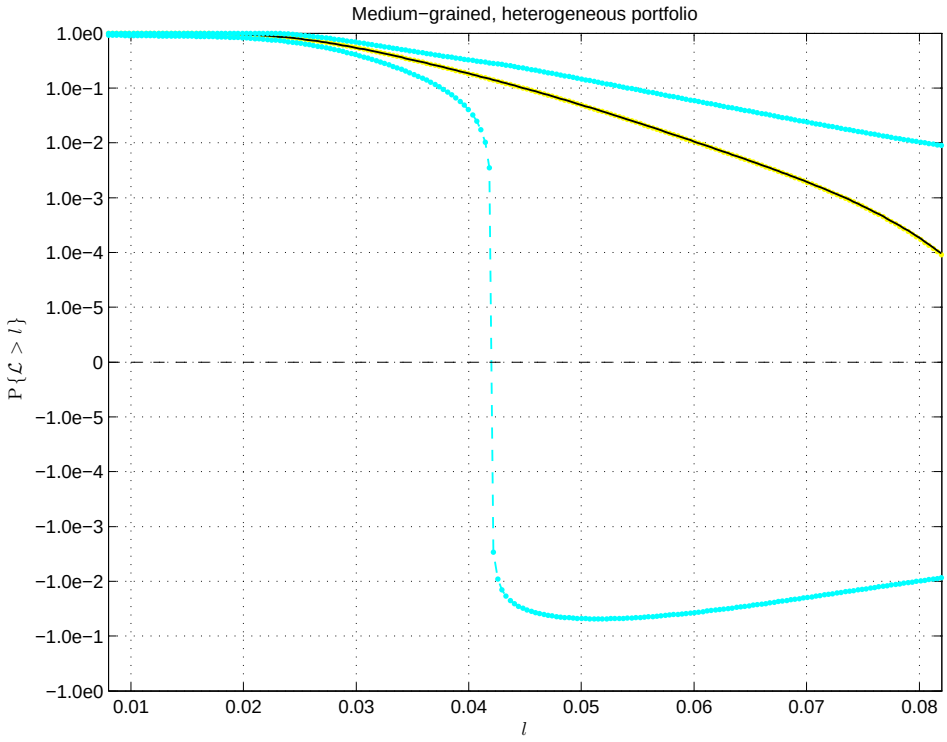
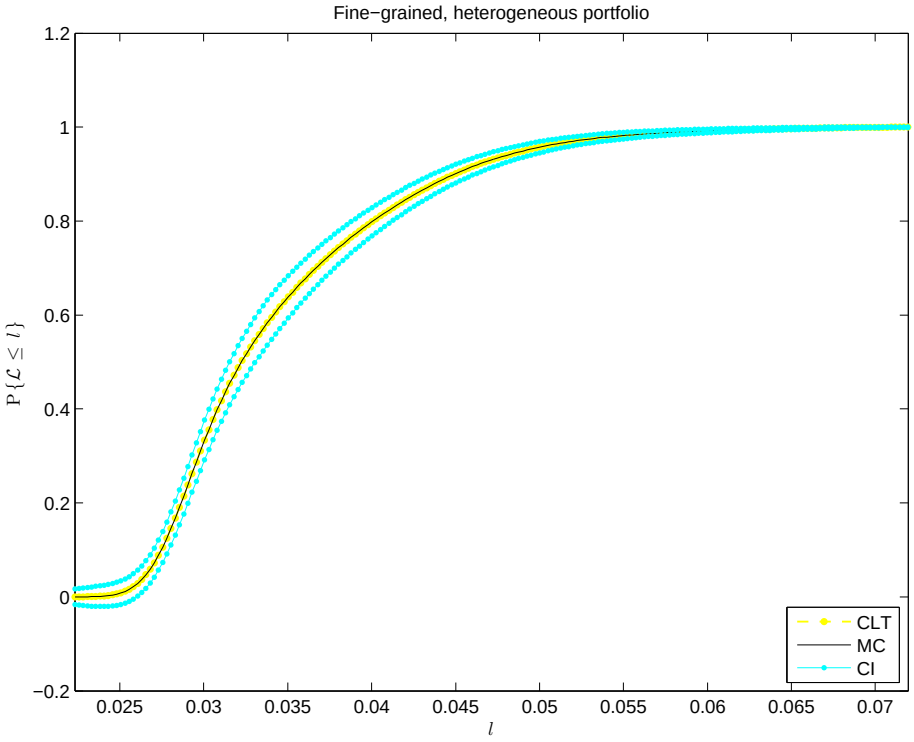
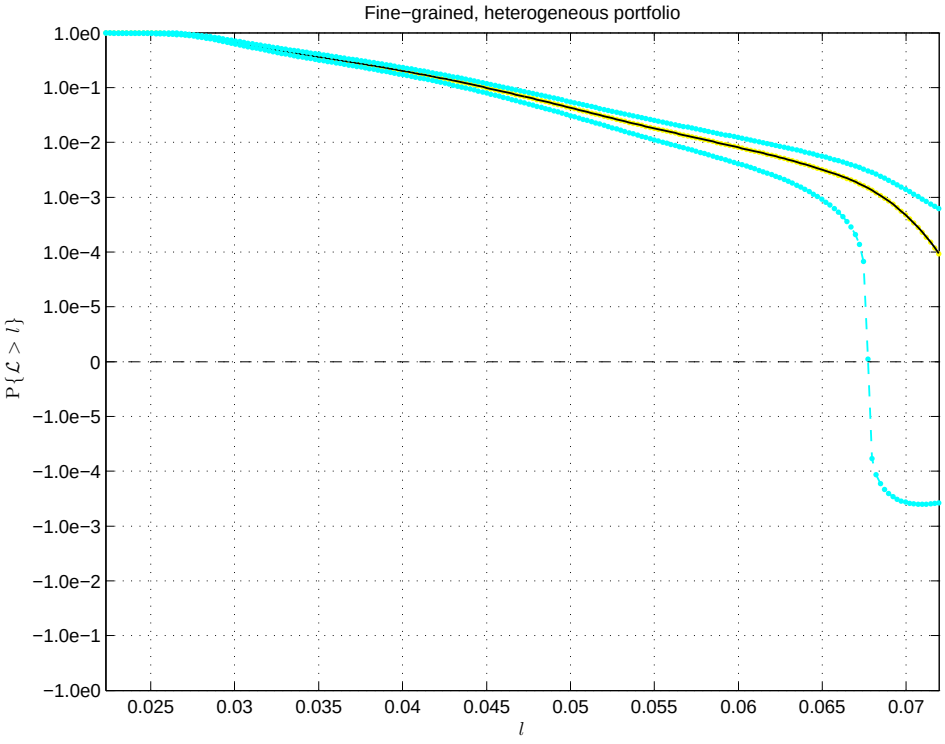


Figure 4.6: Loss distribution generated by the CLT approximation: Portfolios Π_6
yellow: CLT, black: MC, cyan: CI

(a) $\mathbb{P}\{\mathcal{L} \leq l\}$: Portfolio Π_6



(b) $\mathbb{P}\{\mathcal{L} > l\}$: Portfolio Π_6



From Figures 4.1 - 4.6, we make the following observations.

1. The loss probabilities generated by the CLT approximation agree well with those computed by the two-level MC benchmark, especially in the tail. The CLT approximation is more accurate for more fine-grained portfolios, since unsystematic risk is diversified away in large portfolios, and it has more impact for small portfolios.

2. By comparing Figure 4.1a and Figure 4.4a we find that, for the same portfolio size, the CLT approximation is more accurate for the heterogeneous portfolio than for the homogeneous one. Since the true portfolio losses are discrete rather than continuous, for small homogeneous portfolios, the two-level MC benchmark distribution has obvious jumps. However, the CLT approximation is continuous. Thus part of the error in Figure 4.1a and Figure 4.4a comes from approximating a discrete distribution by a continuous one. The jumps in the two-level MC benchmark are much smaller for the corresponding heterogeneous portfolio. Moreover, as the portfolio size increases, the two-level MC benchmark becomes smoother, and the accuracy of the CLT approximation improves.

3. Confidence intervals shown in Figures 4.1 - 4.6 become tighter as the portfolio size increases, and, for the same portfolio, confidence intervals are narrower in the tails and wider in the middle. By comparing with probabilities calculated by a Quadrature+CLT approximation, where the outer-level integration is calculated by a quadrature and conditional probabilities are estimated by the CLT approximation, the length of the confidence intervals depends mainly on the bound on the asymptotic errors (4.3.2). More specifically, it is proportional to

$$B\left(\frac{l-\mu^{(N)}(\boldsymbol{z})}{\sigma^{(N)}(\boldsymbol{z})}\right) \quad = \quad \min\left(C_0\rho^{(N)}(\boldsymbol{z}), \frac{C_1\rho^{(N)}(\boldsymbol{z})}{1+\left|\frac{l-\mu^{(N)}(\boldsymbol{z})}{\sigma^{(N)}(\boldsymbol{z})}\right|^3}\right), \tag{4.3.3}$$

where

$$\rho^{(N)}(\boldsymbol{z}) = \sum_{n=1}^{N} \mathbb{E}\left[\left|\frac{\widetilde{L}_n(\boldsymbol{z},\boldsymbol{\mathcal{E}}) - \widetilde{\mu}_n(\boldsymbol{z})}{\sigma^{(N)}(\boldsymbol{z})}\right|^3\right].$$

If we keep the portfolio and $\boldsymbol{z}$ unchanged, then $\rho^{(N)}(\boldsymbol{z})$, $\mu^{(N)}(\boldsymbol{z})$ and $\sigma^{(N)}(\boldsymbol{z})$ are fixed. As l moves towards either tail, it moves away from $\mu^{(N)}(\boldsymbol{z})$, whence $\left|\frac{l-\mu^{(N)}(\boldsymbol{z})}{\sigma^{(N)}(\boldsymbol{z})}\right|^3$ becomes larger. Hence the bound in (4.3.3) gets smaller, which explains why confidence intervals are narrower in the tails and wider in the middle. If we fix l and $\boldsymbol{z}$, and increase the size of a homogeneous portfolio, then

$\rho^{(N)}(\boldsymbol{z})$ decreases, since

$$\begin{aligned}\rho^{(N)}(\boldsymbol{z}) &= \frac{\sum_{n=1}^{N}|\omega_n|^3\,\mathbb{E}\left[\left|L_n(\boldsymbol{z},\boldsymbol{\mathcal{E}})-\mu_n(\boldsymbol{z})\right|^3\right]}{\sqrt{\sum_{n=1}^{N}\omega_n^2\sigma_n^2(\boldsymbol{z})}}\\ &\leq \frac{\sum_{n=1}^{N}|\omega_n|^3}{\sqrt{\sum_{n=1}^{N}\omega_n^2}}A(\boldsymbol{z})\\ &= \frac{\sum_{n=1}^{N}\frac{1}{N^3}}{\sqrt{\sum_{n=1}^{N}\frac{1}{N^2}}}A(\boldsymbol{z})\\ &= N^{-3/2}A(\boldsymbol{z})\end{aligned}$$

decreases with N, where $A(\boldsymbol{z})=\mathbb{E}\left[\left|L_n(\boldsymbol{z},\boldsymbol{\mathcal{E}})-\mu_n(\boldsymbol{z})\right|^3\right]/\sigma_n(\boldsymbol{z})$,[1] and

$$\begin{aligned}\left|\frac{l-\mu^{(N)}(\boldsymbol{z})}{\sigma^{(N)}(\boldsymbol{z})}\right|^3 &= \left|\frac{l-\sum_{n=1}^{N}\omega_n\mu_n(\boldsymbol{z})}{\sqrt{\sum_{n=1}^{N}\omega_n^2\sigma_n^2(\boldsymbol{z})}}\right|^3\\ &= \left|\frac{l-\sum_{n=1}^{N}\frac{1}{N}\mu(\boldsymbol{z})}{\sqrt{\sum_{n=1}^{N}\frac{1}{N^2}}\sigma(\boldsymbol{z})}\right|^3\\ &= N^{3/2}\left|\frac{l-\mu(\boldsymbol{z})}{\sigma(\boldsymbol{z})}\right|^3\end{aligned}$$

increases with N. This partly explains why confidence intervals are narrower for larger portfolios.

4. Confidence intervals are too wide for practical use for small portfolios, but they are acceptable for large portfolios with large quantiles, because the Berry-Esseen-type bound in (4.3.2) is not tight enough for a small number of random variables, but becomes tighter as the number of random variables increases. Although one can narrow the confidence intervals by further bounding them by $[0,1]$, since the estimator is a probability which is within $[0,1]$, this will not improve the confidence intervals very much.

4.3.3 Comparison: MC, CLT and LLN

In this subsection we compare the accuracy and efficiency of the MC, CLT and LLN approximations for the non-lumpy portfolios Π_1-Π_6. Table 4.3 shows the CPU time in seconds used to generate the loss distribution and the speedup (numbers in parentheses) relative to the MC approximation (CPU time over MC CPU time). From Table 4.3, it is clear that the MC approximation is extremely time-consuming

[1] Notice that, for a homogenous portfolio, $A(\boldsymbol{z})=\mathbb{E}\left[\left|L_n(\boldsymbol{z},\boldsymbol{\mathcal{E}})-\mu_n(\boldsymbol{z})\right|^3\right]/\sigma_n(\boldsymbol{z})$ is identical and not dependent on N for each obligor n.

compared to the LLN and CLT approximations, and that the CLT approximation uses only slightly more CPU time than the LLN approximation.

Table 4.3: Comparison of total CPU time for non-lumpy portfolios (in seconds)

	Homogeneous					
	Coarse-grained		**Medium-grained**		**Fine-grained**	
MC	1.090×10^3	(–)	6.980×10^3	(–)	6.368×10^4	(–)
LLN	4.263×10^0	(2.556×10^2)	6.463×10^0	(1.080×10^3)	2.633×10^1	(2.419×10^3)
CLT	4.645×10^0	(2.347×10^2)	7.028×10^0	(9.932×10^2)	2.646×10^1	(2.406×10^3)

	Heterogeneous					
	Coarse-grained		**Medium-grained**		**Fine-grained**	
MC	1.121×10^3	(–)	7.022×10^3	(–)	6.438×10^4	(–)
LLN	4.220×10^0	(2.656×10^2)	6.439×10^0	(1.091×10^3)	2.581×10^1	(2.494×10^3)
CLT	4.606×10^0	(2.434×10^2)	7.002×10^0	(1.003×10^3)	2.637×10^1	(2.441×10^3)

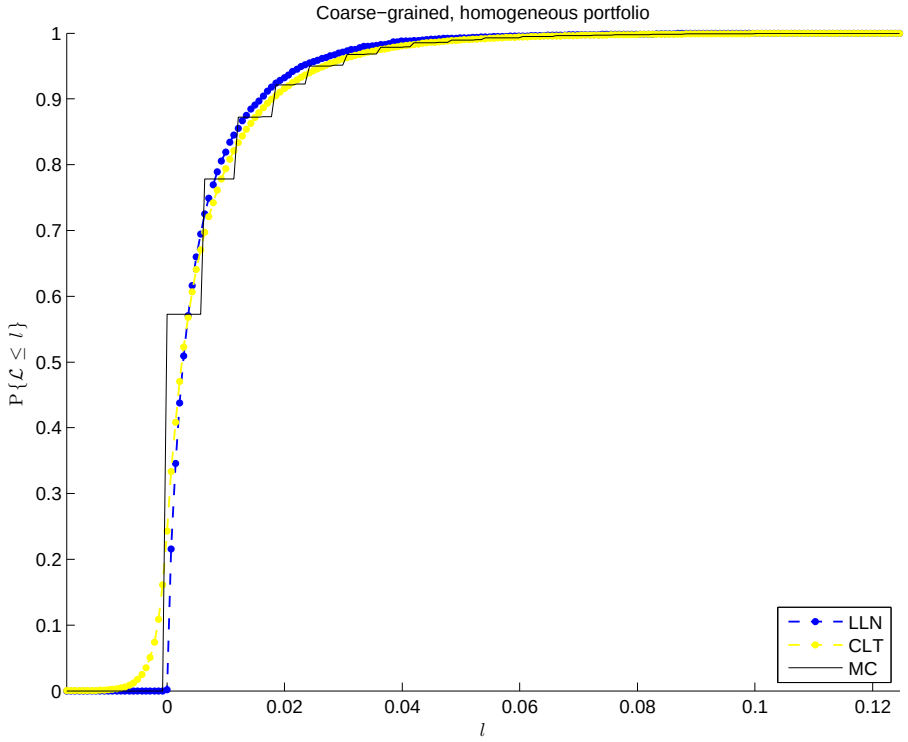
To compare the accuracy of the methods, we first plot the loss distributions generated by the three different approximations, using plots similar to those used in the preceding subsection. The results for portfolios Π_1 - Π_6 are presented in Figures 4.7 - 4.12, respectively. Using probabilities generated by the MC approximation as a benchmark, we compute the errors associated with the probabilities generated by the CLT and LLN approximations:

$$\begin{aligned} e_{lln}(l) &= \mathbb{P}_{LLN}\{\mathcal{L} \leq l\} - \mathbb{P}_{MC}\{\mathcal{L} \leq l\}, \\ e_{clt}(l) &= \mathbb{P}_{CLT}\{\mathcal{L} \leq l\} - \mathbb{P}_{MC}\{\mathcal{L} \leq l\}. \end{aligned}$$

We also compute the difference of the absolute errors associated with the CLT and LLN approximations: $\Delta e(l) = |e_{clt}(l)| - |e_{lln}(l)|$. These errors are plotted in Figures 4.13 - 4.18.

Figure 4.7: Comparison of the loss distribution for the MC, LLN and CLT approximations: Portfolios Π_1
blue: LLN, yellow: CLT, black: MC

(a) $\mathbb{P}\{\mathcal{L} \leq l\}$: Portfolio Π_1



(b) $\mathbb{P}\{\mathcal{L} > l\}$: Portfolio Π_1

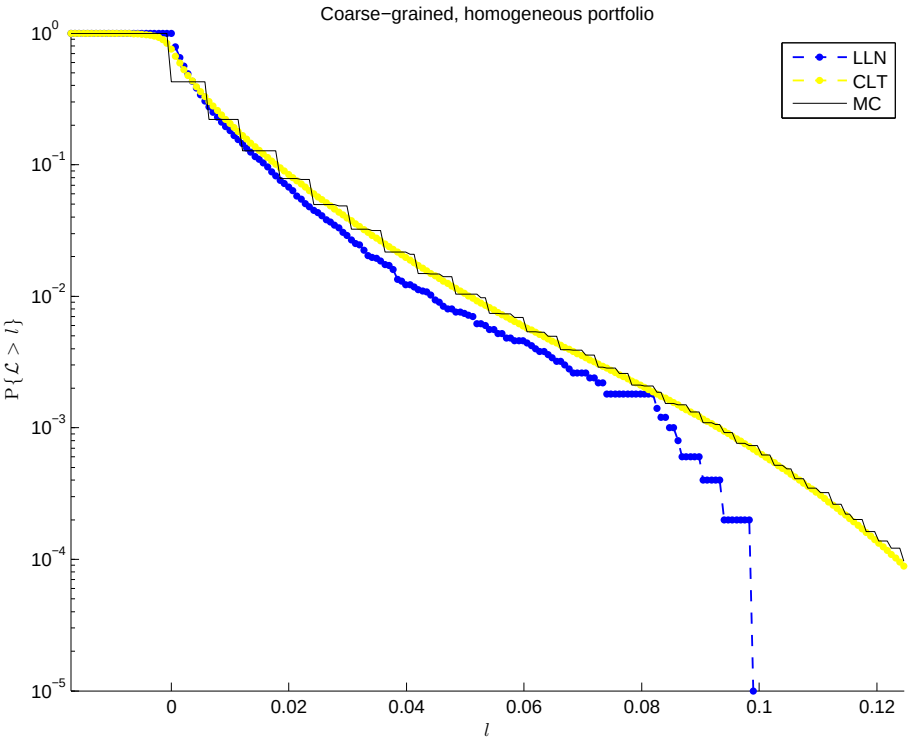
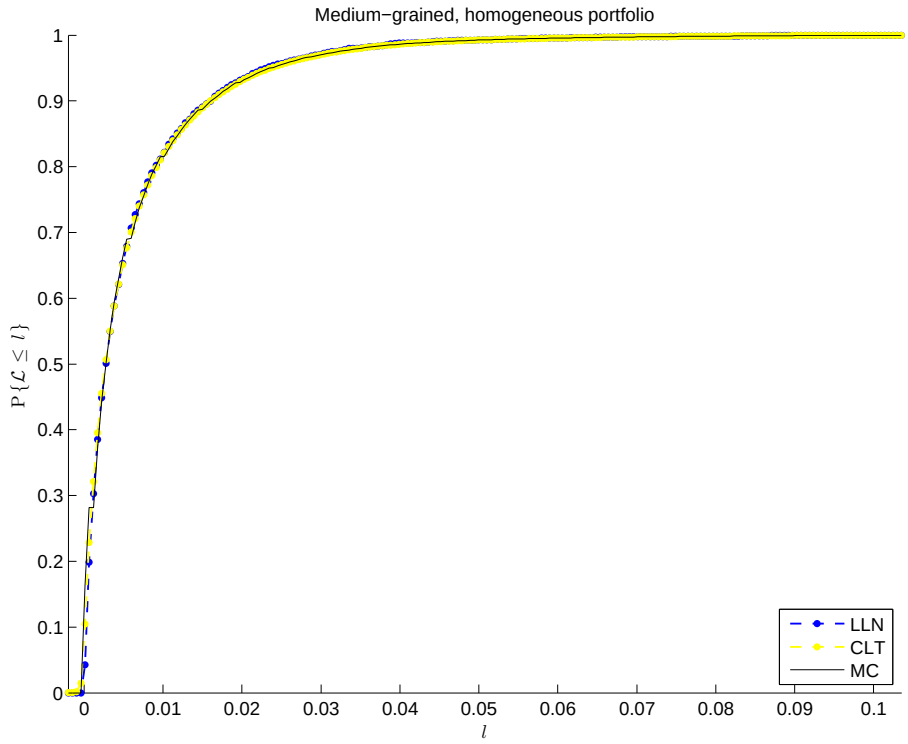


Figure 4.8: Comparison of the loss distribution for the MC, LLN and CLT approximations: Portfolios Π_2
blue: LLN, yellow: CLT, black: MC

(a) $\mathbb{P}\{\mathcal{L} \leq l\}$: Portfolio Π_2



(b) $\mathbb{P}\{\mathcal{L} > l\}$: Portfolio Π_2

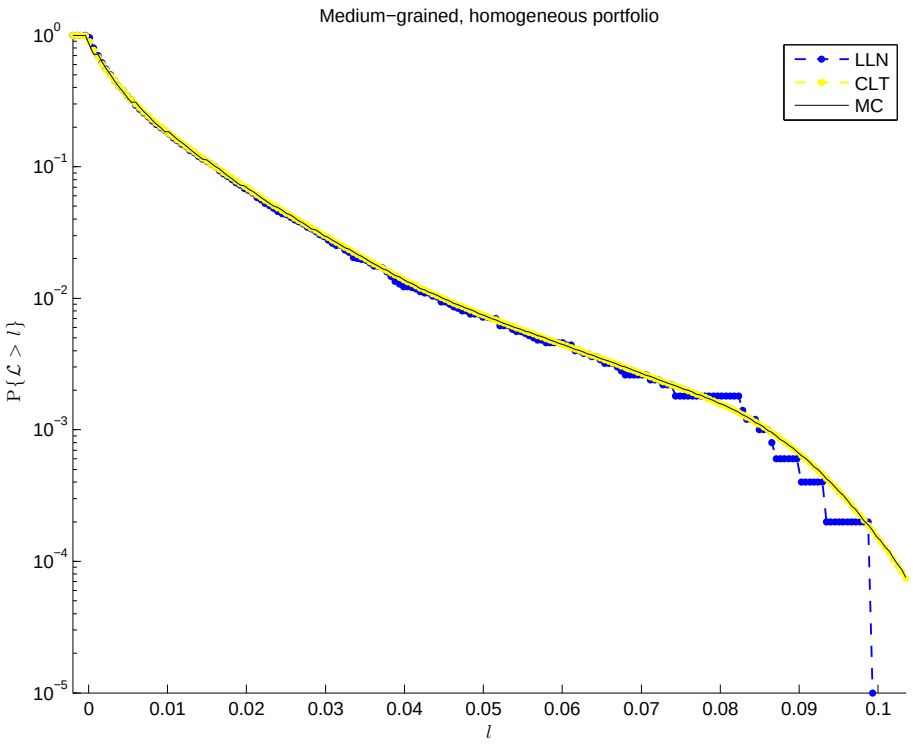
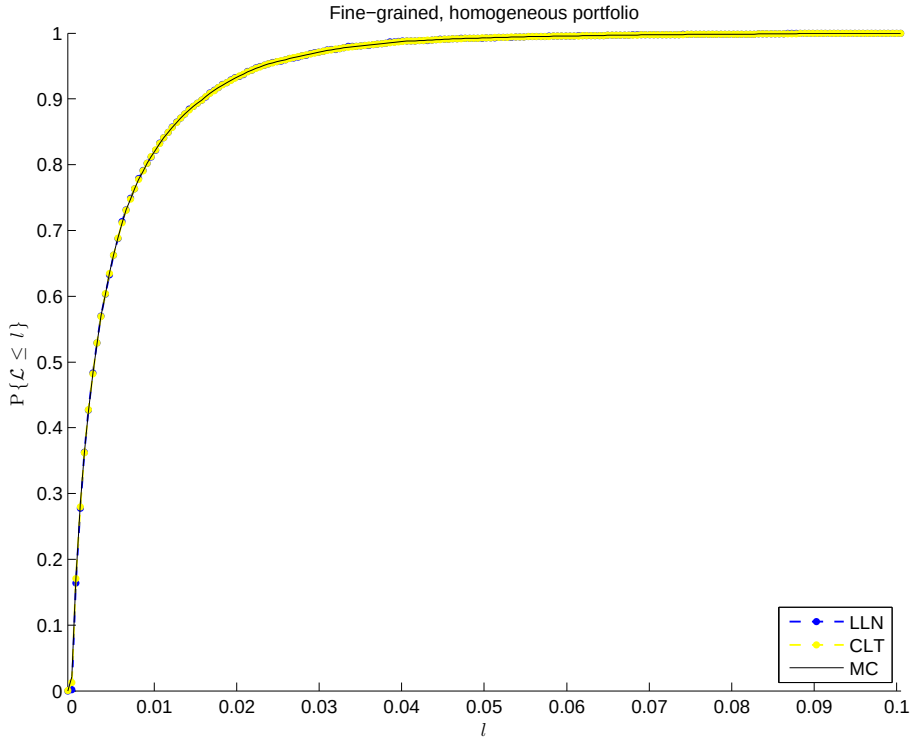


Figure 4.9: Comparison of the loss distribution for the MC, LLN and CLT approximations: Portfolios Π_3
blue: LLN, yellow: CLT, black: MC

(a) $\mathbb{P}\{\mathcal{L} \leq l\}$: Portfolio Π_3



(b) $\mathbb{P}\{\mathcal{L} > l\}$: Portfolio Π_3

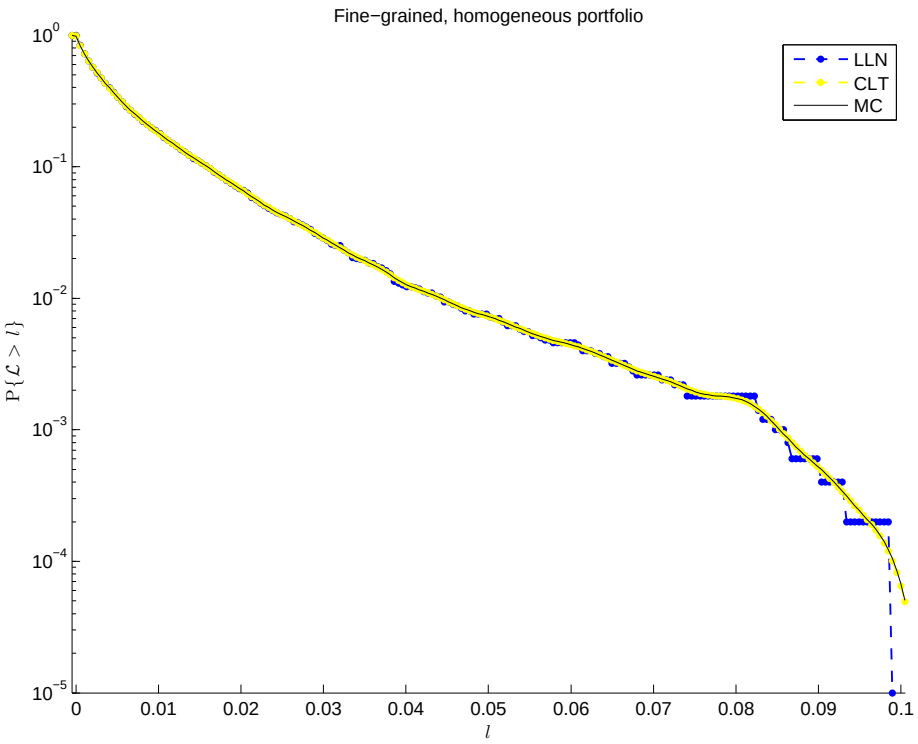
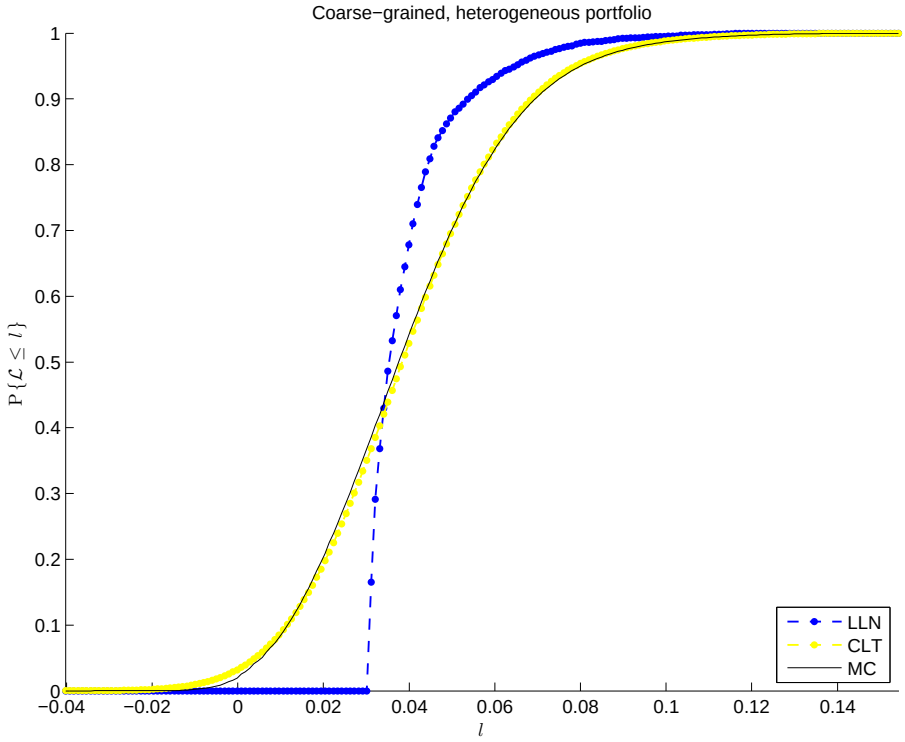


Figure 4.10: Comparison of the loss distribution for the MC, LLN and CLT approximations: Portfolios Π_4
blue: LLN, yellow: CLT, black: MC

(a) $\mathbb{P}\{\mathcal{L} \leq l\}$: Portfolio Π_4



(b) $\mathbb{P}\{\mathcal{L} > l\}$: Portfolio Π_4

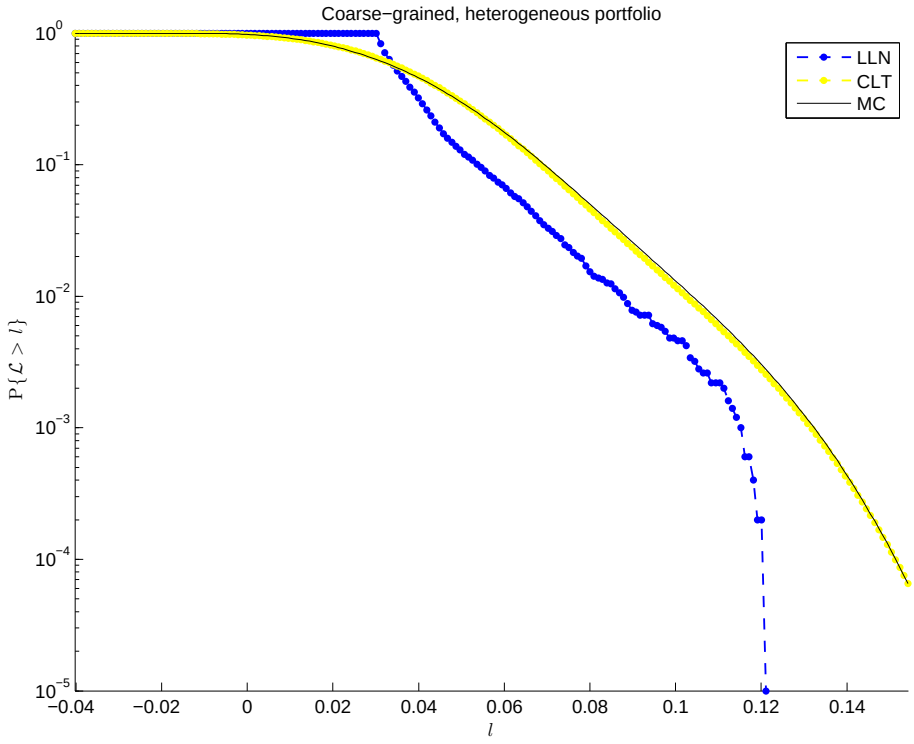
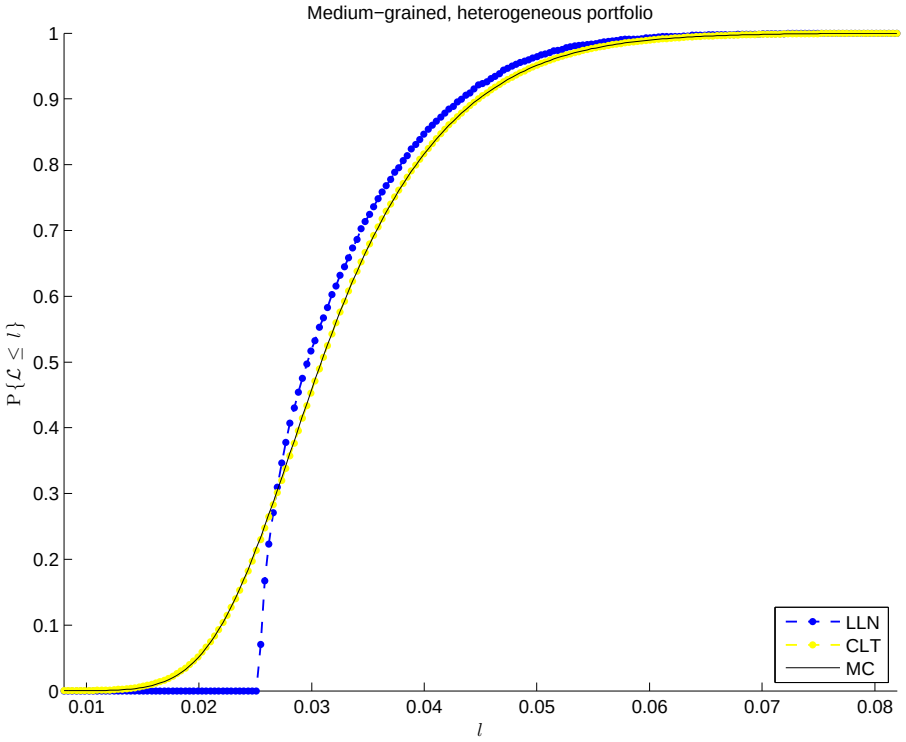


Figure 4.11: Comparison of the loss distribution for the MC, LLN and CLT approximations: Portfolios Π_5
blue: LLN, yellow: CLT, black: MC

(a) $\mathbb{P}\{\mathcal{L} \leq l\}$: Portfolio Π_5



(b) $\mathbb{P}\{\mathcal{L} > l\}$: Portfolio Π_5

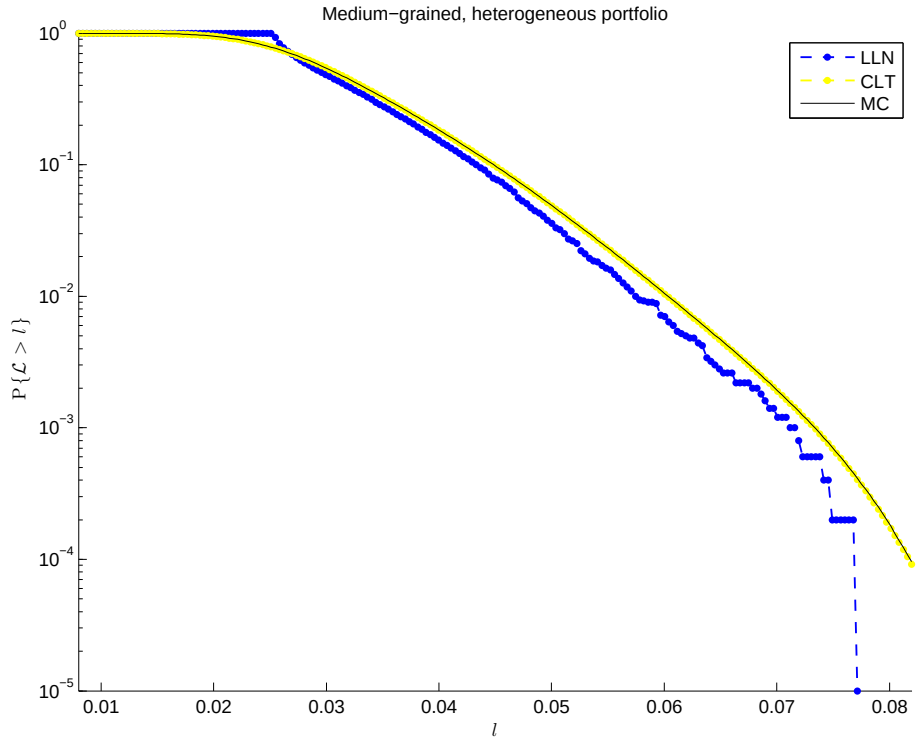
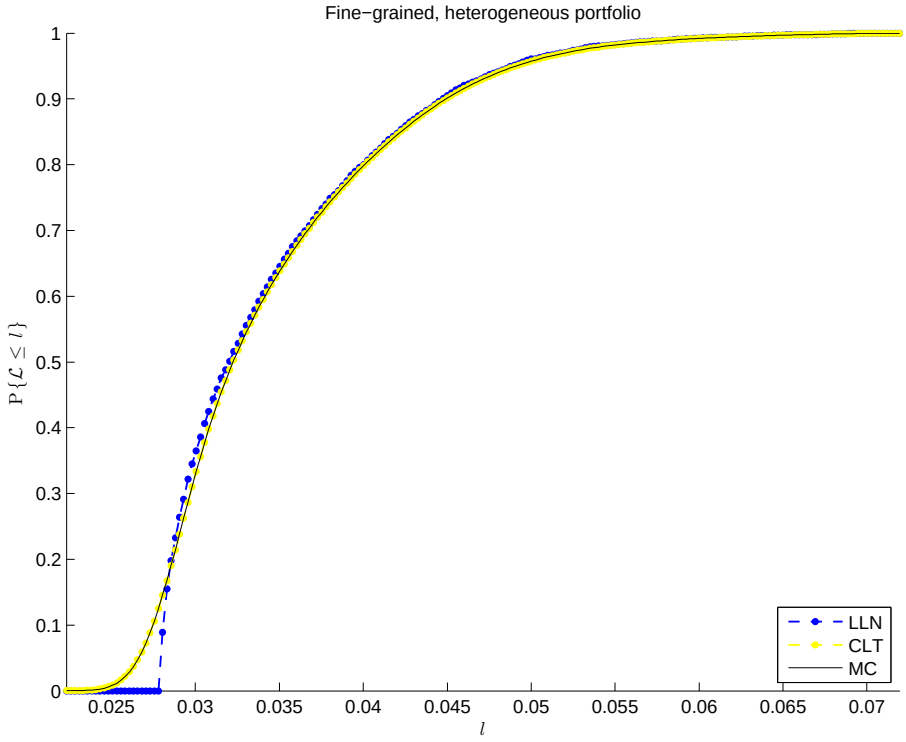


Figure 4.12: Comparison of the loss distribution for the MC, LLN and CLT approximations: Portfolios Π_6
blue: LLN, yellow: CLT, black: MC

(a) $\mathbb{P}\{\mathcal{L} \leq l\}$: Portfolio Π_6



(b) $\mathbb{P}\{\mathcal{L} > l\}$: Portfolio Π_6

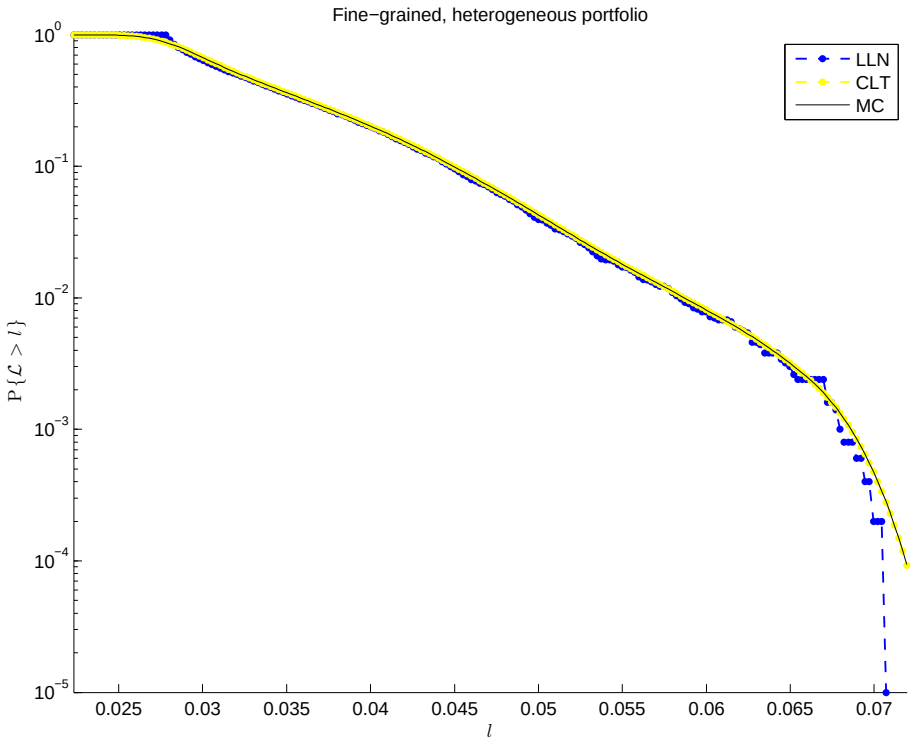
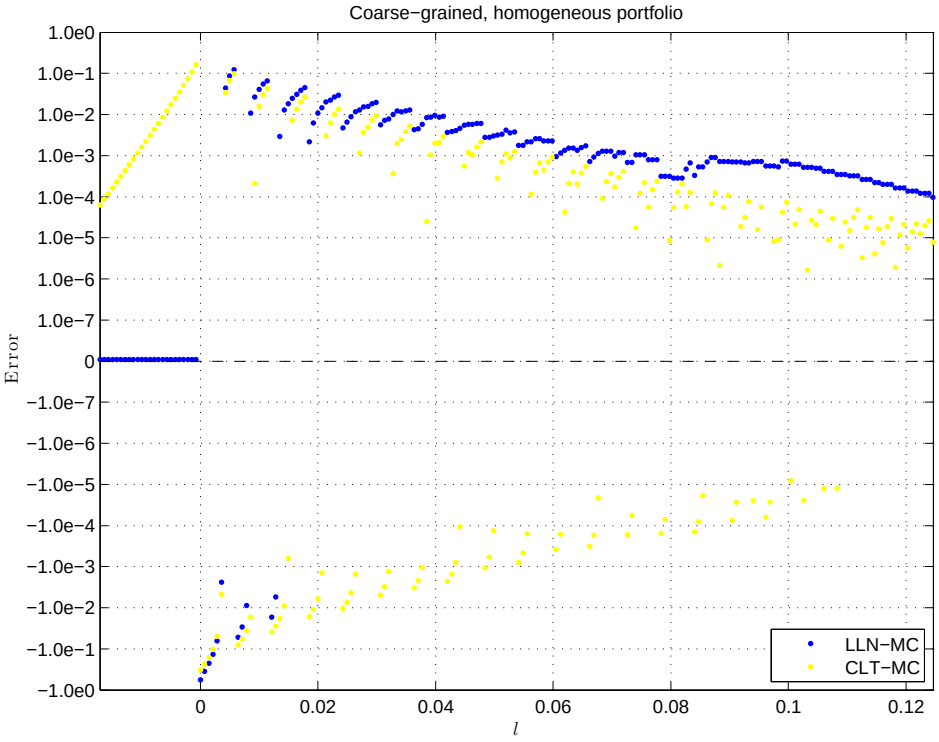


Figure 4.13: Comparison of the errors for the LLN and CLT approximations: Portfolios Π_1

(a) Errors: Portfolio Π_1, *blue:* $e_{lln}(l)$, *yellow:* $e_{clt}(l)$



(b) Difference in errors: Portfolio Π_1, *cyan:* $|e_{clt}(l)| - |e_{lln}(l)|$

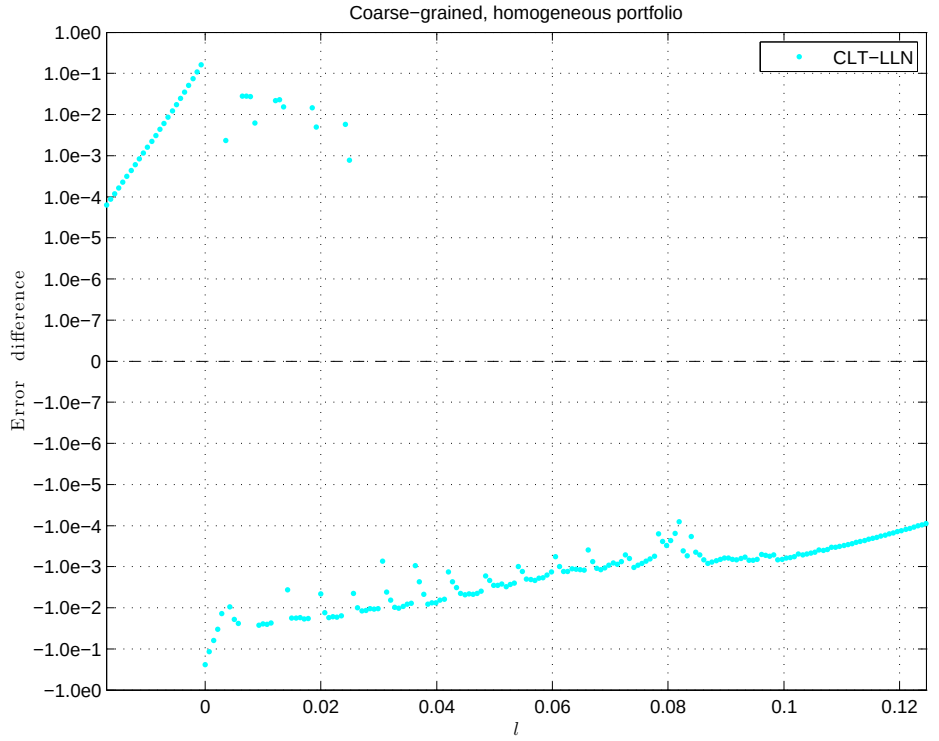
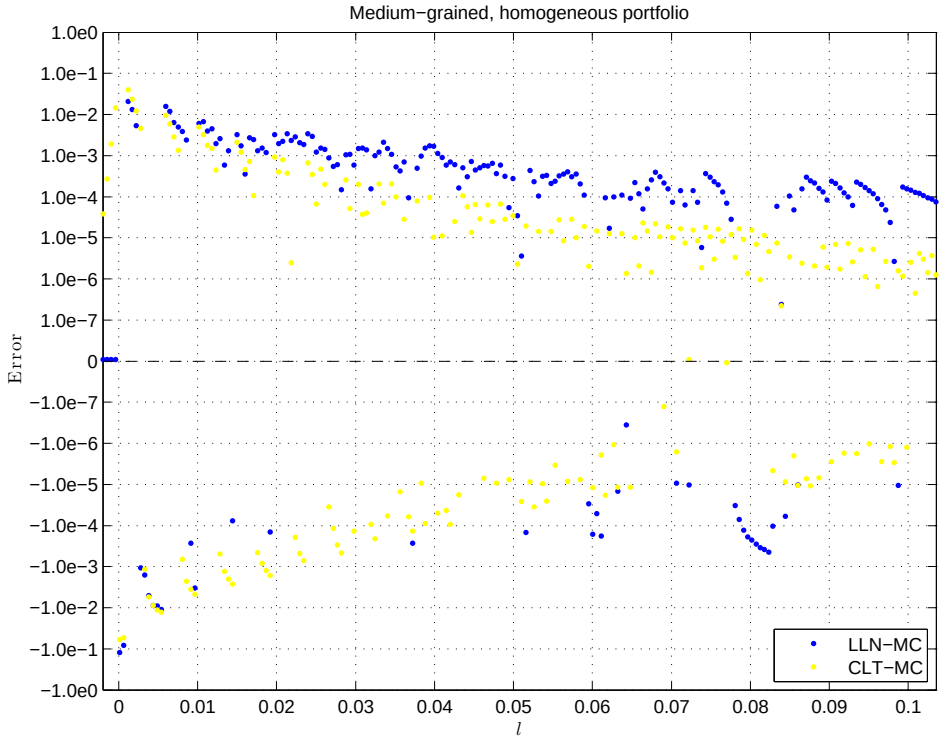


Figure 4.14: Comparison of the errors for the LLN and CLT approximations: Portfolios Π_2

(a) Errors: Portfolio Π_2, *blue:* $e_{lln}(l)$, *yellow:* $e_{clt}(l)$



(b) Difference in errors: Portfolio Π_2, *cyan:* $|e_{clt}(l)| - |e_{lln}(l)|$

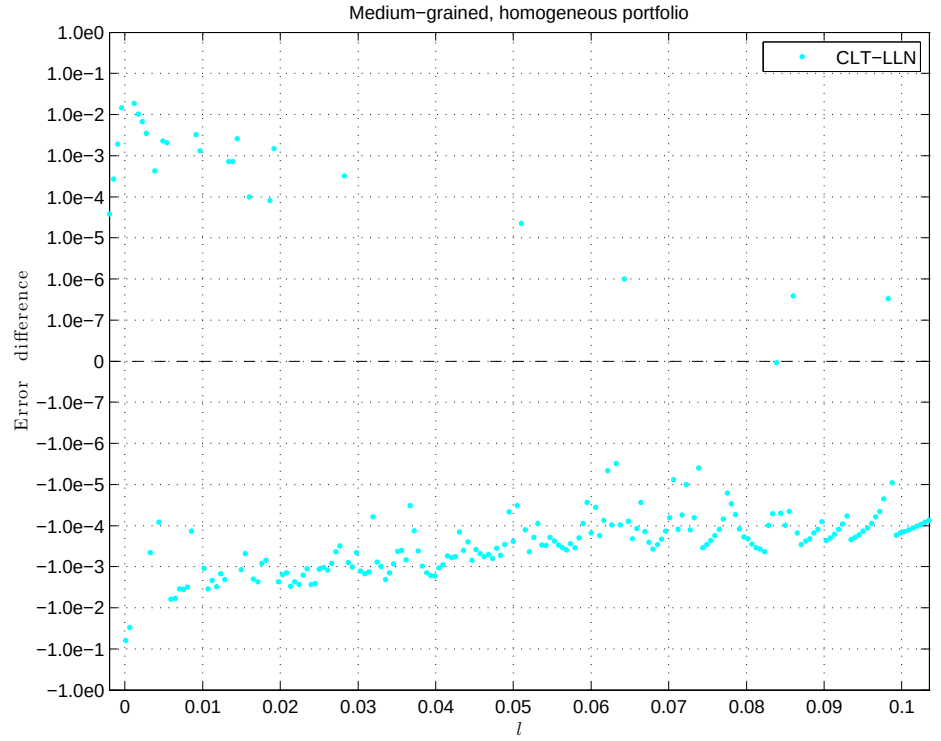
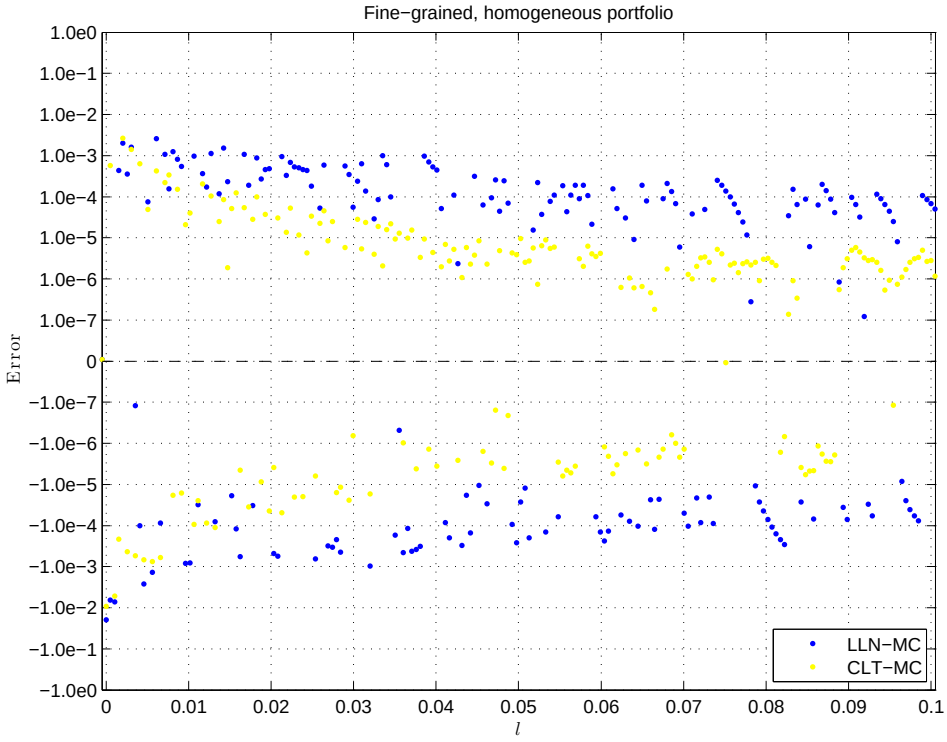


Figure 4.15: Comparison of the errors for the LLN and CLT approximations: Portfolios Π_3

(a) Errors: Portfolio Π_3, *blue:* $e_{lln}(l)$, *yellow:* $e_{clt}(l)$



(b) Difference in errors: Portfolio Π_3, *cyan:* $|e_{clt}(l)| - |e_{lln}(l)|$

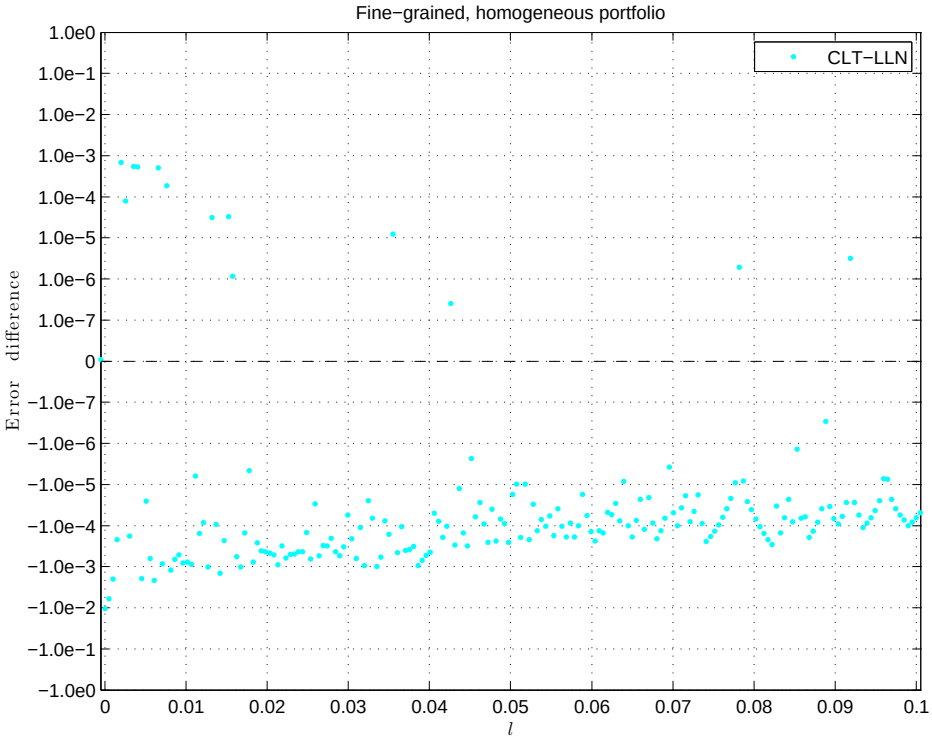
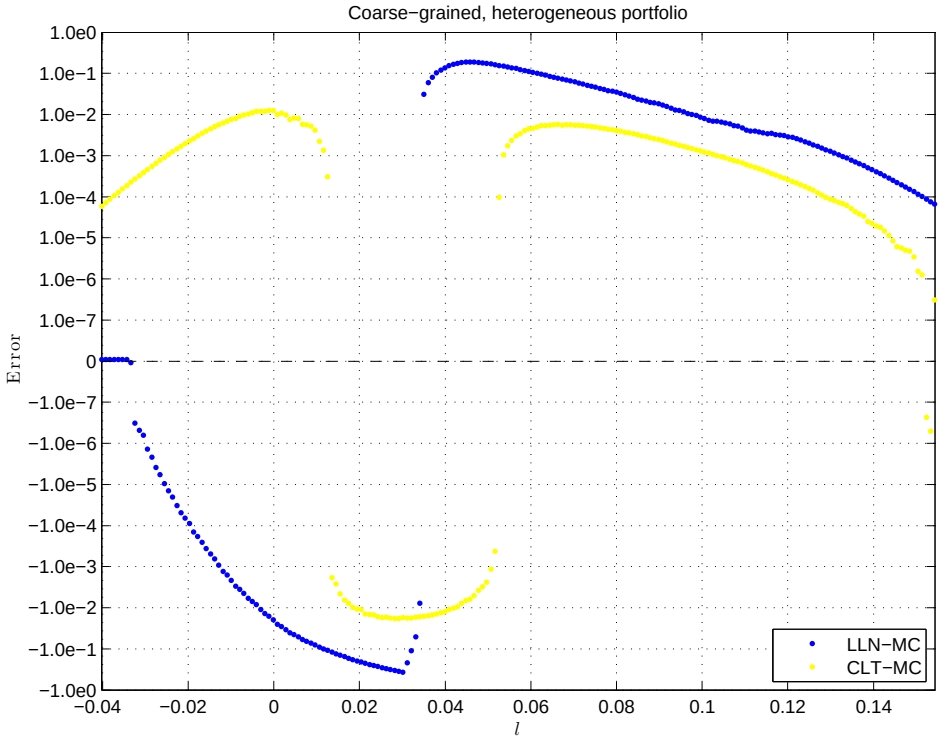


Figure 4.16: Comparison of the errors for the LLN and CLT approximations: Portfolios Π_4

(a) Errors: Portfolio Π_4, *blue:* $e_{lln}(l)$, *yellow:* $e_{clt}(l)$



(b) Difference in errors: Portfolio Π_4, *cyan:* $|e_{clt}(l)| - |e_{lln}(l)|$

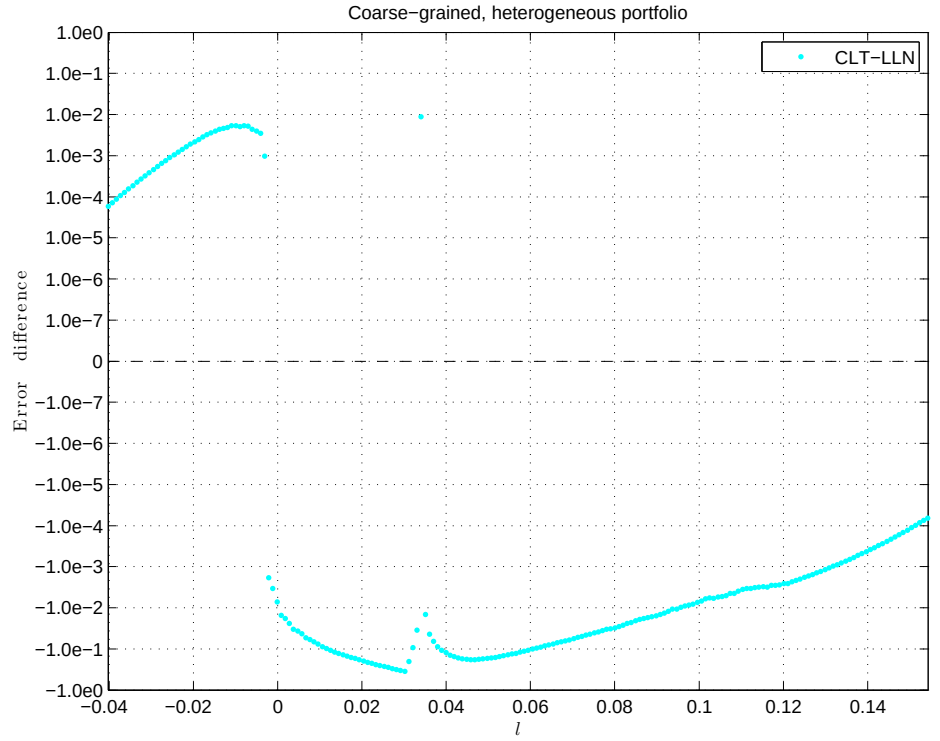
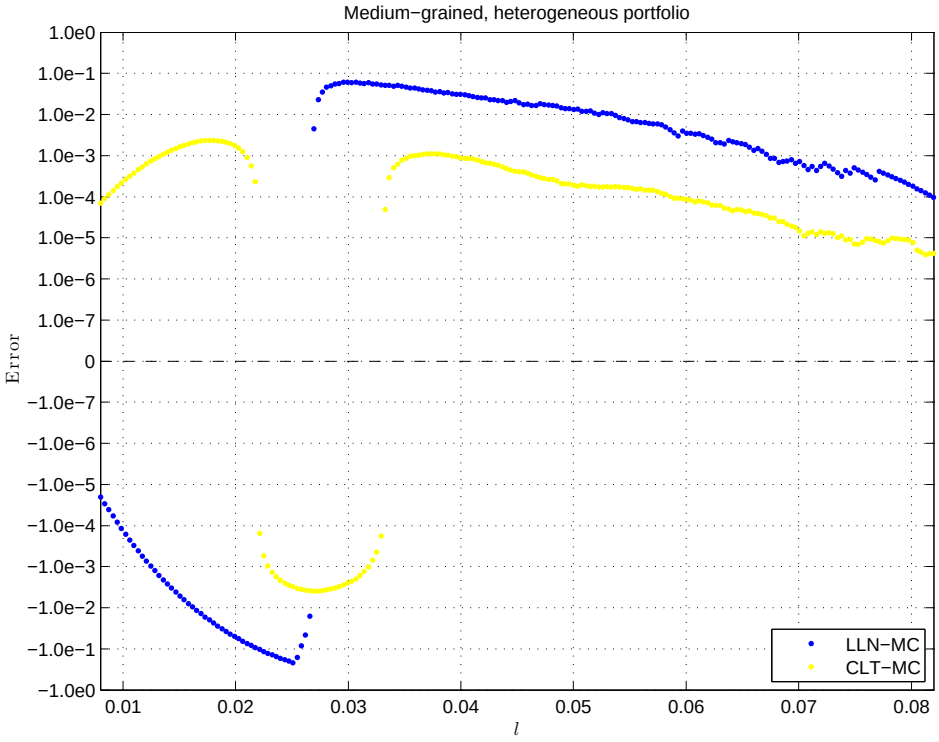


Figure 4.17: Comparison of the errors for the LLN and CLT approximations: Portfolios Π_5

(a) Errors: Portfolio Π_5, *blue:* $e_{lln}(l)$, *yellow:* $e_{clt}(l)$



(b) Difference in errors: Portfolio Π_5, *cyan:* $|e_{clt}(l)| - |e_{lln}(l)|$

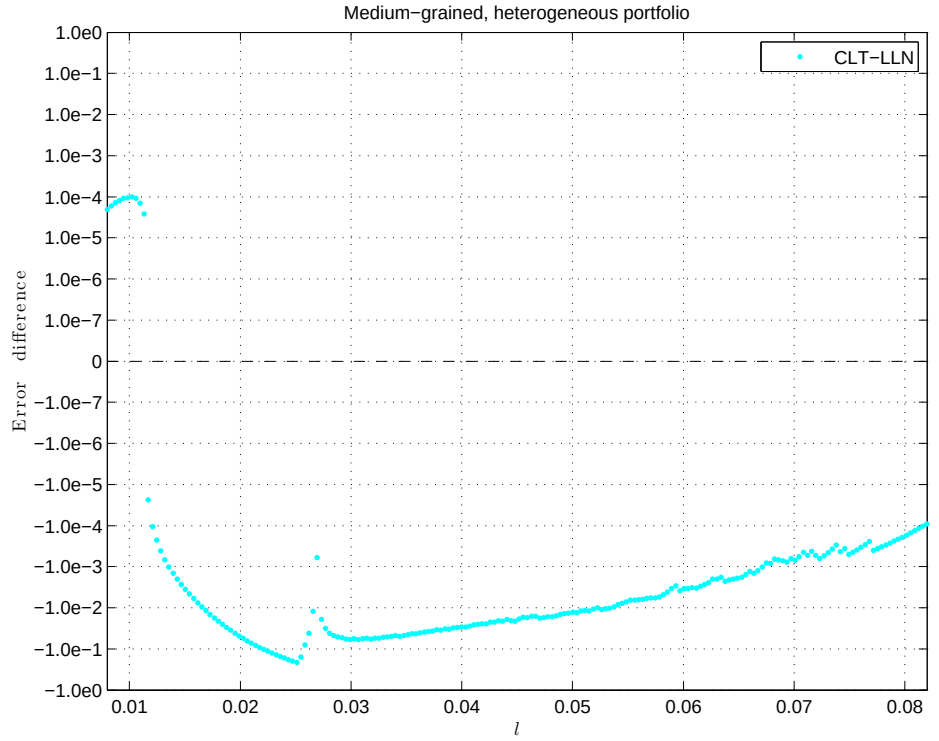
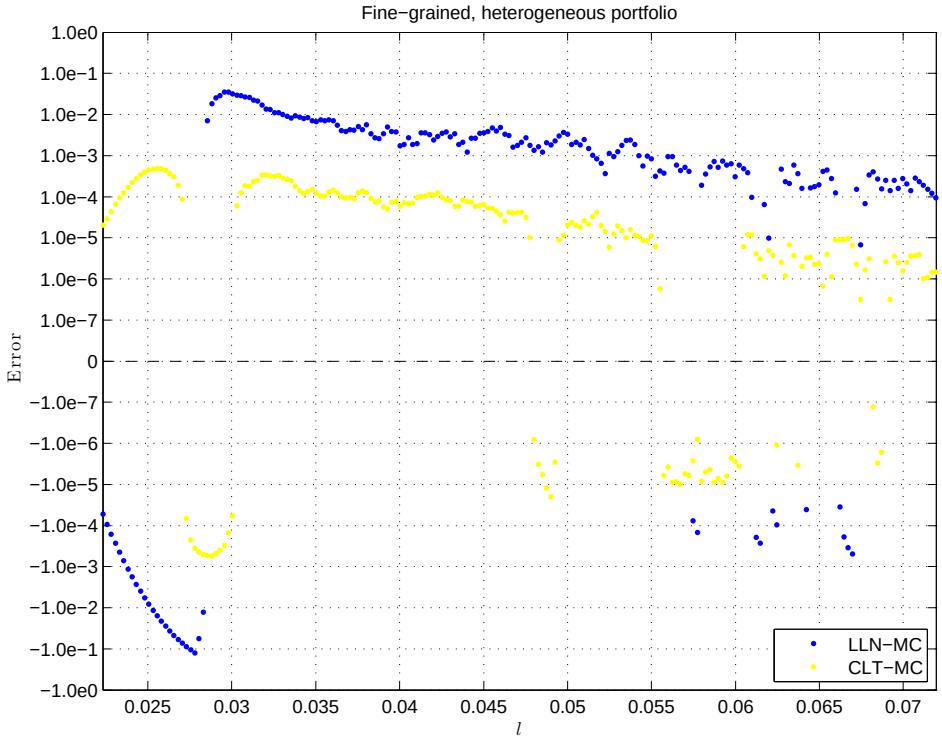
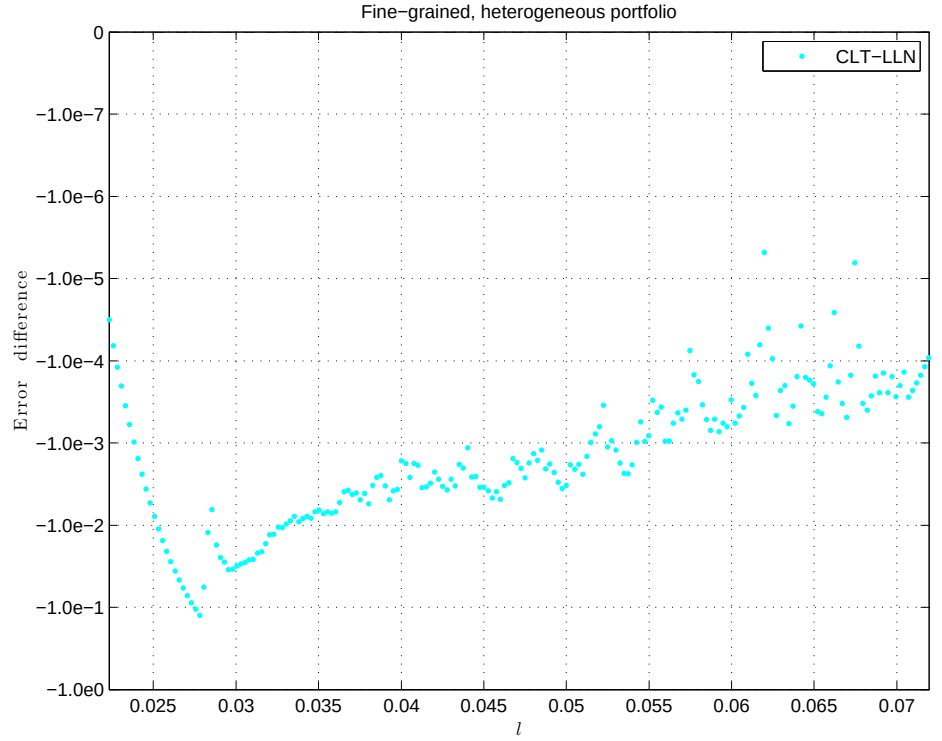


Figure 4.18: Comparison of the errors for the LLN and CLT approximations: Portfolios Π_6

(a) Errors: Portfolio Π_6, *blue:* $e_{lln}(l)$, *yellow:* $e_{clt}(l)$



(b) Difference in errors: Portfolio Π_6, *cyan:* $|e_{clt}(l)| - |e_{lln}(l)|$



We make the following observations about the numerical results in Figures 4.7 - 4.18.

1. The LLN and CLT approximations both work better as the portfolios become more homogeneous and more fine-grained. This is due to the asymptotic nature of both approximations.

2. The top subplots of Figures 4.13 - 4.18 show that the error associated with the CLT approximation ($e_{\text{CLT}}(l)$) is usually closer to zero than the error associated with the LLN approximation ($e_{\text{LLN}}(l)$). Equivalently, $\Delta e(l)$ is mostly below zero in the bottom subplots of Figures 4.13 - 4.18. That is, our numerical results support our belief that the CLT approximation is more accurate than the LLN approximation in most cases. In particular, the big gap between the LLN and CLT loss probabilities in Figure 4.10a, and the large negative values for $\Delta e(l)$ in Figure 4.16b suggest that the LLN approximation is not accurate for small heterogeneous portfolios. On the other hand, the CLT approximation produces usable results in these cases. Also, when the quantile l is large, the CLT approximation produces accurate probabilities $\left(|e_{\text{CLT}}| < 10^{-4}\right)$. Moreover, the CLT approximation usually produces better results than the LLN approximation in the high probability tail. All of these observations make the CLT approximation more attractive than the LLN approximation for downside risk measures, such as VaR and ES. The only exception is in the left tail of the distribution, where the CLT approximation is less accurate than the LLN approximation. However, the left tail is less important than the right tail in practice. For homogeneous portfolios, in the top subplots of Figures 4.13 - 4.18, $e_{\text{CLT}}(l)$ is scattered on both sides of zero, which implies the error comes mainly from approximating a discrete distribution by a continuous one. For heterogeneous portfolios, the LLN approximation outperforms the CLT approximation for the small loss probabilities for the portfolios Π_4 and Π_5, but the difference is very small, as can be seen in Figure 4.16b and Figure 4.17b.

3. The error associated with the CLT approximation, $e_{\text{CLT}}(l)$, in Figures 4.13 - 4.15 is scattered with jumps between positive and negative values. This is most pronounced in Figure 4.15a. The plots of $e_{\text{CLT}}(l)$ appear more "continuous" in the top subplots of Figures 4.16 - 4.18. As explained earlier, this is because the loss distribution for the MC approximation has large jumps for the homogeneous portfolios, while the loss distribution for the MC approximation for heterogeneous portfolios have smaller jumps. The distribution generated by the CLT approximation is continuous, therefore the graph of the errors for the CLT distribution is more "jagged" for homogeneous portfolios than for heterogeneous portfolios.

4. From the top subplots of Figures 4.16 - 4.18, we see that $e_{\text{LLN}}(l)$ and $e_{\text{CLT}}(l)$ are often above

zero for the large values of the quantile l plotted. This shows that both approximations tend to underestimate the risk. However, $e_{\mathrm{CLT}}(l)$ is below $e_{\mathrm{LLN}}(l)$ for most of the large values of the quantile l plotted, which verifies once again that the CLT approximation is more accurate than the LLN approximation.

Chapter 5

Exact Methods

Since the conditional portfolio loss is the sum of independent individual losses, the conditional portfolio loss probability can be computed by an $N-$fold discrete linear convolution, which leads to another class of approaches to compute the conditional portfolio loss probability: exact methods [3, 13, 15, 41, 43]. Unlike the asymptotic methods, which suffer asymptotic errors due to the LLN or CLT approximations, and the MC simulation, which endures sampling errors, the exact methods do not produce any approximation error, although they are still subject to roundoff errors and possibly aliasing errors. However, the existing exact methods are very computationally intensive. To improve the efficiency of the exact methods, we exploit the sparsity that often arises in the obligors' conditional losses to accelerate the computation.

In Section 5.1.2, we first review the standard full convolution method, then we develop a sparse convolution method, and further improve its efficiency by extending it to a truncated sparse convolution method. Similarly, in Section 5.2, we first discuss the full FFT algorithm, then propose a sparse FFT method and a truncated sparse FFT method with exponential windowing to reduce the aliasing error. Finally we present some numerical results and compare different methods in Section 5.3.

5.1 Sparse Convolution Method

5.1.1 Full Convolution Method

Let $L_n^c \doteq \omega_n \mathrm{LGC}_n^c$, then (3.3.3) shows that the conditional loss from obligor n, denoted by $\mathcal{L}_n^{\boldsymbol{z}}$, is a discrete random variable with support $\mathbb{S}_n = \left\{L_n^0, \ldots, L_n^{C-1}\right\}$ and probability mass function (PMF)

$$P_n^{\boldsymbol{z}}(x) = \begin{cases} \Phi\left(\frac{H_{c(n)}^0 - \boldsymbol{\beta}_n^T \boldsymbol{z}}{\sqrt{1-\boldsymbol{\beta}_n^T \boldsymbol{\beta}_n}}\right), & \text{if } x = L_n^0, \\ \Phi\left(\frac{H_{c(n)}^1 - \boldsymbol{\beta}_n^T \boldsymbol{z}}{\sqrt{1-\boldsymbol{\beta}_n^T \boldsymbol{\beta}_n}}\right) - \Phi\left(\frac{H_{c(n)}^0 - \boldsymbol{\beta}_n^T \boldsymbol{z}}{\sqrt{1-\boldsymbol{\beta}_n^T \boldsymbol{\beta}_n}}\right), & \text{if } x = L_n^1, \\ \vdots & \vdots \\ \Phi\left(\frac{H_{c(n)}^{C-1} - \boldsymbol{\beta}_n^T \boldsymbol{z}}{\sqrt{1-\boldsymbol{\beta}_n^T \boldsymbol{\beta}_n}}\right) - \Phi\left(\frac{H_{c(n)}^{C-2} - \boldsymbol{\beta}_n^T \boldsymbol{z}}{\sqrt{1-\boldsymbol{\beta}_n^T \boldsymbol{\beta}_n}}\right), & \text{if } x = L_n^{C-1}, \\ 0, & \text{otherwise.} \end{cases} \tag{5.1.1}$$

Thus, the conditional portfolio loss, $\mathcal{L}^{\boldsymbol{z}}$, is a sum of independent discrete random variables $\mathcal{L}_n^{\boldsymbol{z}}$: $\mathcal{L}^{\boldsymbol{z}} = \sum_{n=0}^{N-1} \mathcal{L}_n^{\boldsymbol{z}}$. Elementary probability theory shows that the sum of two independent discrete random variables is a discrete random variable with PMF given by the linear convolution of the PMFs of the two summands. That is,

$$\mathbb{P}\left\{\mathcal{L}_0^{\boldsymbol{z}} + \mathcal{L}_1^{\boldsymbol{z}} = x\right\} = P_0^{\boldsymbol{z}} * P_1^{\boldsymbol{z}}(x) \doteq \sum_{x_0 \in \mathbb{S}_0} P_0^{\boldsymbol{z}}(x_0) P_1^{\boldsymbol{z}}(x - x_0). \tag{5.1.2}$$

Consequently, for N independent discrete random variables $\mathcal{L}_0^{\boldsymbol{z}}, \ldots, \mathcal{L}_{N-1}^{\boldsymbol{z}}$, the PMF of their sum is the N-fold linear convolution of the PMFs of each $\mathcal{L}_n^{\boldsymbol{z}}$:

$$\mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} = x\right\} = \left(\overset{N-1}{\underset{n=0}{\circledast}} P_n^{\boldsymbol{z}}\right)(x), \tag{5.1.3}$$

where the N-fold linear convolution is defined recursively by

$$\begin{cases} \left(\overset{N-1}{\underset{n=0}{\circledast}} P_n^{\boldsymbol{z}}\right)(x) = \left(\left(\overset{N-2}{\underset{n=0}{\circledast}} P_n^{\boldsymbol{z}}\right) * P_{N-1}^{\boldsymbol{z}}\right)(x) \\ \left(\overset{1}{\underset{n=0}{\circledast}} P_n^{\boldsymbol{z}}\right)(x) = \left(P_0^{\boldsymbol{z}} * P_1^{\boldsymbol{z}}\right)(x). \end{cases}$$

Therefore, the conditional portfolio loss probability can be computed by

$$\mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} \leq l\right\} = \sum_{x \leq l} \left(\overset{N-1}{\underset{n=0}{\circledast}} P_n^{\boldsymbol{z}}\right)(x). \tag{5.1.4}$$

To compute (5.1.4), we must discretize the portfolio losses. Let

$$L_{min} \doteq \sum_{n=0}^{N-1} L_n^{C-1}, \tag{5.1.5}$$

$$L_{max} \doteq \sum_{n=0}^{N-1} L_n^{0}, \tag{5.1.6}$$

be the minimum and maximum portfolio loss, respectively. Then choose δ, K_{max} and K_{min} to construct a "perfect" discretization grid. That is, for every possible choice of L_n^c, $n = 0, \ldots, N-1$ and $c = 0, \ldots, C-1$,

$$\sum_{n=0}^{N-1} L_n^c = k\delta, \tag{5.1.7}$$

for some $k \in \{K_{min}, \ldots, K_{max}\}$, where

$$\begin{aligned} L_{min} &= K_{min}\delta, \\ L_{max} &= K_{max}\delta. \end{aligned}$$

To obtain the perfect discretization grid, we can find the optimal $\delta = \delta^*$ such that

$$\delta^* = \max\{\delta \,|k_n^c \doteq L_n^c/\delta \in \mathbb{N},\ \forall n \in \{0, \ldots, N-1\}, \forall c \in \{0, \ldots, C-1\}\}.$$

Therefore, $K_{min} = \sum_{n=0}^{N-1} k_n^{C-1}$ and $K_{max} = \sum_{n=0}^{N-1} k_n^0$, and the number of points on this grid is

$$\begin{aligned} K &= \frac{L_{max} - L_{min}}{\delta} + 1 \\ &= K_{max} - K_{min} + 1 \\ &= \sum_{n=0}^{N-1} \left(k_n^0 - k_n^{C-1}\right) + 1. \end{aligned}$$

Note that all L_n^c and their sums are on the grid. Therefore, there is no discretization error in computing (5.1.4).[1]

It is worth mentioning that, subject to certain level of discretization error, one could round some of the values of l_k in (5.1.8) below to a coarser grid, $l_k \approx k\hat{\delta}$ for $\hat{\delta} > \delta$, to reduce the total number of points, K, on the grid and consequently to accelerate the computation. Though we do not pursue this option in this thesis, one could combine this coarsening approach with the other techniques described in this

[1]The discretization error we are referring to is the error resulting from some L_n^c, or their sums, not being included on the discretization grid for the portfolio losses. For example, if the gird is [0, 0.01, 0.02, ..., 0.05], but one of L_n^c has the value 0.012, then we must round L_n^c to some point on the grid. This leads to a discretization error in representing L_n^c.

thesis to achieve a more efficient, but less accurate, approximation to the conditional portfolio loss.

Based on this discretization scheme, the portfolio losses are restricted to the values

$$l_k = k\delta, \tag{5.1.8}$$

for $k = K_{min}, \ldots, K_{max}$, and the distribution of the conditional loss from each obligor can be discretized by

$$\boldsymbol{p}_n^{\boldsymbol{z}} = \left[p_n^{\boldsymbol{z}}[K_{min}], \ldots, p_n^{\boldsymbol{z}}[K_{max}]\right]^T,$$

where $p_n^{\boldsymbol{z}}[k] = P_n^{\boldsymbol{z}}(l_k)$ represents the probability that the loss from obligor n is l_k conditional on $\boldsymbol{\mathcal{Z}} = \boldsymbol{z}$. The conditional portfolio loss (5.1.3) can be computed by an N-fold linear convolution of the vectors $\boldsymbol{p}_n^{\boldsymbol{z}}$:

$$\mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} = l_k\right\} = \left(\overset{N-1}{\underset{n=0}{\circledast}} \boldsymbol{p}_n^{\boldsymbol{z}}\right)[k], \tag{5.1.9}$$

where the linear convolution of two real vectors $\boldsymbol{x} = [x[0], \ldots, x[K_x - 1]]^T$ and $\boldsymbol{y} = [y[0], \ldots, y[K_y - 1]]^T$ is a mapping from $\mathbb{R}^{K_x} \times \mathbb{R}^{K_y}$ to $\mathbb{R}^{K_x + K_y - 1}$ defined by

$$\boldsymbol{x} * \boldsymbol{y}[k] \doteq \sum_{u+v=k} x[u]y[v], \tag{5.1.10}$$

for $k = 0, \ldots, K_x + K_y - 2$ with the understanding that the indices u and v do not go out of range (i.e., $0 \le u \le K_x - 1$ and $0 \le v \le K_y - 1$). Therefore, the conditional portfolio loss probability in (5.1.4) can be written as

$$\mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} \le l_j\right\} = \sum_{k \le j} \left(\overset{N-1}{\underset{n=0}{\circledast}} \boldsymbol{p}_n^{\boldsymbol{z}}\right)[k]. \tag{5.1.11}$$

As we mentioned in Section 2.3, LGC_n^c can be negative if obligor n is upgraded, which results in negative L_n^c. If we use the discretization scheme (5.1.8), then some negative indices k will be encountered, which complicates an implementation of this scheme. To solve this problem, first notice that

$$\mathbb{P}\left\{\sum_{n=0}^{N-1} \mathcal{L}_n^{\boldsymbol{z}} = l_k\right\} = \mathbb{P}\left\{\sum_{n=0}^{N-1} \left(\mathcal{L}_n^{\boldsymbol{z}} - L_n^{C-1}\right) = l_k - \sum_{n=0}^{N-1} L_n^{C-1}\right\}$$

for $k = K_{min}, K_{min} + 1 \ldots, K_{max} - 1, K_{max}$. Define shifted conditional individual losses by

$$\underline{\mathcal{L}}_n^{\boldsymbol{z}} \doteq \mathcal{L}_n^{\boldsymbol{z}} - L_n^{C-1},$$

then $0 \leq \underline{\mathcal{L}}_n^{\boldsymbol{z}} \leq L_n^0 - L_n^{C-1}$, since L_n^{C-1} is the minimum of the possible values of $\mathcal{L}_n^{\boldsymbol{z}}$. Therefore, the discretized distribution of $\underline{\mathcal{L}}_n^{\boldsymbol{z}}$ on the grid is $\underline{\boldsymbol{p}}_n^{\boldsymbol{z}} \doteq \left[\underline{p}_n^{\boldsymbol{z}}[0], \ldots, \underline{p}_n^{\boldsymbol{z}}[k_n^0 - k_n^{C-1}]\right]^T$. Hence,

$$\begin{aligned}
\mathbb{P}\left\{\sum_{n=0}^{N-1} \mathcal{L}_n^{\boldsymbol{z}} = l_k\right\} &= \mathbb{P}\left\{\sum_{n=0}^{N-1} \underline{\mathcal{L}}_n^{\boldsymbol{z}} = l_k - \sum_{n=0}^{N-1} L_n^{C-1}\right\}, \\
&= \mathbb{P}\left\{\sum_{n=0}^{N-1} \underline{\mathcal{L}}_n^{\boldsymbol{z}} = (k - K_{min})\,\delta\right\} \\
&= \left(\overset{n}{\underset{j=0}{\circledast}} \underline{\boldsymbol{p}}_j^{\boldsymbol{z}}\right)[k - K_{min}]\,. \qquad (5.1.12)
\end{aligned}$$

To compute (5.1.12), we can apply the convolution methods described earlier in this subsections, since the indices of $\underline{\boldsymbol{p}}_n^{\boldsymbol{z}}$ are all nonnegative.

Without loss of generality, we assume from now on that $L_{min} = 0$. Therefore,

$$l_k = k\delta,\ k = 0, \ldots, K-1, \qquad (5.1.13)$$

$$\delta = L_{max}/(K-1), \qquad (5.1.14)$$

and $\boldsymbol{p}_n^{\boldsymbol{z}} = \left[p_n^{\boldsymbol{z}}[0], \ldots, p_n^{\boldsymbol{z}}[K-1]\right]^T$. However, from (5.1.12), we see that our approach easily extends to handle the case that $L_{min} < 0$.

One way to analyze the complexity of the computation in (5.1.10) is to view the discrete linear convolution (5.1.10) as a Toepliz matrix[2] generated by the vector $\boldsymbol{y}$ multiplying the other vector $\boldsymbol{x}$. For example, the linear convolution of two vectors $\boldsymbol{x} = [x_0, x_1, x_2, x_3, x_4, x_5]^T$ and $\boldsymbol{y} = [y_0, y_1, y_2, y_3]^T$ can be computed as

[2] A Toepliz matrix is a matrix whose negative-sloping diagonals are constant. For example,

$$\mathbf{A} = \begin{bmatrix} a & e & f & g & h \\ b & a & e & f & g \\ c & b & a & e & f \\ d & c & b & a & e \end{bmatrix}.$$

$$\boldsymbol{x} * \boldsymbol{y} = \mathbf{T}_{\boldsymbol{y}}\boldsymbol{x} = \begin{bmatrix} y_0 & 0 & 0 & 0 & 0 & 0 \\ y_1 & y_0 & 0 & 0 & 0 & 0 \\ y_2 & y_1 & y_0 & 0 & 0 & 0 \\ y_3 & y_2 & y_1 & y_0 & 0 & 0 \\ 0 & y_3 & y_2 & y_1 & y_0 & 0 \\ 0 & 0 & y_3 & y_2 & y_1 & y_0 \\ 0 & 0 & 0 & y_3 & y_2 & y_1 \\ 0 & 0 & 0 & 0 & y_3 & y_2 \\ 0 & 0 & 0 & 0 & 0 & y_3 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}.$$

The floating-point operations (FLOPs) required to compute $\boldsymbol{x} * \boldsymbol{y}$ are the FLOPs associated with the nonzero elements in this matrix-vector product. So, assuming that all x_k in $\boldsymbol{x}$ and y_k in $\boldsymbol{y}$ are nonzero, it is clear that the computation of $\boldsymbol{x} * \boldsymbol{y}[k]$ requires

- $k+1$ multiplications and k additions for each $k = 0, 1, \ldots, K_y - 1$;
- K_y multiplications and $K_y - 1$ additions for $k = K_y, \ldots, K_x - 2$;
- $(K_x + K_y - 1) - k$ multiplications and $(K_x + K_y - 1) - k - 1$ additions for each $k = K_x - 1, \ldots, K_x + K_y - 2$.

Therefore,

$$\sum_{k=0}^{K_y-1} (k+1) + \sum_{k=K_y}^{K_x-2} K_y + \sum_{k=K_x-1}^{K_x+K_y-2} (K_x + K_y - 1) - k = K_x K_y$$

multiplications and $K_x K_y - (K_x + K_y - 1)$ additions, which is $2K_x K_y - (K_x + K_y - 1)$ FLOPs in total, are required to compute $\boldsymbol{x} * \boldsymbol{y}$.

Since the maximum portfolio loss is $L_{max} = l_{K-1}$, and there are no negative losses, the sum of the conditional losses from two obligors can not exceed l_{K-1}, which implies $\boldsymbol{p}_0^{\boldsymbol{z}} * \boldsymbol{p}_1^{\boldsymbol{z}}[k] = 0$ for $k \geq K$. Thus, it is not necessary to compute $\boldsymbol{p}_0^{\boldsymbol{z}} * \boldsymbol{p}_1^{\boldsymbol{z}}[k] = 0$ for $k \geq K$. Therefore, to compute $\boldsymbol{p}_0^{\boldsymbol{z}} * \boldsymbol{p}_1^{\boldsymbol{z}}[k]$ in (5.1.10) for $k = 0, \ldots, K-1$, where $K_x = K_y = K$, we need a total of $\sum_{k=0}^{K-1}(k+1) = K(K+1)/2$ multiplications and $\sum_{k=0}^{K-1} k = (K-1)K/2$ additions, for a total of K^2 FLOPs. The same argument can be applied to the computation of all linear convolutions in the N-fold linear convolution in (5.1.9). Therefore, the total number of FLOPs to compute the whole distribution of the conditional portfolio loss, $\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} = l_k\}$, $k = 0, \ldots, K-1$, is

$$(N-1)K^2 = O\left(NK^2\right). \tag{5.1.15}$$

One obvious improvement over the straightforward convolution method described above is to take advantage of the trailing zeros in $\boldsymbol{p}_n^{\boldsymbol{z}}$. Notice that $p_n^{\boldsymbol{z}}[k] = 0$ for $k > k_n^0$, whence we can avoid all

multiplications and additions involving with $p_n^{\boldsymbol{z}}[k]$ for $k > k_n^0$. Therefore, define $\overline{\boldsymbol{p}}_n^{\boldsymbol{z}} = \boldsymbol{p}_n^{\boldsymbol{z}}\left[0 : k_n^0\right]$,[3] and note that, for $k = 0, \ldots K-1$, we have $\left(\overset{N-1}{\underset{j=0}{\circledast}} \overline{\boldsymbol{p}}_j^{\boldsymbol{z}}\right)[k] = \left(\overset{N-1}{\underset{j=0}{\circledast}} \boldsymbol{p}_j^{\boldsymbol{z}}\right)[k]$. To analyze the number of FLOPs to compute $\left(\overset{N-1}{\underset{j=0}{\circledast}} \bar{\boldsymbol{p}}_j^{\boldsymbol{z}}\right)[k]$, we denote the length of $\bar{\boldsymbol{p}}_n^{\boldsymbol{z}}$ and $\overset{n}{\underset{j=0}{\circledast}} \bar{\boldsymbol{p}}_j^{\boldsymbol{z}}$ to be K_n and $\tilde{K}_n$ respectively for $n = 0, \ldots N-1$, then $\tilde{K}_0 = K_0$, and $\tilde{K}_n = \tilde{K}_{n-1} + K_n - 1$ for $n = 1, \ldots N-1$. The total number of FLOPs to compute $\overset{N-1}{\underset{j=0}{\circledast}} \bar{\boldsymbol{p}}_j^{\boldsymbol{z}}$ is

$$\sum_{n=1}^{N-1} 2\tilde{K}_{n-1}K_n - \left(\tilde{K}_{n-1} + K_n - 1\right). \tag{5.1.16}$$

If all $\bar{\boldsymbol{p}}_j^{\boldsymbol{z}}$ have the same length $\bar{K}$, then for $n = 0, \ldots N-1$,

$$K_n = \bar{K} \tag{5.1.17}$$

$$\tilde{K}_n = \tilde{K}_{n-1} + \bar{K} - 1 = (n+1)\bar{K} - n, \tag{5.1.18}$$

and

$$K = \tilde{K}_{N-1} = N\bar{K} - (N-1). \tag{5.1.19}$$

Substituting (5.1.17) and (5.1.18) into (5.1.16), the total number of FLOPs becomes

$$\frac{\left(2\bar{K}-1\right)\left(\bar{K}-1\right)N(N-1)}{2} + (N-1)\bar{K}. \tag{5.1.20}$$

From (5.1.19) we know

$$\bar{K} = \frac{K-1}{N} + 1. \tag{5.1.21}$$

Substituting (5.1.21) into (5.1.20), the total number of FLOPs in (5.1.16) is

$$\left(1 - \frac{1}{N}\right)K^2 + \left(\frac{N-3}{2} + \frac{1}{N}\right)K + \frac{N-1}{2}, \tag{5.1.22}$$

which is $O\left(K^2 + NK\right)$.

The number of points on the grid, K, can be very large, since a fine grid may be needed to reduce or to eliminate the discretization error. Also, the number of obligors in the portfolio, N, can be thousands for large financial institutions. What's worse, from (3.3.5), we see that we need to compute

[3] We adapt the notation in MATLAB to represent the indices of a vector. The notation $m : k : n$ means to access every kth element in the subvector, starting with the mth element of the original vector and ending at (or before) the nth element of the original vector. If $k = 1$, we leave out the step value k and use $m : n$ instead of $m : 1 : n$. For example, if $\boldsymbol{x} = [x_0, x_1, x_2, x_3, x_4, x_5]^T$, then $\boldsymbol{x}[0 : 2 : 5] = [x_0, x_2, x_4]^T$ and $\boldsymbol{x}[1 : 3] = [x_1, x_2, x_3]^T$. This notation is used throughout this thesis.

the conditional loss distribution U times for different realizations of systematic risk factors. Therefore, this straightforward convolution method and its obvious improvements may be very computationally expensive.

5.1.2 Sparse Convolution Method

To improve the efficiency of the convolution method, we take a closer look at the conditional probability vectors $\boldsymbol{p}_n^{\boldsymbol{z}} = \left[p_n^{\boldsymbol{z}}[0], \ldots, p_n^{\boldsymbol{z}}[K-1]\right]^T$. Notice that, from the PMF in (5.1.1), $\mathcal{L}_n^{\boldsymbol{z}}$ is a discrete random variable with C possible values. The number of rating classes, C, is much less than the length of $\boldsymbol{p}_n^{\boldsymbol{z}}$ (or $\bar{\boldsymbol{p}}_n^{\boldsymbol{z}}$). For example, for long-term corporate bonds, S&P's rating system has 10 rating classes: AAA, AA, A, BBB, BB, B, CCC, CC, C, and D.[4] Other rating agencies' rating systems, as well as financial institutions' internal rating systems, have a similar structure. The number of rating classes for short-term bonds is even smaller. Therefore, all $\boldsymbol{p}_n^{\boldsymbol{z}}$ (or $\bar{\boldsymbol{p}}_n^{\boldsymbol{z}}$) are highly sparse vectors with $C \ll K$ (or $C \ll \bar{K}$) nonzero elements. From (5.1.9) and (5.1.10), it is not difficult to see that the convolution method wastes many FLOPs on multiplying zeros and adding zeros. Hence, it is a natural step to develop a method to speed up the convolution method by taking advantage of the sparsity of $\boldsymbol{p}_n^{\boldsymbol{z}}$.

5.1.2.1 Sequential Sparse Convolution Method

Let $\tilde{\boldsymbol{p}}_0^{\boldsymbol{z}} \doteq \boldsymbol{p}_0^{\boldsymbol{z}}$ and $\tilde{\boldsymbol{p}}_n^{\boldsymbol{z}} \doteq \overset{n}{\underset{j=0}{\circledast}} \boldsymbol{p}_j^{\boldsymbol{z}}$ for $n = 1, \ldots, N-1$. Then $\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} = l_k\} = \tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}}[k]$ and $\tilde{\boldsymbol{p}}_n^{\boldsymbol{z}}$ can be computed recursively by $\tilde{\boldsymbol{p}}_{n+1}^{\boldsymbol{z}} = \tilde{\boldsymbol{p}}_n^{\boldsymbol{z}} * \boldsymbol{p}_{n+1}^{\boldsymbol{z}}$ for $n = 0, \ldots, N-2$. For each obligor n, define a C-element vector $\boldsymbol{d}_n = \left[d_n[0], \ldots, d_n[C-1]\right]^T$, where $d_n[c]$ represents the index of L_n^c on the grid. That is,

$$d_n[c] \doteq L_n^c/\delta, \tag{5.1.23}$$

for $c = 0, \ldots, C-1$. Also, define another C-element vector $\boldsymbol{q}_n^{\boldsymbol{z}} = \left[q_n^{\boldsymbol{z}}[0], \ldots, q_n^{\boldsymbol{z}}[C-1]\right]^T$, where $q_n^{\boldsymbol{z}}[c]$ is the probability that obligor n's loss is L_n^c: $q_n^{\boldsymbol{z}}[c] \doteq p_n^{\boldsymbol{z}}[d_n[c]]$, for $c = 0, \ldots, C-1$. In other words, $\boldsymbol{q}_n^{\boldsymbol{z}}$ contains all the nonzero conditional loss probabilities for obligor n. Similarly, we can define $\tilde{\boldsymbol{d}}_n$ to be the vector of indices of the nonzero elements in $\tilde{\boldsymbol{p}}_n^{\boldsymbol{z}}$.[5]

[4] As presented in Table 2.1, S&P has a finer rating system, in which rating class AA is divided into AA+, AA and AA-, rating class A is divided into A+, A and A-, rating class BBB is divided into BBB+, BBB and BBB, rating class BB is divided into BB+, BB and BB-, rating class B is divided into B+, B and B- and rating class CCC is divided into CCC+, CCC and CCC-. Therefore, there are 22 rating classes in this finer system.

[5] Though $\boldsymbol{q}_n^{\boldsymbol{z}}$ contains all the nonzero conditional loss probabilities for obligor n, it does not mean that all elements in $\boldsymbol{q}_n^{\boldsymbol{z}}$ are nonzero since some $q_n^{\boldsymbol{z}}[c]$ are allowed to be zero. However, these zero elements in $\boldsymbol{q}_n^{\boldsymbol{z}}$ are treated as structurally nonzero in our algorithm. Therefore, $\boldsymbol{d}_n$ is actually the vector of indices of the structurally nonzero elements $\boldsymbol{p}_n^{\boldsymbol{z}}$. A similar argument applies to $\tilde{\boldsymbol{p}}_n^{\boldsymbol{z}}$ and $\tilde{\boldsymbol{d}}_n$. From now on, for convolution methods, the term "nonzero elements" refer to "structurally nonzero elements" described in this footnote.

If we use the formula in (5.1.10) to compute $\tilde{\boldsymbol{p}}_1^{\boldsymbol{z}}$, then

$$\tilde{\boldsymbol{p}}_1^{\boldsymbol{z}}[k] = \sum_{i+j=k} p_0^{\boldsymbol{z}}[i] p_1^{\boldsymbol{z}}[j] \tag{5.1.24}$$

$$= \sum_{(i,j)\in\mathbb{D}_k} q_0^{\boldsymbol{z}}[i] q_1^{\boldsymbol{z}}[j], \tag{5.1.25}$$

where $\mathbb{D}_k = \{(i,j) | \, d_0[i] + d_1[j] = k\}$. If we ignore for now the FLOPs required to determine $\mathbb{D}_k$, the computation in (5.1.25) requires at most $2C-1$ FLOPs (at most C multiplications and $C-1$ additions), while, as discussed in the previous subsection, a naive implementation of (5.1.24) requires $k+1$ multiplications and k additions for $k = 0, \ldots, K-1$. Therefore, (5.1.25) requires much fewer FLOPs than (5.1.24) for most k. However, the computational overhead in determining $\mathbb{D}_k$ can not be ignored. Since there is no special pattern for $\boldsymbol{d}_n$, for each k, we need to search in $\boldsymbol{d}_0$ and $\boldsymbol{d}_1$ to find the pairs (i,j) such that $d_0[i] + d_1[j] = k$. This overhead is not heavy in computing $\tilde{\boldsymbol{p}}_1^{\boldsymbol{z}}$, since both $\boldsymbol{d}_0$ and $\boldsymbol{d}_1$ have only C elements. However, to compute $\tilde{\boldsymbol{p}}_n^{\boldsymbol{z}}$ for $n > 1$, we need to search in $\tilde{\boldsymbol{d}}_{n-1}$ (rather than $\boldsymbol{d}_{n-1}$) and $\boldsymbol{d}_n$ to find such pairs (i,j). The length of $\tilde{\boldsymbol{d}}_n$ increases as n increases since the number of nonzero probabilities in $\tilde{\boldsymbol{p}}_n^{\boldsymbol{z}}$ increases after each convolution. Therefore, the overhead to find pairs (i,j) becomes more and more significant as n increases. Moreover, for many values of k, $\tilde{\boldsymbol{p}}_1^{\boldsymbol{z}}[k]$ is zero, so there is no pair (i,j) such that $d_0[i] + d_1[j] = k$. Hence some computation is wasted in determining that $\mathbb{D}_k = \emptyset$ for many values of k.

5.1.2.2 Concurrent Sparse Convolution Method

A better way to compute the linear convolution $\tilde{\boldsymbol{p}}_{n+1}^{\boldsymbol{z}} = \tilde{\boldsymbol{p}}_n^{\boldsymbol{z}} * \boldsymbol{p}_{n+1}^{\boldsymbol{z}}$ is to compute $\tilde{\boldsymbol{d}}_n$ and $\tilde{\boldsymbol{p}}_{n+1}^{\boldsymbol{z}}$ concurrently, rather than computing them sequentially as described above. The main idea of computing all elements of $\tilde{\boldsymbol{p}}_{n+1}^{\boldsymbol{z}}$ concurrently is to compute the product of nonzero elements of $\tilde{\boldsymbol{p}}_n^{\boldsymbol{z}}$ and $\boldsymbol{p}_{n+1}^{\boldsymbol{z}}$, and assign it to, or add it to, the appropriate element of $\tilde{\boldsymbol{p}}_{n+1}^{\boldsymbol{z}}$. We consider $\tilde{\boldsymbol{p}}_1^{\boldsymbol{z}} = \boldsymbol{p}_0^{\boldsymbol{z}} * \boldsymbol{p}_1^{\boldsymbol{z}}$ as an example to demonstrate how to compute elements of $\tilde{\boldsymbol{p}}_n^{\boldsymbol{z}}$ concurrently.

Consider the example, illustrated in Figure 5.1, for which $\boldsymbol{p}_0^{\boldsymbol{z}}$ is a vector with three nonzero elements at positions $d_0[0]$, $d_0[1]$, and $d_0[2]$, with values $q_0^{\boldsymbol{z}}[0]$, $q_0^{\boldsymbol{z}}[1]$ and $q_0^{\boldsymbol{z}}[2]$, respectively, and $\boldsymbol{p}_1^{\boldsymbol{z}}$ has a similar structure.

First, we initialize all K elements in $\tilde{\boldsymbol{p}}_1^{\boldsymbol{z}}$ to $-\epsilon_{\mathsf{M}}$, where ϵ_{M} is the smallest positive normalized floating-point number in the implementation environment, then for $j = 0$, we compute

$$d_0[i] + d_1[0] \text{ and } q_0^{\boldsymbol{z}}[i] q_1^{\boldsymbol{z}}[0]$$

and assign $d_0[i] + d_1[0]$ to $\tilde{d}_1[i]$ and assign $q_0^{\boldsymbol{z}}[i]q_1^{\boldsymbol{z}}[0]$ to $\tilde{p}_1^{\boldsymbol{z}}\left[\tilde{d}_1[i]\right]$ for $i = 0, \ldots, C-1$. This is illustrated in Figure 5.2.

Figure 5.1: The structure of $\boldsymbol{p}_0^{\boldsymbol{z}}$ and $\boldsymbol{p}_1^{\boldsymbol{z}}$

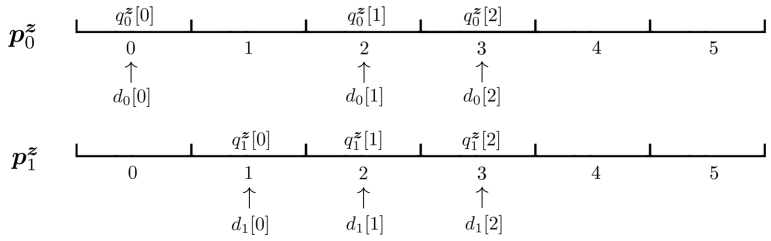
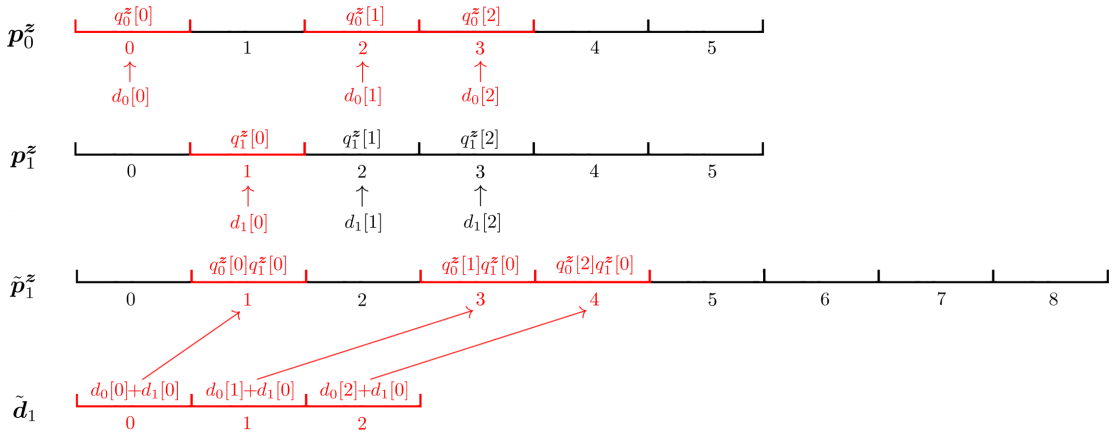


Figure 5.2: Concurrent computation of the linear convolution $\tilde{\boldsymbol{p}}_1^{\boldsymbol{z}} = \boldsymbol{p}_0^{\boldsymbol{z}} * \boldsymbol{p}_1^{\boldsymbol{z}}$ $(j = 0)$



Next, for $j = 1$, we compute

$$d_0[i] + d_1[1] \text{ and } q_0^{\boldsymbol{z}}[i]q_1^{\boldsymbol{z}}[1]$$

for $i = 0, \ldots, C-1$. However, the way that we put $d_0[i] + d_1[1]$ into $\tilde{\boldsymbol{d}}_1$ and $q_0^{\boldsymbol{z}}[i]q_1^{\boldsymbol{z}}[1]$ into $\tilde{\boldsymbol{p}}_1^{\boldsymbol{z}}$ is a little more complex for $j = 1$ than for $j = 0$, since there might exist a and b such that

$$d_0[a] + d_1[0] = d_0[b] + d_1[1]. \tag{5.1.26}$$

That is, there might exist some k such that the set $\mathbb{D}_k$ in (5.1.25) has more than one pair (i, j). For instance, as shown in Figure 5.3, for $k = 4$, we have $d_0[2] + d_1[0] = d_0[1] + d_1[1] = k$. In this case, we should not add an additional element with value $d_0[1] + d_1[1]$ into the vector $\tilde{\boldsymbol{d}}_1$, since this value already exists in $\tilde{\boldsymbol{d}}_1$. Also, we should add the value of $q_0^{\boldsymbol{z}}[1]q_1^{\boldsymbol{z}}[1]$ to $\tilde{p}_1\left[d_0[1] + d_1[1]\right]$, rather than assigning the value of $q_0^{\boldsymbol{z}}[1]q_1^{\boldsymbol{z}}[1]$ to $\tilde{p}_1\left[d_0[1] + d_1[1]\right]$. By the construction of $\tilde{\boldsymbol{p}}_1^{\boldsymbol{z}}$ and $\tilde{\boldsymbol{d}}_1$, to check whether an index, k, is in $\tilde{\boldsymbol{d}}_1$, we only need to check whether $\tilde{p}_1^{\boldsymbol{z}}[k] \geq 0$. Therefore, for $j = 1$, the vectors $\tilde{\boldsymbol{p}}_1^{\boldsymbol{z}}$ and $\tilde{\boldsymbol{d}}_1$ can be

updated concurrently as follows for all $i = 0, \ldots, C-1$.

if $\tilde{p}_1^{\tilde{\boldsymbol{z}}}[d_0[i] + d_1[1]] < 0$ **then**
  $\tilde{d}_1[\text{end}+1] \leftarrow d_0[i] + d_1[1]$;
  $\tilde{p}_1^{\tilde{\boldsymbol{z}}}[d_0[i] + d_1[1]] \leftarrow q_0^{\tilde{\boldsymbol{z}}}[i] q_1^{\tilde{\boldsymbol{z}}}[1]$;
else
  $\tilde{p}_1^{\tilde{\boldsymbol{z}}}[d_0[i] + d_1[1]] \leftarrow q_0^{\tilde{\boldsymbol{z}}}[i] q_1^{\tilde{\boldsymbol{z}}}[1] + \tilde{p}_1^{\tilde{\boldsymbol{z}}}[d_0[i] + d_1[1]]$;
end if

The computation of the linear convolution $\tilde{\boldsymbol{p}}_1^{\tilde{\boldsymbol{z}}} = \boldsymbol{p}_0^{\tilde{\boldsymbol{z}}} * \boldsymbol{p}_1^{\tilde{\boldsymbol{z}}}$ for $j = 1$ is illustrated in Figure 5.3. For $j > 1$, the computation is similar to that for $j = 1$. Figure 5.4 shows the computation for $j = 2$ in our example. As we can see, the final result of $\tilde{\boldsymbol{p}}_1^{\tilde{\boldsymbol{z}}}$ calculated concurrently is the same as that computed sequentially.

Figure 5.3: Concurrent computation of the linear convolution $\tilde{\boldsymbol{p}}_1^{\tilde{\boldsymbol{z}}} = \boldsymbol{p}_0^{\tilde{\boldsymbol{z}}} * \boldsymbol{p}_1^{\tilde{\boldsymbol{z}}}$ $(j = 1)$

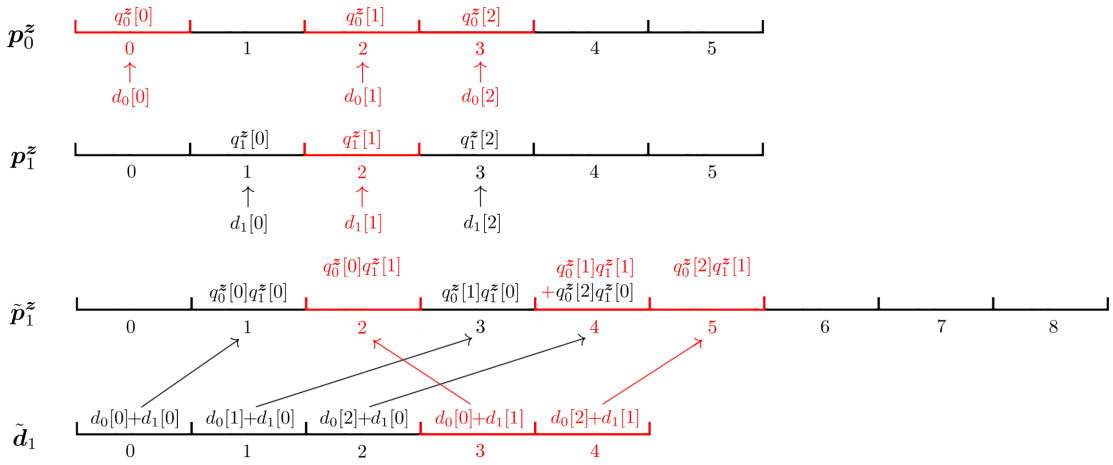
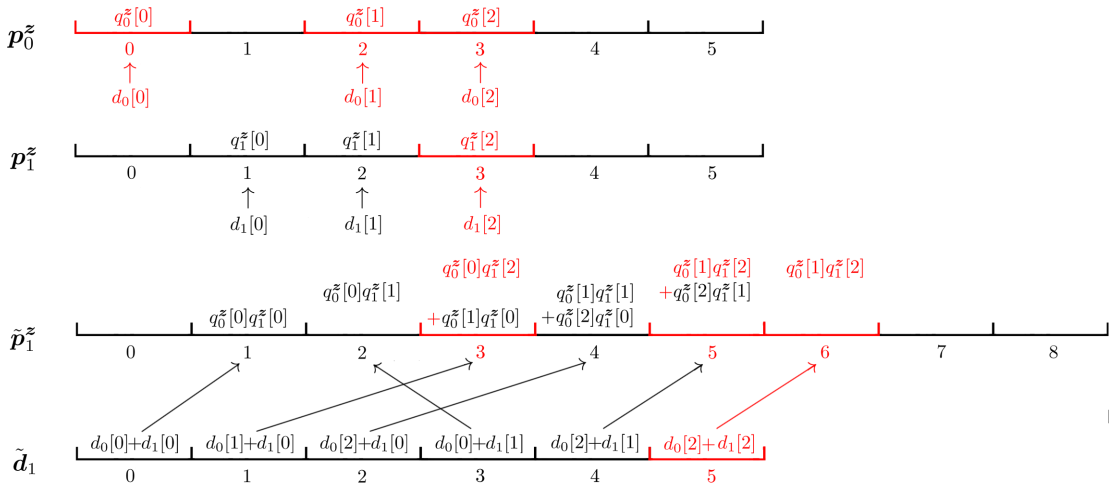


Figure 5.4: Concurrent computation of the linear convolution $\tilde{\boldsymbol{p}}_1^{\tilde{\boldsymbol{z}}} = \boldsymbol{p}_0^{\tilde{\boldsymbol{z}}} * \boldsymbol{p}_1^{\tilde{\boldsymbol{z}}}$ $(j = 2)$



We can apply the method discussed above to compute the $N-$fold linear convolution recursively. At the $(n+1)^{st}$ step, we compute the two-fold linear convolution

$$\tilde{\boldsymbol{p}}_{n+1}^{\boldsymbol{z}} = \tilde{\boldsymbol{p}}_{n}^{\boldsymbol{z}} * \boldsymbol{p}_{n+1}^{\boldsymbol{z}}$$

concurrently using $\tilde{\boldsymbol{p}}_{n}^{\boldsymbol{z}}$ and $\tilde{\boldsymbol{d}}_n$, which are computed in the previous step. Notice that, though we do not store the vector, $\tilde{\boldsymbol{q}}_{n}^{\boldsymbol{z}}$, which holds all nonzero elements of $\tilde{\boldsymbol{p}}_{n}^{\boldsymbol{z}}$, each element of $\tilde{\boldsymbol{q}}_{n}^{\boldsymbol{z}}$ can be obtained easily from vector $\tilde{\boldsymbol{p}}_{n}^{\boldsymbol{z}}$, since $\tilde{\boldsymbol{q}}_{n}^{\boldsymbol{z}}[i] = \tilde{\boldsymbol{p}}_{n}^{\boldsymbol{z}}\left[\tilde{\boldsymbol{d}}_n[i]\right]$. Also, we do not need to store $\tilde{\boldsymbol{p}}_{n}^{\boldsymbol{z}}$ and $\tilde{\boldsymbol{d}}_n$ for all n: we only need to store the $\tilde{\boldsymbol{p}}_{n}^{\boldsymbol{z}}$ and $\tilde{\boldsymbol{d}}_n$ calculated in the previous step. This iterative procedure to compute the $N-$fold linear convolution is summarized in Algorithm 5.1.

Algorithm 5.1 Sparse convolution method to compute the conditional loss probabilities $\tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}}$

Input: $\boldsymbol{q}_n^{\boldsymbol{z}}$, $\boldsymbol{d}_n$ for $n = 0, \ldots, N-1$;
Output: $\tilde{\boldsymbol{p}} = \tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}}$, $\tilde{\boldsymbol{d}} = \tilde{\boldsymbol{d}}_{N-1}$;

$\boldsymbol{p} \leftarrow -\boldsymbol{\epsilon}_{\mathbf{M}}$;
$\boldsymbol{d} \leftarrow \boldsymbol{d}_0$;
for $c = 0 : C-1$ **do**
    $p\,[d[c]] \leftarrow q_0^{\boldsymbol{z}}[c]$;
end for
$\tilde{\boldsymbol{p}} \leftarrow \boldsymbol{p}$;
$\tilde{\boldsymbol{d}} \leftarrow \boldsymbol{d}$;
for $n = 1 : N-1$ **do**
    $\tilde{\boldsymbol{p}} \leftarrow -\boldsymbol{\epsilon}_{\mathbf{M}}$;
    $\tilde{\boldsymbol{d}} \leftarrow$ `NULL`;
    for $j = 0 : C-1$ **do**
        for $i = 0 :$ `length`$(\boldsymbol{d}) - 1$ **do**
            $k \leftarrow d[i] + d_n[j]$;
            if $\tilde{p}[k] < 0$ **then**
                $\tilde{d}[\texttt{end}+1] \leftarrow k$;
                $\tilde{p}[k] \leftarrow p[d[i]] \times q_n^{\boldsymbol{z}}[j]$;
            else
                $\tilde{p}[k] \leftarrow \tilde{p}[k] + p[d[i]] \times q_n^{\boldsymbol{z}}[j]$;
            end if
        end for
    end for
    $\boldsymbol{p} \leftarrow \tilde{\boldsymbol{p}}$;
    $\boldsymbol{d} \leftarrow \tilde{\boldsymbol{d}}$;
end for
return $\tilde{\boldsymbol{p}}$, $\tilde{\boldsymbol{d}}$;

By (3.3.5), we need to compute $\tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}}$ for each $\boldsymbol{z} = \boldsymbol{z}^{(u)}$, $u = 1, \ldots, U$. If we apply Algorithm 5.1 U

times to compute $\tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}^{(u)}}$, $u = 1, \ldots, U$, then we recompute $\tilde{\boldsymbol{d}}_n$, $n = 0, \ldots, N-1$, U times. However, from Line 11 and Line 13 in Algorithm 5.1, we see that $\tilde{\boldsymbol{d}}_n$ is updated from $\boldsymbol{d}_n$, and (5.1.23) shows that $\boldsymbol{d}_n$ does not depend on $\boldsymbol{z}$. Hence $\tilde{\boldsymbol{d}}_n$ does not depend on $\boldsymbol{z}$ either. Therefore, we need to compute $\tilde{\boldsymbol{d}}_n$ once only (not U times) to compute (3.3.5). Based on these observations, instead of applying Algorithm 5.1 U times sequently to compute $\tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}^{(u)}}$ for $u = 1, \ldots, U$, we give an algorithm to calculate $\tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}^{(u)}}$ concurrently, where $\tilde{\boldsymbol{d}}_n$ is computed only once. The algorithm is presented in Algorithm 5.2.

Algorithm 5.2 Improved sparse convolution method to compute the conditional loss probabilities $\tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}^{(u)}}$

Input: $\boldsymbol{q}_n^{(u)} = \boldsymbol{q}_n^{\boldsymbol{z}^{(u)}}$, $\boldsymbol{d}_n$ for $n = 0, \ldots, N-1, u = 1, \ldots, U$;
Output: $\tilde{\boldsymbol{p}}^{(u)} = \tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}^{(u)}}$, $\tilde{\boldsymbol{d}} = \tilde{\boldsymbol{d}}_{N-1}$, for $u = 1, \ldots, U$;

1: $\boldsymbol{p}^{(u)} \leftarrow -\boldsymbol{\epsilon}_{\mathbf{M}}$, for $u = 1, \ldots, U$;
2: $\boldsymbol{d} \leftarrow \boldsymbol{d}_0$;
3: **for** $c = 0 : C-1$ **do**
4: $\quad p^{(u)}[d[c]] \leftarrow q_0^{(u)}[c]$, for $u = 1, \ldots, U$;
5: **end for**
6: $\tilde{\boldsymbol{p}}^{(u)} \leftarrow \boldsymbol{p}^{(u)}$;
7: $\tilde{\boldsymbol{d}} \leftarrow \boldsymbol{d}$;
8: **for** $n = 1 : N-1$ **do**
9: $\quad \tilde{\boldsymbol{p}}^{(u)} \leftarrow -\boldsymbol{\epsilon}_{\mathbf{M}}$, for $u = 1, \ldots, U$;
10: $\quad \tilde{\boldsymbol{d}} \leftarrow$ `NULL`;
11: $\quad$ **for** $j = 0 : C-1$ **do**
12: $\quad\quad$ **for** $i = 0 :$ `length`$(\boldsymbol{d}) - 1$ **do**
13: $\quad\quad\quad k \leftarrow d[i] + d_n[j]$;
14: $\quad\quad\quad$ **if** $\tilde{p}^{(u)}[k] < 0$ **then**
15: $\quad\quad\quad\quad \tilde{d}[\texttt{end} + 1] \leftarrow k$;
16: $\quad\quad\quad\quad \tilde{p}^{(u)}[k] \leftarrow p^{(u)}[d[i]] \times q_n^{(u)}[j]$, for $u = 1, \ldots, U$;
17: $\quad\quad\quad$ **else**
18: $\quad\quad\quad\quad \tilde{p}^{(u)}[k] \leftarrow \tilde{p}^{(u)}[k] + p^{(u)}[d[i]] \times q_n^{(u)}[j]$, for $u = 1, \ldots, U$;
19: $\quad\quad\quad$ **end if**
20: $\quad\quad$ **end for**
21: $\quad$ **end for**
22: $\quad \boldsymbol{p}^{(u)} \leftarrow \tilde{\boldsymbol{p}}^{(u)}$, for $u = 1, \ldots, U$;
23: $\quad \boldsymbol{d} \leftarrow \tilde{\boldsymbol{d}}$;
24: **end for**
25: **return** $\tilde{\boldsymbol{p}}^{(u)}$, $\tilde{\boldsymbol{d}}$;

5.1.2.3 Complexity of Concurrent Sparse Convolution Method

Compared with the full convolution method, the speedup of the concurrent sparse convolution method comes from avoiding unnecessary multiplications and additions involving zeros. Consequently, the ef-

ficiency of the concurrent sparse convolution method depends on the sparsity of the vectors $\tilde{\boldsymbol{p}}_n^{\boldsymbol{z}}$. The following theorem gives the relationship between the complexity of this method and the sparsity of $\tilde{\boldsymbol{p}}_n^{\boldsymbol{z}}$.

Theorem 5.1. *Let H_n be the number of nonzero elements in $\tilde{\boldsymbol{p}}_n^{\boldsymbol{z}}$ for all $\boldsymbol{z} = \boldsymbol{z}^{(u)}$, $u = 1, \ldots, U$, and assume that the vectors $\boldsymbol{q}_n^{\boldsymbol{z}}$ are of length C, for $n = 0, \ldots, N-1$ and all $\boldsymbol{z}$. Then the total number of FLOPs for the concurrent sparse convolution method (Algorithm 5.2) to compute $\tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}} = \overset{N-1}{\underset{j=0}{\circledast}} \boldsymbol{p}_j^{\boldsymbol{z}}$ is*

$$\sum_{n=1}^{N-1} \left(\left(2 + \frac{1}{U} \right) CH_{n-1} - H_n \right) \approx \sum_{n=1}^{N-1} \left(2CH_{n-1} - H_n \right). \tag{5.1.27}$$

Proof. At each iteration n, the FLOPs needed are as follows:

1. Additions to compute $\tilde{d}_{n-1}[i] + d_n[j]$ (Line 13 in Algorithm 5.1), for $i = 0, \ldots, H_{n-1} - 1$, $j = 0, \ldots, C-1$. Hence, there are CH_{n-1} additions in total. However, if we use Algorithm 5.2, as mentioned, $\tilde{\boldsymbol{d}}_n$ does not depend on $\boldsymbol{z}$; hence $\tilde{\boldsymbol{d}}_n$ needs to be computed only once for different $\boldsymbol{z}^{(u)}$, $u = 1, \ldots, U$. Therefore, for each $\boldsymbol{z}^{(u)}$, the amount of computation required to calculate $\tilde{\boldsymbol{d}}_n$ is CH_{n-1}/U.

2. Multiplications to compute $\tilde{p}_{n-1}^{\boldsymbol{z}} \left[\tilde{d}_{n-1}[i] + d_n[j] \right] \times q_n^{\boldsymbol{z}}[j]$ (Line 16 or Line 18 in Algorithm 5.1), for $i = 0, \ldots, H_{n-1} - 1$, $j = 0, \ldots, C-1$. Hence there are CH_{n-1} multiplications in total.

3. Additions to compute the summation of $p[k]$ and $\tilde{p}_{n-1}^{\boldsymbol{z}} \left[\tilde{d}_{n-1}[i] + d_n[j] \right] \times q_n^{\boldsymbol{z}}[j]$ (Line 18 in Algorithm 5.1) if needed. If the additions were performed no matter whether $\tilde{p}[d[i] + d_n[j]]$ is negative or not, then there would be CH_{n-1} additions. However, the additions are performed only when $\tilde{p}_n^{\boldsymbol{z}} \left[\tilde{d}_{n-1}[i] + d_n[j] \right] \geq 0$, and from Line 15 it is clear that once the number of nonzero elements in $\tilde{\boldsymbol{p}}_n^{\boldsymbol{z}}$ is increased by one, then one addition in Line 18 is skipped. Therefore, the number of additions that are skipped (because Line 16 is executed, rather than Line 18) is H_n. Hence, the number of additions is $CH_{n-1} - H_n$.

Summing the FLOPs above, there are $\left(2 + \frac{1}{U}\right) CH_{n-1} - H_n$ FLOPs at each step $n = 1, \ldots, N-1$. Hence, the total number of FLOPs to compute the N-fold linear convolution $\tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}} = \overset{N-1}{\underset{j=0}{\circledast}} \boldsymbol{p}_j^{\boldsymbol{z}}$ is $\sum_{n=1}^{N-1} \left(\left(2 + \frac{1}{U}\right) CH_{n-1} - H_n \right)$. Since U is the sample size for the systematic risk factors, which is usually very large, $2 + 1/U \approx 2$, whence $\sum_{n=1}^{N-1} \left(\left(2 + \frac{1}{U}\right) CH_{n-1} - H_n \right) \approx \sum_{n=1}^{N-1} \left(2CH_{n-1} - H_n \right)$. □

Recall that, in our problem, $\boldsymbol{d}_n$ is related to the conditional individual loss rate by $d_n[c] \doteq L_n^c/\delta$, $c = 0, \ldots, C-1$. Therefore, there is no particular pattern to the elements of $\boldsymbol{d}_n$. Hence there is no general rule to determine H_n at each step. However, Theorem 5.1 allows us to analyze the complexity of the concurrent sparse method for fixed N and C in the best case and in the worse case. All analysis below is based on Algorithm 5.2, and the FLOP counts are for each u.

Notice that $H_n \leq CH_{n-1}$, since H_n equals the number of times Line 15 is executed, which can be at

most CH_{n-1} (i.e. at most once for each i and j in the nested loop). On the other hand, $H_n \geq H_{n-1}$, therefore, $H_n \in [H_{n-1}, CH_{n-1}]$. This indicates that the minimum/maximum number of FLOPs at each step n is an increasing function of H_{n-1}. Therefore, to find the best case, H_{n-1} should be minimized in each step n, while, to find the worst case, H_{n-1} should be maximized.

Best case Minimizing H_n requires all $\boldsymbol{d}_n$ to have the form

$$\boldsymbol{d}_n = [0, \alpha, 2\alpha, \ldots, (C-1)\alpha]^T, \tag{5.1.28}$$

for some $\alpha \in \mathbb{N}^+$. In this case, after each iteration, H_n has $C-1$ more elements than H_{n-1}:

$$H_n = H_{n-1} + (C-1). \tag{5.1.29}$$

To illustrate this, we consider $C = 3$ and $\boldsymbol{d}_0 = \boldsymbol{d}_1 = [0, \alpha, 2\alpha]^T$. Define a matrix $\mathbf{A}$ with elements $A_{ij} = d_0[i] + d_1[j]$, for $i = 0, 1, 2$ and $j = 0, 1, 2$, then

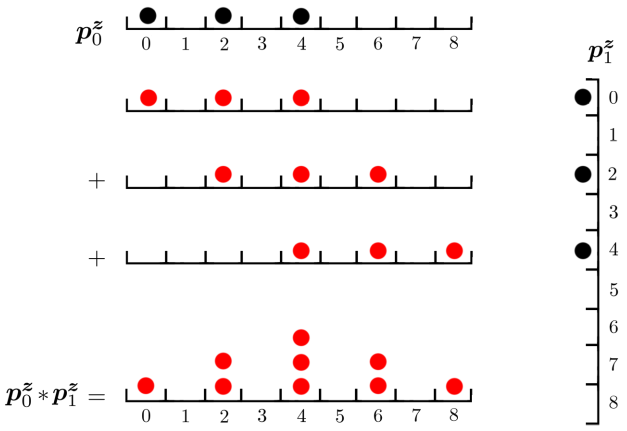
$$\mathbf{A} = \begin{bmatrix} 0 & \alpha & 2\alpha \\ \alpha & 2\alpha & 3\alpha \\ 2\alpha & 3\alpha & 4\alpha \end{bmatrix}.$$

Then it is easy to see that

$$\tilde{\boldsymbol{d}}_1 = [0, \alpha, 2\alpha, 3\alpha, 4\alpha]^T.$$

It is clear that the positive diagonals of $\mathbf{A}$ are the same, and compared with $\tilde{\boldsymbol{d}}_0 = \boldsymbol{d}_0$, only $C - 1 = 2$ extra elements, 3α and 4α, are added into $\tilde{\boldsymbol{d}}_1$. This can be seen more clearly in Figure 5.5. From (5.1.29)

Figure 5.5: Best case ($C = 3$, $\alpha = 2$)



and $H_0 = C$, we have

$$H_n = C + n(C-1). \tag{5.1.30}$$

Substitute (5.1.30) into (5.1.27), the total number of FLOPs in the best case is

$$\Psi^b_{sconv} = \frac{1}{2}\left(2C^2 - 3C + 1\right)N^2 - \frac{1}{2}\left(2C^2 - 5C + 1\right)N - C. \tag{5.1.31}$$

Worst case To maximize H_n, consider the pattern:

$$d_n[k] = \alpha k C^n, \tag{5.1.32}$$

for some $\alpha \in \mathbb{N}^+$, $k = 0, \ldots, C-1$ and $n = 0, \ldots, N-1$. In this case, Line 16 is never executed, and H_n and H_{n-1} have the relationship

$$H_n = CH_{n-1}. \tag{5.1.33}$$

To illustrate this, let $C = 3$, then $\boldsymbol{d}_0 = [0, \alpha, 2\alpha]^T$, $\boldsymbol{d}_1 = [0, 3\alpha, 6\alpha]^T$. Again, define the matrix $\mathbf{A}$ with elements $A_{ij} = d_0[i] + d_1[j]$ for $i = 0, 1, 2$, and $j = 0, 1, 2$. That is,

$$\mathbf{A} = \begin{bmatrix} 0 & \alpha & 2\alpha \\ 3\alpha & 4\alpha & 5\alpha \\ 6\alpha & 7\alpha & 8\alpha \end{bmatrix}.$$

Then it is easy to see that

$$\tilde{\boldsymbol{d}}_1 = [0, \alpha, 2\alpha, 3\alpha, 4\alpha, 5\alpha, 6\alpha, 7\alpha, 8\alpha]^T.$$

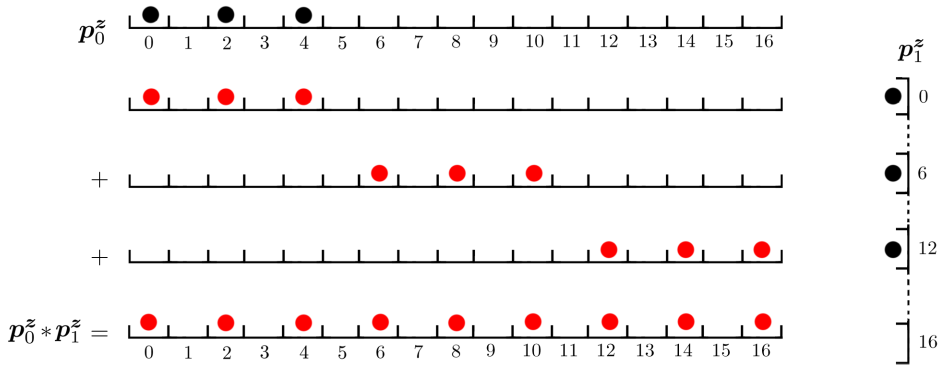
Notice that all elements in $\mathbf{A}$ are different, hence $(C-1)C$ extra elements are added into $\tilde{\boldsymbol{d}}_1$. This is illustrated in Figure 5.6. From (5.1.33) and $H_0 = C$, we have

$$H_n = C^{n+1}. \tag{5.1.34}$$

Substituting (5.1.34) into (5.1.27), we see that the total number of FLOPs in the worst case is

$$\Psi^w_{sconv} = \frac{C^2(C^{N-1} - 1)}{C - 1}. \tag{5.1.35}$$

Figure 5.6: Worst case ($C = 3$, $\alpha = 2$)



Recall that the number of FLOPs required by the convolution method discussed in Subsection 5.1.1 is

$$\Psi_{conv} = \sum_{n=1}^{N-1} 2\tilde{K}_{n-1}K_n + \left(\tilde{K}_{n-1} + K_n - 1\right), \tag{5.1.36}$$

where K_n and $\tilde{K}_n$ are the length of $\bar{\boldsymbol{p}}_n^{\boldsymbol{z}}$ and $\overset{n}{\underset{j=0}{\circledast}} \bar{\boldsymbol{p}}_j^{\boldsymbol{z}}$ respectively. In the best case,

$$K_n = (C-1)\alpha + 1 \tag{5.1.37}$$

for $n = 0, \ldots, N-1$, then by (5.1.20),

$$\begin{aligned}
\Psi_{conv}^b &= \frac{\left(2\bar{K}-1\right)\left(\bar{K}-1\right)N(N-1)}{2} + (N-1)\bar{K} \\
&= \left(\alpha^2(C-1)^2 + \frac{1}{2}\alpha(C-1)\right)N^2 + \left(-(C-1)^2\alpha^2 + \frac{1}{2}(C-1)\alpha\right)N - \left((C-1)\alpha + 1\right).
\end{aligned}$$

Consequently, the speedup of the concurrent sparse convolution method compared with the full convolution method in the best case is

$$\begin{aligned}
\frac{\Psi_{conv}^b}{\Psi_{sconv}^b} &= \frac{\left(\alpha^2(C-1)^2 + \frac{1}{2}\alpha(C-1)\right)N^2 + \left(-(C-1)^2\alpha^2 + \frac{1}{2}(C-1)\alpha\right)N - \left((C-1)\alpha+1\right)}{\frac{1}{2}\left(2C^2-3C+1\right)N^2 - \frac{1}{2}\left(2C^2-5C+1\right)N - C} \\
&= \frac{(C-1)^2\alpha^2N^2\left(\left(1+\frac{1}{2(C-1)\alpha}\right) + \left(\frac{1}{2(C-1)\alpha}-1\right)\frac{1}{N} - \left(\frac{1}{(C-1)\alpha} + \frac{1}{(C-1)^2\alpha^2}\right)\frac{1}{N^2}\right)}{(C-1)N^2\left(\frac{2C-1}{2} - \left(\frac{2C^2-5C+1}{2(C-1)}\right)\frac{1}{N} - \frac{C}{C-1}\frac{1}{N^2}\right)} \\
&\approx \frac{(C-1)^2\alpha^2N^2}{(C-1)^2N^2}\left(1+\frac{1}{2(C-1)\alpha}\right) \\
&= \alpha^2\left(1+\frac{1}{2(C-1)\alpha}\right) \sim \Omega(\alpha^2).
\end{aligned} \tag{5.1.38}$$

In the worst case, since $\tilde{K}_n = \tilde{K}_{n-1} + K_n - 1$, for $n = 1, \ldots, N-1$, and

$$K_n = \alpha\,(C-1)\,C^n + 1, \tag{5.1.39}$$

for $n = 0, \ldots, N-1$, then $\tilde{K}_n - \tilde{K}_{n-1} = K_n - 1 = \alpha\,(C-1)\,C^n$, which implies

$$\tilde{K}_n = 1 + \sum_{j=0}^{n} \alpha\,(C-1)\,C^j = \alpha C^{n+1} - (\alpha - 1), \tag{5.1.40}$$

for $n = 0, \ldots, N-1$. Hence, substituting (5.1.39) and (5.1.40) into (5.1.36), we have

$$\begin{aligned}
\Psi_{conv}^{w} &= \sum_{n=1}^{N-1} 2\tilde{K}_{n-1}K_n + \left(\tilde{K}_{n-1} + K_n - 1\right) \\
&= 2\alpha^2(C-1)\sum_{n=1}^{N-1} C^{2n} + (2(\alpha+C) - \alpha C)\sum_{n=1}^{N-1} C^n - 3(\alpha-1)(N-1) \\
&= \frac{2\alpha^2\left(C^{2N} - C^2\right)}{C+1} + \frac{(2(\alpha+C) - \alpha C)\left(C^N - C\right)}{C-1} - 3(\alpha-1)(N-1).
\end{aligned} \tag{5.1.41}$$

Consequently, the speedup of the concurrent sparse convolution method compared with the full convolution method in the worst case is

$$\begin{aligned}
\frac{\Psi_{conv}^{w}}{\Psi_{sconv}^{w}} &= \frac{\frac{2\alpha^2\left(C^{2N}-C^2\right)}{C+1} + \frac{(2(\alpha+C)-\alpha C)\left(C^N-C\right)}{C-1} - 3(\alpha-1)(N-1)}{\frac{C^2(C^{N-1}-1)}{C-1}} \\
&= \frac{\alpha^2 C^{2N}\left(\frac{2-\frac{C^2}{\alpha^2 C^{2N}}}{C+1} + \frac{(2(\alpha+C)-\alpha C)\frac{C^N-C}{\alpha^2C^{2N}}}{C-1} - \frac{3(\alpha-1)(N-1)}{\alpha^2 C^{2N}}\right)}{\frac{C^{N+1}\left(1-\frac{1}{C^{N-1}}\right)}{C-1}} \\
&\approx \frac{\frac{2\alpha^2C^{2N}}{C+1}}{\frac{C^{N+1}}{C-1}} \approx 2\alpha^2 C^{N-1} \sim \Omega\left(\alpha^2 C^N\right).
\end{aligned} \tag{5.1.42}$$

Comparing (5.1.38) and (5.1.42), we can see that the speedup in the worst case is much higher than that in the best case. The reason is that the vectors $\bar{\boldsymbol{p}}_n^{\boldsymbol{z}}$ are much longer in the worst case than in the best case, which follows from comparing (5.1.37) and (5.1.39). This implies that the vectors $\bar{\boldsymbol{p}}_n^{\boldsymbol{z}}$ in the worst case are much more sparse than those in the best case, since the number of nonzero elements in $\bar{\boldsymbol{p}}_n^{\boldsymbol{z}}$, which is C, is fixed. Therefore, in the worst case, the full convolution method wastes more FLOPs in computing multiplications and additions where at least one of the arguments is zero. Although the sparse convolution method is slower in the worst case than it is in the best case, the full convolution method is much slower in the worst case than it is in the best case; hence the speedup in the worst case is much higher than that in the best case.

5.1.3 Truncated Sparse Convolution Method

For the sparse convolution method described in the last subsection, we compute every nonzero $\tilde{p}^{\mathbf{z}}_{n-1}[i]p^{\mathbf{z}}_n[j]$ when computing $\tilde{\boldsymbol{p}}^{\mathbf{z}}_n$ at each step n. However, as shown in Figure 5.7, when n is large, many elements in $\tilde{\boldsymbol{p}}^{\mathbf{z}}_n$ become very small, since we multiply $\tilde{p}^{\mathbf{z}}_{n-1}[i]$, which is less than 1, by $p^{\mathbf{z}}_n[j]$, which is also less than 1. In our sparse convolution method, we treat those extremely small $\tilde{p}^{\mathbf{z}}_{n-1}[i]p^{\mathbf{z}}_n[j]$ as nonzero. Some of these very small values may create a new nonzero element (See Line 16 of Algorithm 5.1 or 5.2). This increases H_n and slows down the computation. However, financial institutions do not require extremely high accuracy. Therefore, it is reasonable to ignore extremely small $\tilde{p}^{\mathbf{z}}_{n-1}[i]p^{\mathbf{z}}_n[j]$. That is, instead of computing $\tilde{\boldsymbol{p}}^{\mathbf{z}}_n$ by

$$\tilde{p}^{\mathbf{z}}_n[k] = \sum_{u+v=k} \tilde{p}^{\mathbf{z}}_{n-1}[i]p^{\mathbf{z}}_n[j], \tag{5.1.43}$$

we approximate $\tilde{\boldsymbol{p}}^{\mathbf{z}}_n$ by its truncated version $\widehat{\tilde{\boldsymbol{p}}^{\mathbf{z}}_n}$, where

$$\begin{aligned}
&\widehat{\tilde{p}^{\mathbf{z}}_0}[k] = \tilde{p}^{\mathbf{z}}_0[k], \\
&\widehat{\tilde{p}^{\mathbf{z}}_n}[k] = \sum_{i+j=k} \mathbb{I}_{\left\{\widehat{\tilde{p}^{\mathbf{z}}_{n-1}}[i]p^{\mathbf{z}}_n[j]>\epsilon\right\}} \widehat{\tilde{p}^{\mathbf{z}}_{n-1}}[i]p^{\mathbf{z}}_n[j], \text{ for } n>0,
\end{aligned} \tag{5.1.44}$$

where

$$\mathbb{I}_{\{x>\epsilon\}} = \begin{cases} 1 & \text{if } x > \epsilon, \\ 0 & \text{if } x \le \epsilon. \end{cases}$$

The truncation error incurred in approximating $\tilde{\boldsymbol{p}}^{\mathbf{z}}_n$ by $\widehat{\tilde{\boldsymbol{p}}^{\mathbf{z}}_n}$ can be estimated; Theorem 5.2 gives a bound of the total truncation error incurred in the N-fold linear convolution.

Theorem 5.2. *Assume $\tilde{p}^{\mathbf{z}}_n[k]$ is defined by* (5.1.43) *and $\widehat{\tilde{p}^{\mathbf{z}}_n}[k]$ is defined by* (5.1.44)*, and let*

$$\hat{n} = \min\{n \,|\exists i,j \in \{0,\dots,K-1\},\ \tilde{p}^{\mathbf{z}}_{n-1}[i]p^{\mathbf{z}}_n[j] \le \epsilon\}, \tag{5.1.45}$$

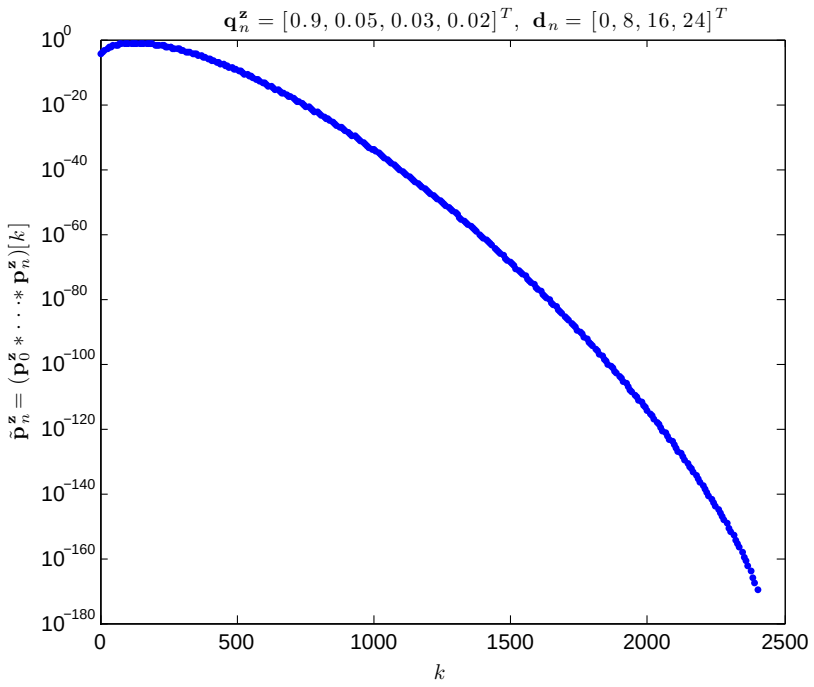
then

$$0 \le \tilde{p}^{\mathbf{z}}_{N-1}[k] - \widehat{\tilde{p}^{\mathbf{z}}_{N-1}}[k] \le (N-\hat{n})C\epsilon.$$

Proof. Since $\tilde{p}^{\mathbf{z}}_n[k] = \widehat{\tilde{p}^{\mathbf{z}}_n}[k]$ for $n < \hat{n}$, it is sufficient to prove

$$0 \le \tilde{p}^{\mathbf{z}}_{\hat{n}+t}[k] - \widehat{\tilde{p}^{\mathbf{z}}_{\hat{n}+t}}[k] \le (t+1)C\epsilon, \tag{5.1.46}$$

Figure 5.7: Small probability mass ($n = 100$)



for all $t \in \mathbb{N}$. We prove (5.1.46) by induction on t. For $t = 0$, given the definition of $\tilde{p}_n^{\mathbf{z}}[k]$ and $\widehat{\tilde{p}_n^{\mathbf{z}}}[k]$ in (5.1.43) and (5.1.44), we have

$$\begin{aligned}
\tilde{p}_{\hat{n}}^{\mathbf{z}}[k] - \widehat{\tilde{p}_{\hat{n}}^{\mathbf{z}}}[k] &= \sum_{i+j=k} \left(\tilde{p}_{\hat{n}-1}^{\mathbf{z}}[i] p_{\hat{n}}^{\mathbf{z}}[j] - \mathbb{I}_{\left\{\widehat{\tilde{p}_{\hat{n}-1}^{\mathbf{z}}}[i] p_{\hat{n}}^{\mathbf{z}}[j] > \epsilon\right\}} \widehat{\tilde{p}_{\hat{n}-1}^{\mathbf{z}}}[i] p_{\hat{n}}^{\mathbf{z}}[j] \right) \\
&= \sum_{i+j=k} \left(\tilde{p}_{\hat{n}-1}^{\mathbf{z}}[i] p_{\hat{n}}^{\mathbf{z}}[j] - \mathbb{I}_{\left\{\tilde{p}_{\hat{n}-1}^{\mathbf{z}}[i] p_{\hat{n}}^{\mathbf{z}}[j] > \epsilon\right\}} \tilde{p}_{\hat{n}-1}^{\mathbf{z}}[i] p_{\hat{n}}^{\mathbf{z}}[j] \right) \\
&= \sum_{i+j=k} \left(1 - \mathbb{I}_{\left\{\tilde{p}_{\hat{n}-1}^{\mathbf{z}}[i] p_{\hat{n}}^{\mathbf{z}}[j] > \epsilon\right\}} \right) \tilde{p}_{\hat{n}-1}^{\mathbf{z}}[i] p_{\hat{n}}^{\mathbf{z}}[j].
\end{aligned}$$

Let $f(x, \epsilon) = \left(1 - \mathbb{I}_{\{x > \epsilon\}}\right) x$, then

$$f(x, \epsilon) = \begin{cases} 0 & \text{if } x > \epsilon, \\ x & \text{if } x \leq \epsilon. \end{cases}$$

Hence, $0 \leq f(x, \epsilon) \leq \epsilon$ and

$$0 \leq \tilde{p}_{\hat{n}}^{\mathbf{z}}[k] - \widehat{\tilde{p}_{\hat{n}}^{\mathbf{z}}}[k] = \sum_{i+j=k} f\left(\tilde{p}_{\hat{n}-1}^{\mathbf{z}}[i] p_{\hat{n}}^{\mathbf{z}}[j], \epsilon\right) \leq \sum_{i+j=k} \epsilon \leq C\epsilon,$$

where the rightmost inequality in the line above follows from the observation that the vector $\boldsymbol{p}_{\hat{n}}^{\mathbf{z}}$ has only

C nonzero elements. For the induction step, assume

$$0 \leq \tilde{p}^{\mathbf{z}}_{\hat{n}+t}[k] - \widehat{\tilde{p}^{\mathbf{z}}_{\hat{n}+t}}[k] \leq (t+1)C\epsilon \tag{5.1.47}$$

for some $t \geq 0$, and consider

$$\begin{aligned}
&\tilde{p}^{\mathbf{z}}_{\hat{n}+(t+1)}[k] - \widehat{\tilde{p}^{\mathbf{z}}_{\hat{n}+(t+1)}}[k] \\
&= \sum_{i+j=k} \left(\tilde{p}^{\mathbf{z}}_{\hat{n}+t}[i] p^{\mathbf{z}}_{\hat{n}+(t+1)}[j] - \mathbb{I}_{\left\{\widehat{\tilde{p}^{\mathbf{z}}_{\hat{n}+t}}[i] p^{\mathbf{z}}_{\hat{n}+(t+1)}[j] > \epsilon\right\}} \widehat{\tilde{p}^{\mathbf{z}}_{\hat{n}+t}}[i] p^{\mathbf{z}}_{\hat{n}+(t+1)}[j] \right).
\end{aligned} \tag{5.1.48}$$

First note that $\tilde{p}^{\mathbf{z}}_{\hat{n}+(t+1)}[k] - \widehat{\tilde{p}^{\mathbf{z}}_{\hat{n}+(t+1)}}[k] \geq 0$ since (5.1.47) implies that each term in the sum on the right side of (5.1.48) is nonnegative. To prove the upper bound on $\tilde{p}^{\mathbf{z}}_{\hat{n}+(t+1)}[k] - \widehat{\tilde{p}^{\mathbf{z}}_{\hat{n}+(t+1)}}[k]$, note that $\tilde{p}^{\mathbf{z}}_{\hat{n}+t}[i] \leq \widehat{\tilde{p}^{\mathbf{z}}_{\hat{n}+t}}[i] + (t+1)C\epsilon$, whence from (5.1.48)

$$\begin{aligned}
\tilde{p}^{\mathbf{z}}_{\hat{n}+(t+1)}[k] - \widehat{\tilde{p}^{\mathbf{z}}_{\hat{n}+(t+1)}}[k] &\leq \sum_{i+j=k} \left(\left(\widehat{\tilde{p}^{\mathbf{z}}_{\hat{n}+t}}[i] + (t+1)C\epsilon \right) p^{\mathbf{z}}_{\hat{n}+(t+1)}[j] \right. \\
&\quad \left. - \mathbb{I}_{\left\{\widehat{\tilde{p}^{\mathbf{z}}_{\hat{n}+t}}[i] p^{\mathbf{z}}_{\hat{n}+(t+1)}[j] > \epsilon\right\}} \widehat{\tilde{p}^{\mathbf{z}}_{\hat{n}+t}}[i] p^{\mathbf{z}}_{\hat{n}+(t+1)}[j] \right) \\
&= \sum_{i+j=k} \left(1 - \mathbb{I}_{\left\{\widehat{\tilde{p}^{\mathbf{z}}_{\hat{n}+t}}[i] p^{\mathbf{z}}_{\hat{n}+(t+1)}[j] > \epsilon\right\}} \right) \widehat{\tilde{p}^{\mathbf{z}}_{\hat{n}+t}}[i] p^{\mathbf{z}}_{\hat{n}+(t+1)}[j] \\
&\quad + (t+1)C\epsilon \sum_{i+j=k} p^{\mathbf{z}}_{\hat{n}+(t+1)}[j] \\
&= \sum_{i+j=k} f\left(\widehat{\tilde{p}^{\mathbf{z}}_{\hat{n}+t}}[i] p^{\mathbf{z}}_{\hat{n}+(t+1)}[j], \epsilon \right) + (t+1)C\epsilon \sum_{i+j=k} p^{\mathbf{z}}_{\hat{n}+(t+1)}[j] \\
&\leq C\epsilon + (t+1)C\epsilon \\
&= ((t+1)+1)C\epsilon.
\end{aligned}$$

Therefore, (5.1.46) is true for all $t \in \mathbb{N}$. Consequently, $0 \leq \tilde{p}^{\mathbf{z}}_{N-1}[k] - \widehat{\tilde{p}^{\mathbf{z}}_{N-1}}[k] \leq (N - \hat{n})C\epsilon$. □

We can apply Theorem 5.2 to obtain an error bound for the truncated sparse convolution method when computing the conditional cumulative probabilities. This result is presented in the following corollary.

Corollary 5.3. *Let $\tilde{p}^{\mathbf{z}}_n[k]$, $\widehat{\tilde{p}^{\mathbf{z}}_n}[k]$ and $\hat{n}$ be defined as in Theorem 5.2. Then*

$$\mathbb{P}\left\{\mathcal{L}^{\mathbf{z}} \leq l_j\right\}_{sconv} - \mathbb{P}\left\{\mathcal{L}^{\mathbf{z}} \leq l_j\right\}_{\widehat{sconv}} \leq j(N - \hat{n})C\epsilon.$$

Proof. From (5.1.11), we have

$$\mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} \leq l_j\right\}_{sconv} = \sum_{k \leq j} \tilde{\boldsymbol{p}}^{\boldsymbol{z}}_{N-1}[k],$$

$$\mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} \leq l_j\right\}_{\widehat{sconv}} = \sum_{k \leq j} \widehat{\tilde{\boldsymbol{p}}^{\boldsymbol{z}}_{N-1}}[k].$$

Therefore, from Theorem 5.2,

$$\begin{aligned}\mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} \leq l_j\right\}_{sconv} - \mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} \leq l_j\right\}_{\widehat{sconv}} &= \sum_{k \leq j} \left(\tilde{\boldsymbol{p}}^{\boldsymbol{z}}_{N-1}[k] - \widehat{\tilde{\boldsymbol{p}}^{\boldsymbol{z}}_{N-1}}[k]\right) \\ &\leq j(N-\hat{n})C\epsilon.\end{aligned}$$

□

In practice, we wish to pre-define a threshold Tol such that

$$\tilde{p}^{\boldsymbol{z}}_{N-1}[k] - \widehat{\tilde{p}^{\boldsymbol{z}}_{N-1}}[k] \leq Tol,$$

or

$$\mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} \leq l_j\right\}_{sconv} - \mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} \leq l_j\right\}_{\widehat{sconv}} \leq Tol,$$

and based on this threshold to determine ϵ in (5.1.44). Theorem 5.2 shows that if we set

$$\epsilon = \frac{Tol}{(N-\hat{n})C}, \tag{5.1.49}$$

then

$$\tilde{p}^{\boldsymbol{z}}_{N-1}[k] - \widehat{\tilde{p}^{\boldsymbol{z}}_{N-1}}[k] \leq (N-\hat{n})C\epsilon = Tol.$$

However, from (5.1.45), we know that $\hat{n}$ is determined during the computation, so ϵ can not be determined before computation by (5.1.49). To solve this problem, notice that

$$\tilde{p}^{\boldsymbol{z}}_{N-1}[k] - \widehat{\tilde{p}^{\boldsymbol{z}}_{N-1}}[k] \leq (N-\hat{n})C\epsilon \leq NC\epsilon.$$

Hence, for the truncated version of sparse convolution, if we can set

$$\epsilon = \frac{Tol}{NC}, \tag{5.1.50}$$

then

$$\tilde{p}^{\boldsymbol{z}}_{N-1}[k] - \widehat{\tilde{p}^{\boldsymbol{z}}_{N-1}}[k] \leq NC\epsilon = Tol.$$

Since N and C are known before computation, ϵ obtained by (5.1.50) can be determined before computation.

Similarly, we can let

$$\epsilon = \frac{Tol}{jNC}, \tag{5.1.51}$$

so that for any pre-defined Tol, we have

$$\mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} \leq l_j\right\}_{sconv} - \mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} \leq l_j\right\}_{\widehat{sconv}} \leq Tol.$$

It is trivial to modify Algorithm 5.1 to incorporate the truncation. The only change required is to add an `if` statement to test the condition $\tilde{p}^{\boldsymbol{z}}_{n-1}[i]p^{\boldsymbol{z}}_n[j] > \epsilon$. If the condition is satisfied, then Line 13 to Line 19 in Algorithm 5.1 are executed. Otherwise, the new algorithm jumps to Line 20 in Algorithm 5.1. The algorithm for the truncated sparse convolution method is listed in Algorithm 5.3. However, Algorithm 5.2 can not be modified to accommodate this feature, since the `if` statement to test the condition $\tilde{p}^{\boldsymbol{z}}_{n-1}[i]p^{\boldsymbol{z}}_n[j] > \epsilon$ makes $\tilde{\boldsymbol{d}}_n$ dependent on $\boldsymbol{z}$. Therefore, $\tilde{\boldsymbol{d}}_n$ may be different for different realizations of the risk factors. Hence, we have to compute $\tilde{\boldsymbol{d}}_n$ U times (once for each $\boldsymbol{z}^{(u)}$) rather than just once.

Algorithm 5.3 Truncated sparse convolution method to compute the conditional loss probabilities

Input: $\boldsymbol{q}_n^{\boldsymbol{z}}$, $\boldsymbol{d}_n$ for $n = 0, \ldots, N-1, \epsilon$;

Output: $\tilde{\boldsymbol{p}} = \tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}}$, $\tilde{\boldsymbol{d}} = \tilde{\boldsymbol{d}}_{N-1}$;

$\boldsymbol{p} \leftarrow -\boldsymbol{\epsilon}_{\mathbf{M}}$;
$\boldsymbol{d} \leftarrow \boldsymbol{d}_0$
for $c = 0 : C-1$ **do**
    $p\,[d[c]] \leftarrow q_0^{\boldsymbol{z}}[c]$;
end for
$\tilde{\boldsymbol{p}} \leftarrow \boldsymbol{p}$;
$\tilde{\boldsymbol{d}} \leftarrow \boldsymbol{d}$;
for $n = 1 : N-1$ **do**
    $\tilde{\boldsymbol{p}} \leftarrow -\boldsymbol{\epsilon}_{\mathbf{M}}$;
    $\tilde{\boldsymbol{d}} \leftarrow$ `NULL`;
    for $j = 0 : C-1$ **do**
        for $i = 0 :$ `length`$(\boldsymbol{d}) - 1$ **do**
            $a \leftarrow p[d[i]] \times q_n[j]$;
            if $a > \epsilon$ **then**
                $k \leftarrow d[i] + d_n[j]$;
                if $\tilde{p}[k] < 0$ **then**
                    $\tilde{d}[\texttt{end} + 1] \leftarrow k$;
                    $\tilde{p}[k] \leftarrow a$;
                else
                    $\tilde{p}[k] \leftarrow \tilde{p}[k] + a$;
                end if
            end if
        end for
    end for
    $\boldsymbol{p} \leftarrow \tilde{\boldsymbol{p}}$;
    $\boldsymbol{d} \leftarrow \tilde{\boldsymbol{d}}$;
end for
return $\tilde{\boldsymbol{p}}$, $\tilde{\boldsymbol{d}}$;

5.2 Sparse FFT Method

Another efficient way to compute the N-fold linear convolution in (5.1.9) is to use the fast Fourier transform (FFT). In this section, we first describe how to apply the full FFT algorithm to compute the N-fold linear convolution. Then we develop a sparse FFT algorithm and a truncated sparse FFT algorithm.

5.2.1 Computation of the Conditional Loss Probability by the Fourier Transform

We begin by introducing the discrete circular convolution and the discrete Fourier transform. The circular convolution of two real vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, both of length K, is a mapping from $\mathbb{R}^K \times \mathbb{R}^K$ to $\mathbb{R}^K$ defined by

$$\boldsymbol{x} \star \boldsymbol{y}[k] \doteq \sum_{j=0}^{K-1} x[j] y\left[(k-j) \bmod K\right] \tag{5.2.1}$$

for $k = 0, \ldots, K-1$. Like the discrete linear convolution, the discrete circular convolution can be represented in matrix form using a circulant matrix $\mathbf{C}_{\boldsymbol{y}}$. For example, suppose that both $\boldsymbol{x}$ and $\boldsymbol{y}$ have four elements, then

$$\begin{aligned} \boldsymbol{x} \star \boldsymbol{y} &= \mathbf{C}_{\boldsymbol{y}} \boldsymbol{x} \\ &= \begin{bmatrix} y_0 & y_3 & y_2 & y_1 \\ y_1 & y_0 & y_3 & y_2 \\ y_2 & y_1 & y_0 & y_3 \\ y_3 & y_2 & y_1 & y_0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}. \end{aligned}$$

The discrete Fourier transform of a complex vector $\boldsymbol{x}$ of length K is a mapping from $\mathbb{C}^K$ to $\mathbb{C}^K$ defined by

$$\mathcal{F}(\boldsymbol{x})[k] = \sum_{j=0}^{K-1} x[j] w^{jk},$$

for $k = 0, \ldots, K-1$, where $w = e^{\frac{-2\pi i}{K}}$ and $i = \sqrt{-1}$. In matrix form, the discrete Fourier transform (DFT) of a K-element vector $\boldsymbol{x}$ can be written as the product of a DFT matrix $\mathbf{F}$ and the vector $\boldsymbol{x}$:

$$\mathcal{F}(\boldsymbol{x}) = \mathbf{F}\boldsymbol{x} = \begin{bmatrix} w^{0\cdot 0} & w^{0\cdot 1} & \cdots & w^{0\cdot(K-1)} \\ w^{1\cdot 0} & w^{1\cdot 1} & \cdots & w^{1\cdot(K-1)} \\ \vdots & \vdots & \ddots & \vdots \\ w^{(K-1)\cdot 0} & w^{(K-1)\cdot 1} & \cdots & w^{(K-1)\cdot(K-1)} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{K-1} \end{bmatrix}. \tag{5.2.2}$$

Similarly, the inverse discrete Fourier transform (iDFT) is defined by

$$\mathcal{F}^{-1}(\boldsymbol{x})[k] = \frac{1}{K}\sum_{j=0}^{K-1} x[j]w^{-jk},$$

for $k = 0, \ldots, K-1$, and the matrix form of the iDFT is

$$\mathcal{F}^{-1}(\boldsymbol{x}) = \mathbf{F}^{-1}\boldsymbol{x} = \frac{1}{K}\begin{bmatrix} w^{-0\cdot 0} & w^{-0\cdot 1} & \cdots & w^{-0\cdot(K-1)} \\ w^{-1\cdot 0} & w^{-1\cdot 1} & \cdots & w^{-1\cdot(K-1)} \\ \vdots & \vdots & \ddots & \vdots \\ w^{-(K-1)\cdot 0} & w^{-(K-1)\cdot 1} & \cdots & w^{-(K-1)\cdot(K-1)} \end{bmatrix}\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{K-1} \end{bmatrix}. \tag{5.2.3}$$

Notice that

$$\left(\mathbf{F}^{-1}\mathbf{F}\right)[m,n] = \frac{1}{K}\sum_{k=0}^{K-1} w^{-m\cdot k}w^{k\cdot n} = \frac{1}{K}\sum_{k=0}^{K-1} w^{(n-m)k}.$$

- If $m \neq n$, $\sum_{k=0}^{K-1} w^{(n-m)k}$ is a geometric series. Hence,

$$\left(\mathbf{F}^{-1}\mathbf{F}\right)[m,n] = \frac{1}{K}\cdot\frac{w^{(n-m)K}-1}{w^{(n-m)}-1}.$$

  Since $w = e^{\frac{-2\pi i}{K}}$ is a primitive Kth root of unity, which implies $w^K = 1$, whence, $w^{(n-m)K} = 1$. However, $w^{(n-m)} \neq 1$. Therefore,

$$\left(\mathbf{F}^{-1}\mathbf{F}\right)[m,n] = 0$$

  for $m \neq n$.

- If $m = n$, then $w^{(n-m)K} = w^0 = 1$. Hence,

$$\left(\mathbf{F}^{-1}\mathbf{F}\right)[m,n] = 1.$$

Therefore, we have $\mathbf{F}^{-1}\mathbf{F} = \mathbf{I}$, where $\mathbf{I}$ is an identity matrix. This implies $\mathcal{F}^{-1}(\mathcal{F}(\boldsymbol{x})) = \boldsymbol{x}$.

The circular convolution theorem [8] links the discrete circular convolution and the DFT:

$$\boldsymbol{x} \star \boldsymbol{y} = \mathcal{F}^{-1}\left(\mathcal{F}(\boldsymbol{x}) \odot \mathcal{F}(\boldsymbol{y})\right),$$

where $\odot$ is the component-wise product. Moreover, the circular convolution theorem can be extended to an N-fold discrete circular convolution:

$$\overset{N-1}{\underset{n=0}{\circledast}} \boldsymbol{x}_n = \mathcal{F}^{-1}\left(\overset{N-1}{\underset{n=0}{\odot}} \mathcal{F}(\boldsymbol{x}_n)\right). \tag{5.2.4}$$

The conditional loss probabilities we need to compute in (5.1.9) are given by an N-fold discrete linear convolution rather than an N-fold discrete circular convolution. However, Theorem 5.4 below gives the relationship between an N-fold discrete linear convolution and the corresponding N-fold discrete circular convolution.

Theorem 5.4. *Let $N \geq 2$ and $\{\boldsymbol{x}_n\}_{n=0}^{N-1}$ be a sequence of vectors, each of length K. Then*

$$\left(\circledast_{n=0}^{N-1} \boldsymbol{x}_n\right)[k] = \sum_{u=0}^{U_k^{N-1}} \left(\circledast_{n=0}^{N-1} \boldsymbol{x}_n\right)[uK+k] \tag{5.2.5}$$

for $k = 0, \ldots, K-1$, where

$$U_k^{N-1} = \max\{u \in \mathbb{N} \,|\, 0 \leq uK + k \leq N(K-1)\}.$$

A proof of this theorem is given in Appendix D. Note that the upper and lower bounds in the sum on the right side of (5.2.5) make indices $uK+k$, for $k = 0, \ldots, K-1$, range from the first index of $\circledast_{n=0}^{N-1} \boldsymbol{x}_n$, 0, to the last index of $\circledast_{n=0}^{N-1} \boldsymbol{x}_n$, $N(K-1)$. Theorem 5.4 shows that the N-fold discrete circular convolution is the "wrapped" version of the N-fold discrete linear convolution, and the N-fold discrete circular convolution at k can be obtained by summing the N-fold discrete linear convolution at the points $uK+k$, which can be "wrapped around" to k. An immediate consequence of this theorem is the following corollary, which shows that, due to the way that we construct the length, K, of the vectors, $\boldsymbol{p}_n^{\boldsymbol{z}}$, the N-fold linear convolution agrees with the N-fold discrete circular convolution when each are used to compute $\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} = l_k\}$.

Corollary 5.5. *Assume the perfect discretization scheme discussed in Section 5.1.1 is applied, where $l_k = k\delta$, $p_n^{\boldsymbol{z}}[k] = \mathbb{P}\{\mathcal{L}_n^{\boldsymbol{z}} = l_k\}$, $k = 0, \ldots, K-1$, and $l_{K-1} = \sum_{n=0}^{N-1} L_n^0$, then*

$$\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} = l_k\} = \left(\circledast_{n=0}^{N-1} \boldsymbol{p}_n^{\boldsymbol{z}}\right)[k] = \left(\circledast_{n=0}^{N-1} \boldsymbol{p}_n^{\boldsymbol{z}}\right)[k],$$

for $k = 0, \ldots, K-1$.

Proof. We have shown $\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} = l_k\} = \left(\circledast_{n=0}^{N-1} \boldsymbol{p}_n^{\boldsymbol{z}}\right)[k]$ in Subsection 5.1.1. By Theorem 5.4,

$$\left(\circledast_{n=0}^{N-1} \boldsymbol{p}_n^{\boldsymbol{z}}\right)[k] = \left(\circledast_{n=0}^{N-1} \boldsymbol{p}_n^{\boldsymbol{z}}\right)[k] + \sum_{u=1}^{U_k^{N-1}} \left(\circledast_{n=0}^{N-1} \boldsymbol{p}_n^{\boldsymbol{z}}\right)[uK+k], \tag{5.2.6}$$

for $k = 0, \ldots, K-1$. Notice that $l_{K-1} = \sum_{n=0}^{N-1} L_n^0$, is the maximum portfolio loss. Thus, $\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} = l_k\} = 0$

for $k \geq K$. Hence,

$$\mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} = l_{uK+k}\right\} = \left(\overset{N-1}{\underset{n=0}{\circledast}} \boldsymbol{p}_n^{\boldsymbol{z}}\right)[uK+k] = 0$$

for $u \geq 1$ and $k = 0, \ldots, K-1$. Therefore, equation (5.2.6) reduces to

$$\left(\overset{N-1}{\underset{n=0}{\circledast}} \boldsymbol{p}_n^{\boldsymbol{z}}\right)[k] = \left(\overset{N-1}{\underset{n=0}{\circledast}} \boldsymbol{p}_n^{\boldsymbol{z}}\right)[k],$$

which completes the proof. □

According to Corollary 5.5, the conditional loss probabilities can be computed by an N-fold circular convolution. So, by (5.2.4), we have

$$\mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} = l_k\right\} = \left(\overset{N-1}{\underset{n=0}{\circledast}} \boldsymbol{p}_n^{\boldsymbol{z}}\right)[k] = \left(\mathcal{F}^{-1}\left(\overset{N-1}{\underset{n=0}{\odot}} \mathcal{F}\left(\boldsymbol{p}_n^{\boldsymbol{z}}\right)\right)\right)[k] \tag{5.2.7}$$

for $k = 0, \ldots, K-1$.

If we implement the DFT as a matrix-vector multiply, as shown in (5.2.2), we need K^2 complex multiplications (or $4K^2$ real multiplications and $2K^2$ real additions) and $K(K-1)$ complex additions (or $2K(K-1)$ real additions) to compute one DFT. Hence, if assuming we have the Fourier matrix $\mathbf{F}$, then the FLOPs required to compute the loss probability by (5.2.7) are: □

- NK^2 complex multiplications and $NK(K-1)$ complex additions to compute N DFTs, which is equivalent to $4NK^2$ real multiplications and $2NK^2 + 2NK(K-1)$ real additions;
- $(N-1)K$ complex multiplications to compute $N-1$ component-wise products, which is equivalent to $4(N-1)K$ real multiplications and $2(N-1)K$ real additions;
- $K^2 + K$ complex multiplications and $K(K-1)$ complex additions to compute one iDFT, which is equivalent to $4(K^2+K)$ real multiplications and $2(K^2+K) + 2K(K-1)$ real additions.

Therefore, we need $8(N+1)K^2 + 2(2N-1)K$ FLOPs in total, which is even more than the number of FLOPs required by the full convolution method (as shown in (5.1.15)).

5.2.2 Full FFT Method

Rather than computing (5.2.7) by matrix-vector multiplies, we can accelerate the computation by using the fast Fourier transform (FFT) instead. This brings down the cost of computing each Fourier transform from $O(K^2)$ to $O(K \log_2 K)$. There are many different FFT algorithms. We briefly review

the commonly-used Cooley-Tukey radix-2 FFT algorithm to compute $\mathcal{F}(\boldsymbol{x})$. Readers can find a more complete discussion in [57].

The Cooley-Tukey radix-2 FFT algorithm assumes $K = 2^t$ for some integer t. This leads to the following factorization of the DFT matrix $\mathbf{F}$:

$$\mathbf{F} = \mathbf{A}_t \cdots \mathbf{A}_1 \mathbf{P}_K^T,$$

where

$$\begin{aligned}
\mathbf{A}_q &= \begin{bmatrix} \mathbf{B}_{L_q} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_{L_q} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{B}_{L_q} \end{bmatrix}, && L_q = 2^q, \\
\mathbf{B}_{L_q} &= \begin{bmatrix} \mathbf{I}_{L_q^*} & \mathbf{\Omega}_{L_q^*} \\ \mathbf{I}_{L_q^*} & -\mathbf{\Omega}_{L_q^*} \end{bmatrix}, && L_q^* = L_q/2, \\
\mathbf{\Omega}_{L_q^*} &= \operatorname{diag}\left(w_{L_q}^0, w_{L_q}^1, \ldots, w_{L_q}^{L_q^*-1}\right), \ w_{L_q} = e^{-\frac{2\pi i}{L_q}}, \\
\mathbf{I}_{L_q^*} &= \operatorname{diag}(1, 1, \ldots, 1).
\end{aligned} \tag{5.2.8}$$

Note $\mathbf{A}_q$ is a $K \times K$ matrix, where $K = 2^t$, consisting of 2^{t-q} diagonal blocks $\mathbf{B}_{L_q}$, where $\mathbf{B}_{L_q}$ is an $L_q \times L_q$ matrix and $L_q = 2^q$. $\mathbf{\Omega}_{L_q^*}$ is a diagonal $L_q^* \times L_q^*$ matrix and $\mathbf{I}_{L_q^*}$ is the $L_q^* \times L_q^*$ identity matrix where $L_q^* = L_q/2 = 2^{q-1}$. $\mathbf{P}_K^T$ is a bit-reversal permutation matrix which permutes a vector $\boldsymbol{x}$ by reversing the order of the bits in the binary expansion of its indices:

$$\hat{\boldsymbol{x}}_0 \doteq \mathbf{P}_K^T \boldsymbol{x}, \quad \hat{x}_0\left[(b_{t-1} \cdots b_1 b_0)_2\right] = x\left[(b_0 \cdots b_{t-2} b_{t-1})_2\right]. \tag{5.2.9}$$

Based on this factorization, the Fourier transform, $\mathcal{F}(\boldsymbol{x})$, can be computed as follows.

S1. Permute $\boldsymbol{x}$: $\hat{\boldsymbol{x}}_0 = \mathbf{P}_K^T \boldsymbol{x}$, which is also called "bit reversal";

S2. Compute the weight vectors $\boldsymbol{w}_{L_q^*} = \left[w_{L_q}^0, w_{L_q}^1, \ldots, w_{L_q}^{L_q^*-1}\right]^T$, $L_q^* = 2^0, 2^1, \ldots, 2^{t-1}$;

S3. Compute the matrix-vector products $\hat{\boldsymbol{x}}_q = \mathbf{A}_q \hat{\boldsymbol{x}}_{q-1}$ for $q = 1, \ldots, t$. Due to the special structure of $\mathbf{A}_q$, the matrix-vector product $\mathbf{A}_q \hat{\boldsymbol{x}}_{q-1}$ can be implemented efficiently as a series of computations called "butterfly updates", discussed later in this section.

5.2.2.1 Bit Reversal

Let $k = (b_0 \cdots b_{t-2} b_{t-1})_2$ and denote $r_t(k) \doteq (b_{t-1} \cdots b_1 b_0)_2$. Then the permutation (5.2.9) can be implemented efficiently as

$$\hat{x}_0[r_t(k)] = x[k]. \tag{5.2.10}$$

Thus, we need to map k to $r_t(k)$. Notice that

$$\begin{aligned} k &= b_0 + b_1 \cdot 2 + \cdots + b_{t-1} \cdot 2^{t-1}, \\ r_t(k) &= b_0 \cdot 2^{t-1} + b_1 \cdot 2^{t-2} + \cdots + b_{t-1}. \end{aligned}$$

If we let

$$r_t^q(k) \doteq b_0 \cdot 2^{q-1} + b_1 \cdot 2^{q-2} + \cdots + b_{q-1},$$

then $r_t(k) = r_t^t(k)$ and

$$\begin{aligned} r_t^1(k) &= b_{0,} \\ r_t^2(k) &= 2b_0 + b_1 = 2r_t^1(k) + b_1, \\ r_t^3(k) &= 2(2b_0 + b_1) + b_2 = 2r_t^2(k) + b_2, \\ r_t^4(k) &= 2(2(2b_0 + b_1) + b_2) + b_3 = 2r_t^3(k) + b_3, \\ &\vdots \\ r_t^t(k) &= 2r_t^{t-1}(k) + b_{t-1}. \end{aligned}$$

Therefore, given k, once b_q, $q = 0, \ldots, t-1$, are determined, we can use the iteration above to compute $r_t(k)$. To determine b_q, b_q, $q = 0, \ldots, t-1$, denote

$$h_t^q(k) \doteq b_q + b_{q+1} \cdot 2 + \cdots + b_{t-1} \cdot 2^{t-1-q},$$

for $q = 0, \ldots, t-1$. Then $k = h_t^0(k)$ and

$$b_q = h_t^q(k) - 2h_t^{q+1}(k),\ q = 0, \ldots, t-1.$$

Also, notice that

$$\left\lfloor \frac{h_t^q(k)}{2} \right\rfloor = h_t^{q+1}(k).$$

Hence, we can use the following iteration to determine b_q, $q = 0, \ldots t-1$:

$$\begin{aligned}
h_t^0(k) &= k, \\
h_t^1(k) &= \left\lfloor \frac{h_t^0(k)}{2} \right\rfloor,\ b_0 = h_t^0(k) - 2h_t^1(k), \\
h_t^2(k) &= \left\lfloor \frac{h_t^1(k)}{2} \right\rfloor,\ b_1 = h_t^1(k) - 2h_t^2(k), \\
&\vdots \\
h_t^t(k) &= \left\lfloor \frac{h_t^{t-1}(k)}{2} \right\rfloor = 0,\ b_{t-1} = h_t^{t-1}(k) - 2h_t^t(k).
\end{aligned}$$

Moreover, since $r_t\left(r_t(k)\right) = k$,

$$\hat{x}_0[k] = x[r_t(k)]. \tag{5.2.11}$$

From (5.2.10) and (5.2.11) we see that we only need to swap $x[k]$ and $x[r_t(k)]$ for $k = 0, \ldots, K-1$ to obtain $\hat{\boldsymbol{x}}_0$. Based on the analysis above, an algorithm to perform the bit reversal permutation, with number of integer operations (INOPs) at each line, is presented in Algorithm 5.4 . It is clear that this algorithm requires $5K \log_2 K$ INOPs to compute $\hat{\boldsymbol{x}}_0 = \mathbf{P}_K^T \boldsymbol{x}$.[6]

Algorithm 5.4 Bit reversal to compute $\hat{\boldsymbol{x}}_0 = \mathbf{P}_K^T \boldsymbol{x}$

Input: $\boldsymbol{x}$, K;
Output: $\boldsymbol{x} = \hat{\boldsymbol{x}}_0$;

```
1:  for k = 0 : K − 1 do
2:      r ← 0;   h ← k;
3:      for q = 0 : t − 1 do
4:          h* ← floor(h/2);                          # Klog2(K) INOPs
5:          r ← 2r + (h − 2h*);                       # 4Klog2(K) INOPs
6:          h ← h*;
7:      end for
8:      if r > k then
9:          x[r] ↔ x[k];
10:     end if
11: end for
12: return x;
```

5.2.2.2 Computation of Weights

Since

$$w_{L_q^*}[j] = e^{-\frac{2\pi j i}{L_q}} = e^{-\frac{2\pi (2j) i}{2L_q}} = w_{L_q}[2j],$$

[6] We assume that $\texttt{floor}(h/2)$ can be computed using the equivalent of one integer arithmetic operation.

$\boldsymbol{w}_{L_q^*} = \boldsymbol{w}_{L_q}[0:2:L_q-1]$, which indicates that it is sufficient to compute $\boldsymbol{w}_{K/2}$ only to obtain all $\boldsymbol{w}_{L_q^*}$, $L_q^* = 2^0, 2^1, \ldots, 2^{t-1}$. However, as noted in [57], a straightforward implementation of this method leads to a power-of-two stride access to $\boldsymbol{w}_{K/2}$. That is, during the qth update $\hat{\boldsymbol{x}}_q = \mathbf{A}_q\hat{\boldsymbol{x}}_{q-1}$, we need to reference consecutively elements of $\boldsymbol{w}_{K/2}$ separated by $r = \frac{K/2}{L_q^*} = 2^{t-q}$, since

$$w_{L_q^*}[j] = w_{K/2}[rj].$$

This may result in consecutive accesses to noncontiguous memory blocks, which may severely degrade the performance of the method, especially for small L_q^*, for which the stride r is very large. One way to avoid this deficiency is to compute the following long weight vector with $K-1$ components,[7]

$$\boldsymbol{w}_{long} = \begin{bmatrix} \boldsymbol{w}_{2^0} \\ \boldsymbol{w}_{2^1} \\ \vdots \\ \boldsymbol{w}_{2^{t-1}} \end{bmatrix}.$$

Therefore, at the qth update, the weight vector is

$$\boldsymbol{w}_{L_q^*} = \boldsymbol{w}_{long}\left[L_q^* - 1 : L_q - 1\right],$$

with elements

$$w_{L_q^*}[j] = w_{long}[L_q^* + j - 1] \tag{5.2.12}$$

accessed from contiguous memory.

There are many methods to compute the elements of $\boldsymbol{w}_{L_q^*}$. We apply the direct computation

$$w_{L_q^*}[j] = \cos\left(-\frac{2\pi j}{L_q}\right) + i\sin\left(-\frac{2\pi j}{L_q}\right).$$

Chu [10] has shown that this direct method is stable and gives the most accurate result compared with other methods. An algorithm based on this direct method to compute the long weight vector, with number of INOPs and trigonometric operations (TROPs) at each line is presented in Algorithm 5.5. In total we need $2(K-1)$ TROPs, $K + 2\log_2 K - 1$ INOPs and $K + 2\log_2 K - 1$ FLOPs. [8]

[7] Notice that, for $q = 0, \ldots, t-1$, each vector $\boldsymbol{w}_{2^q}$ has 2^q components .

[8] We assume that 2^q can be computed using the equivalent of one integer arithmetic operation.

Algorithm 5.5 Computation of the long weight vector $\boldsymbol{w}_{long}$

Input: t;

Output: $\boldsymbol{w}_{long}$;

1: **for** $q = 1 : t$ **do**
2: $L_q \leftarrow 2^q$; $L_q^* \leftarrow L_q/2$; `# 2log2(K) INOPs`
3: $\theta_0 \leftarrow 2\pi/L_q$; `# 2log2(K) FLOPs`
4: **for** $j = 0 : L_q^* - 1$ **do**
5: $\theta \leftarrow j\theta_0$; `# K-1 FLOPs`
6: $w_{L_q^*}[j] = \cos(\theta) - i\sin(\theta)$; `# 2(K-1) TROPs, K-1 INOPs`
7: **end for**
8: **end for**
9: **return** $\boldsymbol{w}_{long}$;

5.2.2.3 Butterfly Update

Given the form of $\mathbf{A}_q$ in (5.2.8), it is helpful to write the matrix-vector products $\hat{\boldsymbol{x}}_q = \mathbf{A}_q\hat{\boldsymbol{x}}_{q-1}$ as

$$\begin{bmatrix} \hat{\boldsymbol{x}}_q^{k,H} \\ \hat{\boldsymbol{x}}_q^{k,T} \end{bmatrix} = \begin{bmatrix} \mathbf{I}_{L_q^*} & \boldsymbol{\Omega}_{L_q^*} \\ \mathbf{I}_{L_q^*} & -\boldsymbol{\Omega}_{L_q^*} \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{x}}_{q-1}^{k,H} \\ \hat{\boldsymbol{x}}_{q-1}^{k,T} \end{bmatrix},$$

where

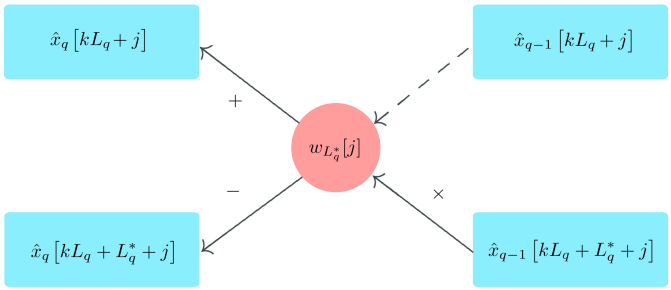
$$\hat{\boldsymbol{x}}_q^{k,H} = \hat{\boldsymbol{x}}_q\left[kL_q : kL_q + L_q^* - 1\right],$$
$$\hat{\boldsymbol{x}}_q^{k,T} = \hat{\boldsymbol{x}}_q\left[kL_q + L_q^* : kL_q + L_q - 1\right],$$

for $k = 0, \ldots, K/L_q - 1$. Therefore, we can compute $\hat{\boldsymbol{x}}_q$ by

$$\hat{x}_q\left[kL_q + j\right] = \hat{x}_{q-1}\left[kL_q + j\right] + w_{L_q^*}[j]\hat{x}_{q-1}\left[kL_q + L_q^* + j\right], \tag{5.2.13}$$
$$\hat{x}_q\left[kL_q + L_q^* + j\right] = \hat{x}_{q-1}\left[kL_q + j\right] - w_{L_q^*}[j]\hat{x}_{q-1}\left[kL_q + L_q^* + j\right], \tag{5.2.14}$$

for $k = 0, \ldots, K/L_q - 1$, $j = 0, \ldots, L_q^* - 1$, implying the computation of $\hat{\boldsymbol{x}}_q$ reduces to a series of butterfly updates shown in Figure 5.8, where a solid line indicates that both data and FLOPs are required, and a dashed line indicates that data are required, but no FLOPs are needed.

Figure 5.8: Butterfly update



An algorithm based on the butterfly update to compute $\hat{\boldsymbol{x}}_q = \mathbf{A}_q\hat{\boldsymbol{x}}_{q-1}$, with number of INOPs and FLOPs indicated at each line, is presented in Algorithm 5.6. The weights, $w_{L_q^*}[j]$, are accessed from the long weight vector $\boldsymbol{w}_{long}$ by (5.2.12). In total, $3K/L_q + 3K/2 + 3$ INOPs and $5K$ FLOPs are needed to compute $\hat{\boldsymbol{x}}_q = \mathbf{A}_q\hat{\boldsymbol{x}}_{q-1}$ for each q.

Algorithm 5.6 Butterfly update to compute $\hat{\boldsymbol{x}}_q = \mathbf{A}_q\hat{\boldsymbol{x}}_{q-1}$

Input: $\boldsymbol{x}$, K, q, $\boldsymbol{w}_{long}$;
Output: $\boldsymbol{x}$;

$L_q = 2^q$; $L_q^* \leftarrow L_q/2$; $r \leftarrow K/L_q$; `# 3 INOPs`
for $k = 0 : r-1$ **do**
    $i_{x0} \leftarrow kL_q$; $i_{x1} \leftarrow i_{x0} + L_q^*$; $i_w \leftarrow L_q^* - 1$; `# 3K/Lq INOPs`
    for $j = 0 : L_q^* - 1$ **do**
        $y_0 \leftarrow x[i_{x0}]$;
        $\tau \leftarrow x[i_{x1}]w_{long}[i_w]$; `# 6K/2 FLOPs`
        $x[i_{x1}] \leftarrow y_0 - \tau$; `# 2K/2 FLOPs`
        $x[i_{x0}] \leftarrow y_0 + \tau$; `# 2K/2 FLOPs`
        $i_{x0} \leftarrow i_{x0} + 1$; $i_{x1} \leftarrow i_{x1} + 1$; `# 2K/2 INOPs`
        $i_w \leftarrow i_w + 1$; `# K/2 INOPs`
    end for
end for
return $\boldsymbol{x}$;

Combining Algorithm 5.4, 5.5 and 5.6, we present in Algorithm 5.7 a complete version of the Cooley-Tukey radix-2 FFT algorithm to compute $\mathcal{F}(\boldsymbol{x})$.

Algorithm 5.7 Cooley-Tukey radix-2 FFT to compute $\mathcal{F}(\boldsymbol{x})$

Input: $\boldsymbol{x}$, K, t;

Output: $\boldsymbol{x}$;

1: $\boldsymbol{x} \leftarrow \mathbf{P}_K^T \boldsymbol{x}$ by Algorithm 5.4;
2: Compute $\boldsymbol{w}_{long}$ by Algorithm 5.5;
3: **for** $q = 1 : t$ **do**
4: $\quad \boldsymbol{x} \leftarrow \mathbf{A}_q \boldsymbol{x}$ by Algorithm 5.6;
5: **end for**
6: **return** $\boldsymbol{x}$;

It is clear that, in Algorithm 5.7, the number of TROPs is $2(K-1)$, the number of INOPs is

$$\begin{aligned}
&5K\log_2 K + (K + 2\log_2 K - 1) + \sum_{q=1}^{t}\left(3\frac{K}{L_q} + \frac{3}{2}K + 3\right) \\
&= 5K\log_2 K + (K + 2\log_2 K - 1) + \left(3\sum_{q=1}^{t} 2^{t-q} + \frac{3}{2}K\log_2 K + 3\log_2 K\right) \\
&= \frac{13}{2}K\log_2 K + 4K + 5\log_2 K - 4.
\end{aligned}$$

and the number of FLOPs is

$$(K + 2\log_2 K - 1) + \sum_{q=1}^{t} 5K = 5K\log_2 K + K + 2\log_2 K - 1.$$

5.2.2.4 Computation of Loss Probability Based on the Full FFT

We can apply Algorithm 5.7 to compute $\mathcal{F}(\boldsymbol{p}_n^{\boldsymbol{z}})$ in (5.2.7) for $n = 0, \ldots, N-1$. Since the weight vector depends only on the length of the vector $\boldsymbol{p}_n^{\boldsymbol{z}}$, and all $\boldsymbol{p}_n^{\boldsymbol{z}}$ have the same length K, we need to compute the weight vector once only. The inverse Fourier transform can be computed similarly, except that we need to negate the imaginary part of the weight vector and scale the result by $1/K$. Algorithm 5.8 shows how to apply the FFT to compute the loss probability.

Algorithm 5.8 Full FFT method to compute the loss probability

Input: $\boldsymbol{p}_n^{\boldsymbol{z}}$, K, t;

Output: $\tilde{\boldsymbol{p}} = \tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}}$;

Compute $\boldsymbol{w}_{long}$ by Algorithm 5.5;
$\tilde{\boldsymbol{p}} \leftarrow [1, 1, \ldots, 1]^T$;

for $n = 0 : N-1$ **do**
    $\boldsymbol{p}_n^{\boldsymbol{z}} \leftarrow \mathbf{P}_K^T \boldsymbol{p}_n^{\boldsymbol{z}}$ by Algorithm 5.4;
    for $q = 1 : t$ **do**
        $\boldsymbol{p}_n^{\boldsymbol{z}} \leftarrow \mathbf{A}_q \boldsymbol{p}_n^{\boldsymbol{z}}$ by Algorithm 5.6;
    end for
    $\tilde{\boldsymbol{p}} \leftarrow \tilde{\boldsymbol{p}} \odot \boldsymbol{p}_n^{\boldsymbol{z}}$;
end for

$\boldsymbol{w}_{long} \leftarrow \overline{\boldsymbol{w}_{long}}$;
$\tilde{\boldsymbol{p}} \leftarrow \mathbf{P}_K^T \tilde{\boldsymbol{p}}$ by Algorithm 5.4;
for $q = 1 : t$ **do**
    $\tilde{\boldsymbol{p}} \leftarrow \overline{\mathbf{A}_q} \tilde{\boldsymbol{p}}$ by Algorithm 5.6; [9]
end for
$\tilde{\boldsymbol{p}} \leftarrow \tilde{\boldsymbol{p}}/K$;
return $\tilde{\boldsymbol{p}}$;

[9] The matrix $\overline{A_q}$ is the conjugate matrix of the matrix A_q, which is defined by

$$\overline{\mathbf{A}_q} = \begin{bmatrix} \overline{\mathbf{B}_{L_q}} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \overline{\mathbf{B}_{\mathbf{L_q}}} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \overline{\mathbf{B}_{\mathbf{L_q}}} \end{bmatrix}, \quad \overline{\mathbf{B}_{\mathbf{L_q}}} = \begin{bmatrix} \mathbf{I}_{L_q^*} & \overline{\mathbf{\Omega}_{L_q^*}} \\ \mathbf{I}_{L_q^*} & -\overline{\mathbf{\Omega}_{L_q^*}} \end{bmatrix},$$

$\overline{\mathbf{\Omega}_{L_q^*}} = \operatorname{diag}\left(\overline{w}_{L_q}^0, \overline{w}_{L_q}^1, \ldots, \overline{w}_{L_q}^{L_q^*-1}\right)$, $\mathbf{I}_{L_q^*} = \operatorname{diag}(1, 1, \ldots, 1)$, and $\overline{w}_{L_q} = e^{\frac{2\pi i}{L_q}}$.

5.2.2.5 Complexity of the Full FFT Method

The computational cost of Algorithm 5.8 is listed in Table 5.1:

Table 5.1: Computational cost of Algorithm 5.8

Line	Number of TROPs	Number of INOPs	Number of FLOPs
1	$2(K-1)$	$K+2\log_2 K-1$	$K+2\log_2 K-1$
5	0	$5NK\log_2 K$	0
7	0	$N\sum_{q=1}^{t}\left(3\frac{K}{L_q}+\frac{3}{2}K+3\right)$	$5NK\log_2 K$
9	0	0	$6(N-1)K$
12	0	0	$K-1$
13	0	$5K\log_2 K$	0
15	0	$\sum_{q=1}^{t}\left(3\frac{K}{L_q}+\frac{3}{2}K+3\right)$	$5K\log_2 K$
17	0	0	K

Therefore, the total computational cost is:

- $\Psi_{fft}^{TROP}=2(K-1)$ TROPs;
- $\Psi_{fft}^{INOP}=\frac{13}{2}(N+1)K\log_2 K+(3N+4)K+(3N+5)\log_2 K-(3N+4)$ INOPs;
- $\Psi_{fft}^{FLOP}=5(N+1)K\log_2 K+2(3N-1)K+2\log_2 K-2$ FLOPs,

which is $\Omega(NK\log_2 K)$.

Denote the computational cost of one TROP (INOP, FLOP) by Γ_{TROP} (Γ_{INOP}, Γ_{FLOP}), and assume that on average,

$$\Gamma_{TROP}=\beta\Gamma_{FLOP},\ \Gamma_{INOP}\approx\Gamma_{FLOP},$$

with $\beta\sim 10$, and that all $\overline{\boldsymbol{p}}_n^{\boldsymbol{z}}$ have the same length $\bar{K}=\frac{K-1}{N}+1$, then by (5.1.36), the speedup of the full FFT method over the full convolution method is

$$
\begin{aligned}
\frac{\Psi_{conv}}{\Psi_{fft}} &= \frac{\left(1-\frac{1}{N}\right)K^2+\left(\frac{N-3}{2}+\frac{1}{N}\right)K+\frac{N-1}{2}}{\beta\Psi_{fft}^{TROP}+\Psi_{fft}^{INOP}+\Psi_{fft}^{FLOP}} \\
&= \frac{\left(1-\frac{1}{N}\right)K^2+\left(\frac{N-3}{2}+\frac{1}{N}\right)K+\frac{N-1}{2}}{\frac{23}{2}(N+1)K\log_2 K+(9N+2\beta+2)K+(3N+7)\log_2 K-(3N+2\beta+6)} \\
&= \frac{K^2\left(\left(1-\frac{1}{N}\right)+\left(\frac{N-3}{2}+\frac{1}{N}\right)\frac{1}{K}+\left(\frac{N-1}{2}\right)\frac{1}{K^2}\right)}{NK\log_2 K\left(\frac{23}{2}(1+\frac{1}{N})+\left(9+\frac{2\beta+2}{N}\right)\frac{1}{\log_2 K}+\left(3+\frac{7}{N}\right)\frac{1}{K}-\left(3+\frac{2\beta+6}{N}\right)\frac{1}{K\log_2 K}\right)} \\
&\overset{K\gg N}{\approx} \frac{K}{\frac{23}{2}N\log_2 K} \\
&= \Omega\left(K/\left(N\log_2 K\right)\right) \qquad (5.2.15)
\end{aligned}
$$

Therefore, if $K \gg N$, then the full FFT method outperforms the full convolution method in terms of computational cost. The speedup decreases as the number of obligors, N, increases, but with a slow rate, since $\bar{K} = \frac{K-1}{N} + 1$, then $K = N\bar{K} - N + 1$, which indicates

$$\Omega\left(\frac{K}{N\log_2 K}\right) = \Omega\left(\frac{\bar{K}}{\log_2 \bar{K} + \log_2 N}\right).$$

5.2.3 Sparse FFT Method

As mentioned in Subsection 5.1.2, in the computation of conditional loss probabilities, the vectors $\boldsymbol{p}_n^{\boldsymbol{z}}$ are very sparse. Therefore, there is some potential to improve the full FFT algorithm described above by utilizing the sparsity. Recall the three steps involved in the full FFT algorithm. The improvements mainly come from the bit reversal and from the butterfly updates. We can also improve the performance of the weights computation, but unlike the bit reversal and the butterfly updates, which must be computed once for each obligor, the weight vector must be computed once only for all obligors. Therefore, the speedup gained from improving the weight vector computation is not significant.

5.2.3.1 Sparse Bit Reversal

As before, we let $\boldsymbol{d} = [d[0], \ldots, d[C-1]]^T$ be the vector of indices of nonzero elements of the vector $\boldsymbol{x}$. We need to compute the index bit reversal for the nonzero elements of $\boldsymbol{x}$ only. Therefore, rather than performing swaps of all elements of $\boldsymbol{x}$, as we did in the full bit reversal algorithm, we assign $x[d[k]]$ to $\hat{x}_0[r_t(d[k])]$ for $k = 0, \ldots, C-1$. The computation of $r_t(d[k])$ is the same as that described earlier for the full bit reversal. In addition, we need only to assign the computed indices to a new vector $\hat{\boldsymbol{d}}$ by $\hat{d}[k] = r_t(d[k])$. Our sparse bit reversal algorithm is presented in Algorithm 5.9. Compared with the full

bit reversal algorithm in Algorithm 5.4, the computational cost is reduced from $5K\log_2 K$ to $5C\log_2 K$.

Algorithm 5.9 Sparse bit reversal to compute $\hat{\boldsymbol{x}}_0 = \mathbf{P}_K^T \boldsymbol{x}$

Input: $\boldsymbol{x}$, $\boldsymbol{d}$, C;
Output: $\hat{\boldsymbol{x}}$, $\hat{\boldsymbol{d}}$;

$\hat{\boldsymbol{x}} \leftarrow \mathbf{0}$;
for $k = 0 : C-1$ **do**
    $r \leftarrow 0$; $h \leftarrow d[k]$;
    for $q = 0 : t-1$ **do**
        $h^* \leftarrow$ `floor`$(h/2)$; `# Clog2(K) INOPs`
        $r \leftarrow 2r + (h - 2h^*)$; `# 4Clog2(K) INOPs`
        $h \leftarrow h^*$;
    end for
    $\hat{x}[r] \leftarrow x[d[k]]$;
    $\hat{d}[k] \leftarrow r$;
end for
return $\hat{\boldsymbol{x}}$, $\hat{\boldsymbol{d}}$;

5.2.3.2 Sparse Butterfly Update

As described in the previous subsection, for each $q = 1, \ldots, t$, the full FFT algorithm computes the matrix-vector multiplication $\mathbf{A}_q \hat{\boldsymbol{x}}_{q-1}$ by performing $K/2$ butterfly updates shown in (5.2.13) and (5.2.14). Due to the sparsity of $\hat{\boldsymbol{x}}_{q-1}$, one or both of $\hat{x}_{q-1}\left[kL_q + j\right]$ and $\hat{x}_{q-1}\left[kL_q + L_q^* + j\right]$ may be zero.

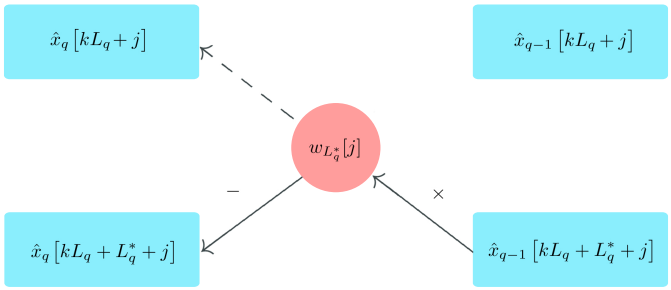
- If both of them are zero, then no update is needed.
- If only $\hat{x}_{q-1}\left[kL_q + j\right]$ is zero, then the full butterfly update (FBU) reduces to the following lower butterfly update (LBU)

$$\hat{x}_q\left[kL_q + j\right] = w_{L_q^*}[j]\hat{x}_{q-1}\left[kL_q + L_q^* + j\right], \tag{5.2.16}$$

$$\hat{x}_q\left[kL_q + L_q^* + j\right] = -w_{L_q^*}[j]\hat{x}_{q-1}\left[kL_q + L_q^* + j\right], \tag{5.2.17}$$

  which is illustrated in Figure 5.9. Comparing with the FBU, which needs one complex multiplication and two complex additions (10 FLOPs in total), one LBU consumes only one complex multiplication and one complex addition for the sign change in (5.2.17) (8 FLOPs in total).

Figure 5.9: Lower butterfly update



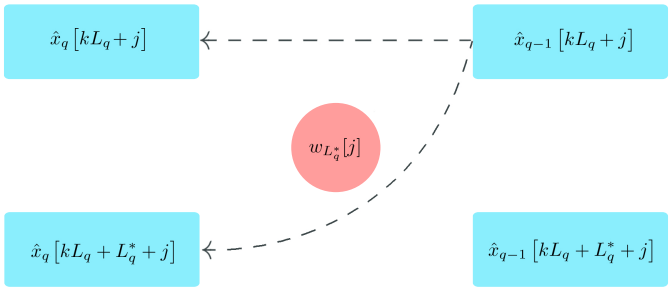
- If only $\hat{x}_{q-1}\left[kL_q + L_q^* + j\right]$ is zero, the FBU reduces to the following upper butterfly update (UBU)

$$\hat{x}_q[kL_q + j] = \hat{x}_{q-1}[kL_q + j], \tag{5.2.18}$$

$$\hat{x}_q\left[kL_q + L_q^* + j\right] = \hat{x}_{q-1}[kL_q + j], \tag{5.2.19}$$

which does not require any FLOPs. The UBU is depicted in Figure 5.10.

Figure 5.10: Upper butterfly update



- If neither $\hat{x}_{q-1}[kL_q + j]$ nor $\hat{x}_{q-1}\left[kL_q + L_q^* + j\right]$ are zero, then the FBU remains unchanged.

5.2.3.3 Sparse FFT Method

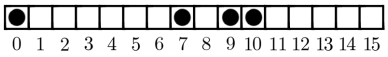
As noted above, there are only C nonzero elements in the initial vector $\boldsymbol{x}$. Since bit reversal only permutes the vector, the vector $\hat{\boldsymbol{x}}_0$ has only C nonzero elements as well. In addition, from (5.2.13) and (5.2.14), it is clear that an FBU does not generate any new nonzero elements.[10] However, LBUs and UBUs do produce new nonzero elements. Therefore, the number of nonzero elements of $\hat{\boldsymbol{x}}_q$ may increase with q. To apply the sparse butterfly updates in $\mathbf{A}_q\hat{\boldsymbol{x}}_{q-1}$ for each q, we need to develop an appropriate

[10] In fact, after an FBU, it is possible that $\hat{x}_q[kL_q + j]$ or $\hat{x}_q\left[kL_q + L_q^* + j\right]$ could be zero even though $\hat{x}_{q-1}[kL_q + j]$ and $\hat{x}_{q-1}\left[kL_q + L_q^* + j\right]$ are nonzero, but this happens so rarely that it is not worth spending the computational effort to check for it. Instead, from now on, once $\hat{x}_{q^*}[j]$ is nonzero for some q^*, we assume $\hat{x}_q[j]$ is nonzero for all $q > q^*$. The same argument applies to LBUs and UBUs as well.

way to keep track of nonzero elements and to identify which type of butterfly update (FBU, LBU or UBU) to perform.

To this end, we first give an example to illustrate how the structure of the vector changes after each matrix-vector product, $\mathbf{A}_q\hat{\boldsymbol{x}}_{q-1}$. Consider the 16-element vector after the bit reversal, $\hat{\boldsymbol{x}}_0$, shown in the Figure 5.11, where there are 4 nonzero elements depicted by black dots.

Figure 5.11: Initial vector: $\hat{\boldsymbol{x}}_0$

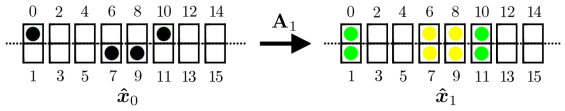


For $q = 1$, $L_q = 2$, and the elements of $\hat{\boldsymbol{x}}_0$ are evenly distributed into 8 blocks. As explained above, each element in an upper half block is used, together with a corresponding element in the lower half within the same block, to compute a butterfly update. For the first block, a UBU is performed since $\hat{x}_0[0] \neq 0$ and $\hat{x}_0[1] = 0$. After the UBU, both elements in this block are nonzero. An LBU is performed in the 4th block, since $\hat{x}_0[6] = 0$ but $\hat{x}_0[7] \neq 0$. After the LBU, both elements in this block are nonzero as well. Similarly, an LBU is performed in the 5th block and a UBU is performed in the 6th block. No operations are required for the second, third, seventh and eighth blocks. As a result, the number of nonzero elements is doubled from 4 to 8. This process is illustrated in Figure 5.12a, where green, yellow and red dots represent nonzero elements involved in UBUs, LBUs and FBUs, respectively.

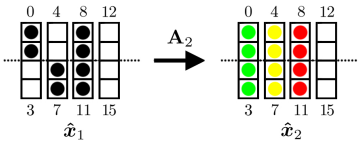
For $q = 2$, $L_q = 4$ and the elements of $\hat{\boldsymbol{x}}_1$ are evenly distributed into 4 blocks. Two UBUs are performed in the first block, and two LBUs are performed in the 2nd block. The number of nonzero elements in these two blocks are doubled.

Figure 5.12: Structure change of $\hat{\boldsymbol{x}}_q$

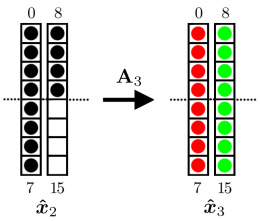
(a) $\hat{\boldsymbol{x}}_1 = \mathbf{A}_1\hat{\boldsymbol{x}}_0$



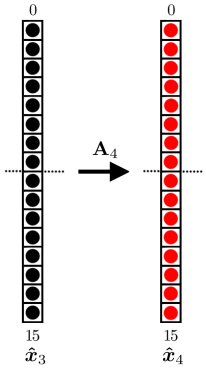
(b) $\hat{\boldsymbol{x}}_2 = \mathbf{A}_2\hat{\boldsymbol{x}}_1$



(c) $\hat{\boldsymbol{x}}_3 = \mathbf{A}_3\hat{\boldsymbol{x}}_2$



(d) $\hat{\boldsymbol{x}}_4 = \mathbf{A}_4\hat{\boldsymbol{x}}_3$



Elements in the third block of $\hat{\boldsymbol{x}}_1$ are all nonzero, thus, two FBUs are performed in this block and the number of nonzero elements in this block remains unchanged. No operations are required for the fourth block. As a result, the number of nonzero elements of $\hat{\boldsymbol{x}}_2$ is increased from 8 to 12. This process is illustrated in Figure 5.12b.

For $q = 3$, $L_q = 8$ and the elements of $\hat{\boldsymbol{x}}_2$ are evenly distributed into 2 blocks. Four FBUs are performed in the first block, and four UBUs are performed in the second block. The number of nonzero elements of $\hat{\boldsymbol{x}}_3$ is increased from 12 to 16. This process is illustrated in Figure 5.12c.

Finally, for $q = 4$, $L_q = 16$ and all elements of $\hat{\boldsymbol{x}}_3$ are merged into 1 block, and 8 FBUs are performed. The number of nonzero elements of $\hat{\boldsymbol{x}}_4$ remains unchanged. This is illustrated in Figure 5.12d.

This example shows that, at each iteration q, the nonzero elements within the same block are all involved in the same type of butterfly update. Moreover, each LBU or UBU generates a new nonzero element, and at the end of the iteration, the blocks where the butterfly updates are performed, which we call the "active blocks", are filled with nonzero elements. Therefore, to keep track of active blocks in the qth iteration, at the end of $(q-1)$st iteration, we put the index of the first element in each active block into an index vector $\hat{\boldsymbol{d}}_{q-1}$, and store the length of $\hat{\boldsymbol{d}}_{q-1}$ in $\hat{C}_{q-1}$. In our example, $\hat{\boldsymbol{d}}_0 = [0, 7, 9, 10]^T$, $\hat{\boldsymbol{d}}_1 = [0, 6, 8, 10]^T$, $\hat{\boldsymbol{d}}_2 = [0, 4, 8]^T$ and $\hat{\boldsymbol{d}}_3 = [0, 8]^T$. Then at the qth iteration, we can use the following criteria to decide which type of butterfly updates should be used in each active block.

- If $\hat{d}_{q-1}[k] \bmod L_q = 0$ and $\hat{d}_{q-1}[k+1] - \hat{d}_{q-1}[k] = L_{q-1}$, then we perform FBUs on all elements in the active block beginning with $\hat{x}_{q-1}\left[\hat{d}_{q-1}[k]\right]$, and the number of active blocks, $\hat{C}_q$ is reduced by 1. For example, as shown in Figure 5.12b, at the second iteration, $q = 2$, $L_2 = 4$, $L_1 = 2$,

$$\hat{d}_1[2] \bmod L_2 = 8 \bmod 4 = 0,$$
$$\hat{d}_1[3] - \hat{d}_1[2] = 10 - 8 = 2 = L_1,$$

  and FBUs are performed on the third block (from $\hat{x}_1[8]$ to $\hat{x}_1[11]$);

- If $\hat{d}_{q-1}[k] \bmod L_q = 0$ and $\hat{d}_{q-1}[k+1] - \hat{d}_{q-1}[k] \neq L_{q-1}$, then we perform UBUs on all elements in the active block beginning with $\hat{x}_{q-1}\left[\hat{d}_{q-1}[k]\right]$, and the number of active blocks, $\hat{C}_q$, remains unchanged. For example, as shown in Figure 5.12b, at the second iteration, $q = 2$, $L_2 = 4$, $L_1 = 2$,

$$\hat{d}_1[0] \bmod L_2 = 0 \bmod 4 = 0,$$
$$\hat{d}_1[1] - \hat{d}_1[0] = 6 - 0 = 6 \neq L_1,$$

and UBUs are performed on the first block (from $\hat{x}_1[0]$ to $\hat{x}_1[3]$);

- If $\hat{d}_{q-1}[k] \bmod L_q \neq 0$, then we perform LBUs on all elements in the active block beginning with $\hat{x}_{q-1}\left[\hat{d}_{q-1}[k]\right]$, and the number of active blocks, $\hat{C}_q$, remains unchanged. For example, as shown in Figure 5.12b, at the second iteration, $q = 2$, $L_2 = 4$, $L_1 = 2$,

$$\hat{d}_1[1] \bmod L_2 = 6 \bmod 4 \neq 0,$$

and LBUs are performed on the second block (from $\hat{x}_1[4]$ to $\hat{x}_1[7]$).

We are now able to specify the sparse FFT algorithm. To begin, Algorithm 5.10 presents the sparse butterfly updates to compute $\hat{\boldsymbol{x}}_q = \mathbf{A}_q\hat{\boldsymbol{x}}_{q-1}$, where we let $\hat{C}_q$, $\hat{C}_q^F$, $\hat{C}_q^L$ and $\hat{C}_q^U$ be the number of active blocks, FBU blocks, LBU blocks and UBU blocks in the qth iteration respectively.[11] In total, $5\hat{C}_q+3(L_q^*+1)\hat{C}_q^F+(3L_q^*+2)\hat{C}_q^L+2L_q^*\hat{C}_q^U+3$ INOPs and $10L_q^*\hat{C}_q^F+8L_q^*\hat{C}_q^L$ FLOPs are required for each $\hat{\boldsymbol{x}}_q = \mathbf{A}_q\hat{\boldsymbol{x}}_{q-1}$. Then Algorithm 5.11 uses Algorithms 5.5, 5.9 and 5.10 to compute the loss probability. It is worth mentioning that, we still apply the full FFT method to compute the inverse FFT, since each $\mathcal{F}(\boldsymbol{p}_n^{\boldsymbol{z}})$ is normally not sparse, which should be obvious from our example above.

[11] We assume that $i_{x0} \bmod L_q$ can be computed using the equivalent of one integer arithmetic operation.

Algorithm 5.10 Sparse butterfly update to compute $\hat{\boldsymbol{x}}_q = \mathbf{A}_q \hat{\boldsymbol{x}}_{q-1}$

Input: $\boldsymbol{x}$, K, q, $\boldsymbol{w}_{long}$, $\hat{\boldsymbol{d}}$, $\hat{C}$;

Output: $\boldsymbol{x}$, $\hat{\boldsymbol{d}}$, $\hat{C}$;

$L_q = 2^q$; $L_q^* \leftarrow L_q/2$; $r \leftarrow K/L_q$; $k^* \leftarrow 0$; $\hat{C}^* \leftarrow \hat{C}$; # 3 INOPs

for $k = 0 : \hat{C} - 1$ **do**

  $i_{x0} \leftarrow \hat{d}[k]$; $i_{x1} \leftarrow i_{x0} + L_q^*$; # $\hat{C}_q$ INOPs

  if $i_{x0} \bmod L_q == 0$ **then** # $\hat{C}_q$ INOPs

    $\hat{d}[k^*] \leftarrow i_{x0}$;

    if $\hat{d}[k+1] - i_{x0} == L_q^*$ **then** # FBU # $2\hat{C}_q$ INOPs

      $i_w \leftarrow L_q^* - 1$; # $\hat{C}_q^F$ INOPs

      for $j = 0 : L_q^* - 1$ **do**

        $y_0 \leftarrow x[i_{x0}]$;

        $\tau \leftarrow x[i_{x1}] w_{long}[i_w]$; # $6L_q^*\hat{C}_q^F$ FLOPs

        $x[i_{x1}] \leftarrow y_0 - \tau$; $x[i_{x0}] \leftarrow y_0 + \tau$; # $4L_q^*\hat{C}_q^F$ FLOPs

        $i_{x0} \leftarrow i_{x0} + 1$; $i_{x1} \leftarrow i_{x1} + 1$; $i_w \leftarrow i_w + 1$; # $3L_q^*\hat{C}_q^F$ INOPs

      end for

      $k \leftarrow k + 1$; # $\hat{C}_q^F$ INOPs

      $\hat{C}^* \leftarrow \hat{C}^* - 1$; # $\hat{C}_q^F$ INOPs

    else # UBU

      for $j = 0 : L_q^* - 1$ **do**

        $x[i_{x1}] \leftarrow x[i_{x1}]$; $x[i_{x0}] \leftarrow x[i_{x0}]$;

        $i_{x0} \leftarrow i_{x0} + 1$; $i_{x1} \leftarrow i_{x1} + 1$; # $2L_q^*\hat{C}_q^U$ INOPs

      end for

    end if

  else # LBU

    $\hat{d}[k^*] \leftarrow i_{x0} - L_q^*$; # $\hat{C}_q^L$ INOPs

    $i_w \leftarrow L_q^* - 1$; # $\hat{C}_q^L$ INOPs

    for $j = 0 : L_q^* - 1$ **do**

      $\tau \leftarrow x[i_{x1}] w_{long}[i_w]$; # $6L_q^*\hat{C}_q^L$ FLOPs

      $x[i_{x1}] \leftarrow -\tau$; $x[i_{x0}] \leftarrow \tau$; # $2L_q^*\hat{C}_q^L$ FLOPs

      $i_{x0} \leftarrow i_{x0} + 1$; $i_{x1} \leftarrow i_{x1} + 1$; $i_w \leftarrow i_w + 1$; # $3L_q^*\hat{C}_q^L$ INOPs

    end for

  end if

  $k^* \leftarrow k^* + 1$; # $\hat{C}_q$ INOPs

end for

$\hat{C} \leftarrow \hat{C}^*$;

return $\boldsymbol{x}$, $\hat{\boldsymbol{d}}$, $\hat{C}$;

Algorithm 5.11 Sparse FFT method to compute the conditional loss probabilities

Input: $\boldsymbol{p}_n^{\boldsymbol{z}}$, K, t;

Output: $\tilde{\boldsymbol{p}}$;

Compute $\boldsymbol{w}_{long}$ by Algorithm 5.5;
$\tilde{\boldsymbol{p}} \leftarrow [1, 1, \ldots, 1]^T$;

for $n = 0 : N - 1$ **do**
    $\boldsymbol{p}_n^{\boldsymbol{z}} \leftarrow \mathbf{P}_K^T \boldsymbol{p}_n^{\boldsymbol{z}}$ and $\hat{\boldsymbol{d}}$ by Algorithm 5.9;
    `sort(`$\hat{\boldsymbol{d}}$`)`;
    for $q = 1 : t$ **do**
        $\boldsymbol{p}_n^{\boldsymbol{z}} \leftarrow \mathbf{A}_q \boldsymbol{p}_n^{\boldsymbol{z}}$ by Algorithm 5.10;
    end for
    $\tilde{\boldsymbol{p}} \leftarrow \tilde{\boldsymbol{p}} \odot \boldsymbol{p}_n^{\boldsymbol{z}}$;
end for

$\boldsymbol{w}_{long} \leftarrow \overline{\boldsymbol{w}_{long}}$;
$\tilde{\boldsymbol{p}} \leftarrow \mathbf{P}_K^T \tilde{\boldsymbol{p}}$ by Algorithm 5.4;
for $q = 1 : t$ **do**
    $\tilde{\boldsymbol{p}} \leftarrow \overline{\mathbf{A}_q} \tilde{\boldsymbol{p}}$ by Algorithm 5.6;
end for
$\tilde{\boldsymbol{p}} \leftarrow \tilde{\boldsymbol{p}} / K$;
return $\tilde{\boldsymbol{p}}$;

It should be mentioned that, the sparse FFT method proposed in this subsection is different from the sparse FFT methods in the existing literatures [1, 7, 16, 27, 28, 29, 30, 35, 37, 38, 50, 54, 2, 42]. The fundamental difference lies in that, our sparse FFT method utilizes the sparsity of the input vector $\boldsymbol{x}$, while the existing sparse FFT methods apply to the cases where the output vector $\mathcal{F}(\boldsymbol{x})$ is sparse.

5.2.3.4 Complexity of Sparse FFT Method

The computational cost of Algorithm 5.11 is presented in Table 5.2.

Table 5.2: Computational cost of Algorithm 5.11

Line	Number of TROPs	Number of INOPs	Number of FLOPs
1	$2(K-1)$	$K+2\log_2 K-1$	$K+2\log_2 K-1$
5	0	$5NC\log_2 K$	0
6	0	$O(C\log_2 C)$ [12]	0
8	0	$\sum_{n=0}^{N-1}\sum_{q=1}^{t}\Big(5\hat{C}_{n,q}+3(L_q^*+1)\hat{C}_{n,q}^F$ $+(3L_q^*+2)\hat{C}_{n,q}^L+2L_q^*\hat{C}_{n,q}^U+3\Big)$ [13]	$\sum_{n=0}^{N-1}\sum_{q=1}^{t}\Big(10L_q^*\hat{C}_{n,q}^F+8L_q^*\hat{C}_{n,q}^L\Big)$
10	0	0	$6(N-1)K$
13	0	0	$K-1$
14	0	$5K\log_2 K$	0
16	0	$\sum_{q=1}^{t}\Big(3\frac{K}{L_q}+\frac{3}{2}K+3\Big)$	$5K\log_2 K$
18	0	0	K

Though it is clear that the sparse FFT method works faster than the full FFT method, the performance of Line 8 depends on the structure of the input data $\boldsymbol{p}_n^{\boldsymbol{z}}$, which makes it difficult to compare the sparse FFT method to other methods in terms of efficiency. However, as for the sparse convolution method, we are able to construct the best and worse cases in terms of complexity for the matrix-vector product in Line 8.

For simplicity, assume $C=2^s$, where $s<t$. Notice that, compared with the full FFT method, at each q-iteration, the computational saving in the sparse FFT method comes from sparse updates (UBD or LBD) and from "null updates", for which both $\hat{x}_{q-1}\left[kL_q+j\right]$ and $\hat{x}_{q-1}\left[kL_q+L_q^*+j\right]$ are zero and no update is needed. The following theorem shows the total number of sparse butterfly updates is fixed.

[12] Good sorting algorithms (Merge sort, Introsort, etc.) can achieve this with $O(C\log_2 C)$ INOPs.

[13] Note that for each obligor n, the number of active blocks, FBU blocks, LBU blocks and UBU blocks in the qth iteration are normally different.

Theorem 5.6. *Let $\boldsymbol{x}$ be a vector with $K = 2^t$ elements and $C > 0$ nonzero elements. If the sparse FFT method is applied to compute $\mathcal{F}(\boldsymbol{x})$, then there are $K - C$ sparse updates (LBU or UBU).*

Proof. First, we prove that all the elements of $\mathcal{F}(\boldsymbol{x})$ are nonzero elements.[14] It is sufficient to show that this is the case for $C = 1$. Suppose $x[k^*] \neq 0$ and $x[k] = 0$ for all $k \neq k^*$, $k = 0, \ldots, K-1$. From the definition of discrete Fourier transform, we have

$$\hat{x}_t[k] = (\mathcal{F}(\boldsymbol{x}))[k] = \sum_{j=0}^{K-1} x[j] w^{jk} = x[k^*] \left(\cos\left(\frac{-2\pi k k^*}{K} \right) + i \sin\left(\frac{-2\pi k k^*}{K} \right) \right).$$

Since $\cos(-2\pi k k^*/K)$ and $\sin(-2\pi k k^*/K)$ can not be zero at the same time and $x[k^*] \neq 0$, $(\mathcal{F}(\boldsymbol{x}))[k] \neq 0$ for all $k = 0, \ldots, K-1$. Moreover, as mentioned earlier, only LBU or UBU produces fill-in (i.e., increases the number of nonzero elements), and each of them adds one nonzero element per update. Therefore, we need $K - C$ sparse updates (either LBU or UBU) to convert the $K - C$ zero elements in $\boldsymbol{x}$ to nonzero elements in $\mathcal{F}(\boldsymbol{x})$. Once $\hat{x}_{q^*}[k]$ becomes nonzero, $\hat{x}_q[k]$ remains nonzero for all $q > q^*$ (except in rare cases, as noted in Footnote 10). Therefore, the total number of sparse updates is $K - C$. □

We need to find a way to allocate these $K - C$ sparse updates to different q-iterations to maximize or minimize the number of zero updates to obtain the best case and the worse case, respectively. Recall that at each q-iteration, the total number of butterfly updates is fixed at $K/2$. A sparse update at the qth iteration converts a zero element to a nonzero element, which reduces the number of zero updates at this iteration by one. Moreover, the zero element involved in this sparse update normally remains nonzero for all later iterations, which means the numbers of zero updates in the later iterations are all reduced by one. Therefore, to minimize the total number of zero updates, we allocate the $K - C$ sparse updates as early as possible. Likewise, we need to delay the $K - C$ sparse updates as much as possible to maximize the total number of zero updates.

Based on the analysis above, the best case and the worst case for the sparse FFT method are constructed as follows.

Best case In this case, we maximize the number of zero updates and make all $K - C$ sparse updates be UBUs. Specifically, we let

$$\begin{cases} \hat{x}_0[k] \neq 0, & k = 0, \ldots, 2^s - 1, \\ \hat{x}_0[k] = 0, & k = 2^s, \ldots, 2^t - 1. \end{cases} \tag{5.2.20}$$

[14] The term "nonzero element" here means a "structurally nonzero element", as explained in Footnote 10.

Then from the first iteration to the sth iteration, only FBUs are performed. Starting from the $(s+1)$th iteration, only UBUs are applied. The vector $\hat{\boldsymbol{x}}_q$ is filled up with nonzeros in the last iteration. This is depicted in Figure 5.13a, in which $K = 16$, $C = 4$. If we use the notation employed in Algorithm 5.10, then

$$\begin{cases} \hat{C}_q = \hat{C}_q^F = 2^{s-q},\ \hat{C}_q^L = \hat{C}_q^U = 0, & q = 1, \ldots, s, \\ \hat{C}_q = \hat{C}_q^U = 1,\ \hat{C}_q^L = \hat{C}_q^F = 0 & q = s+1, \ldots, t. \end{cases} \tag{5.2.21}$$

In total,

$$\sum_{q=1}^{s} \hat{C}_q^F L_q^* = \sum_{q=1}^{s} 2^{s-q} \cdot 2^{q-1} = C \log_2 C/2$$

FBUs and

$$\sum_{q=s+1}^{t} \hat{C}_q^U L_q^* = \sum_{q=s+1}^{t} 2^{q-1} = K - C$$

UBUs are used.

Worse case In this case, we minimize the number of zero updates and make all $K - C$ sparse updates be LBUs. Specifically, we let

$$\begin{cases} \hat{x}_0 \left[j 2^{t-s} - 1 \right] \neq 0, & j = 1, \ldots, 2^s, \\ \hat{x}_0[k] = 0, & \text{otherwise.} \end{cases} \tag{5.2.22}$$

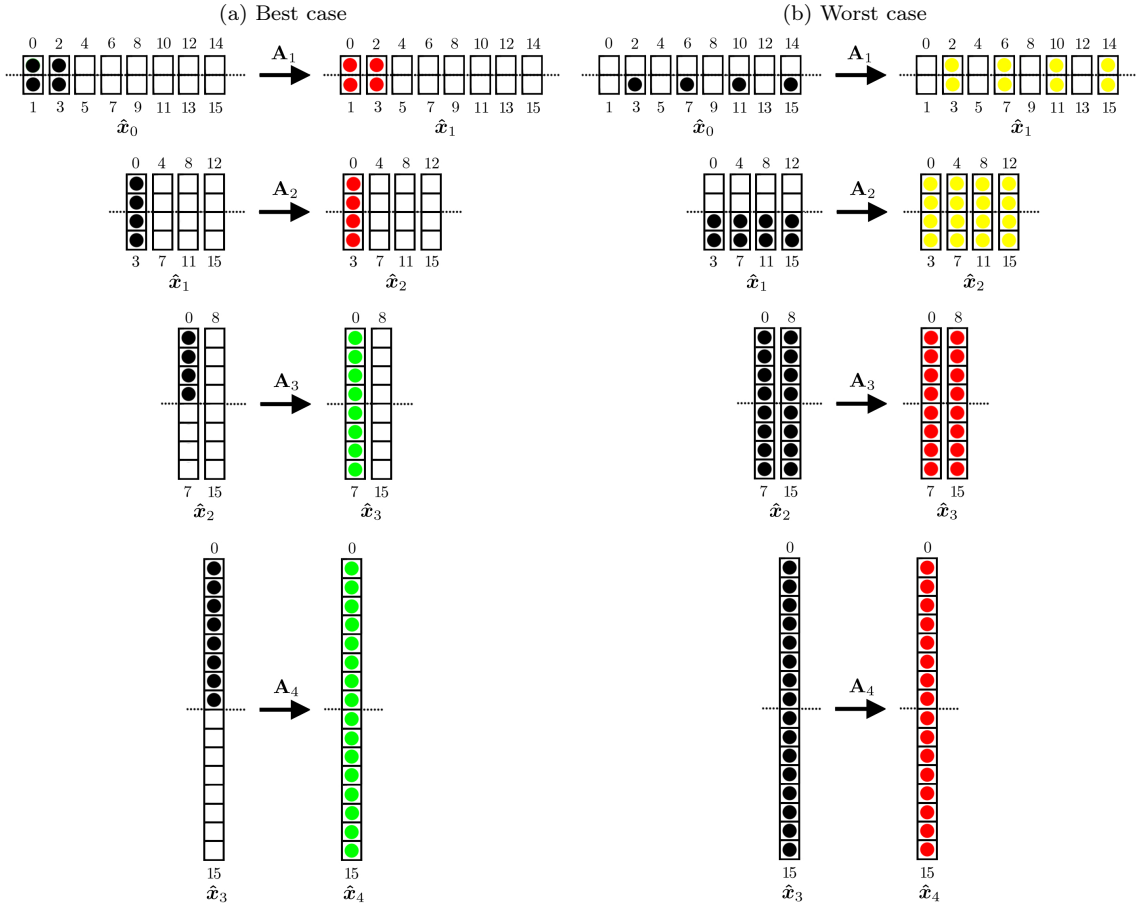
Then from the first iteration to the $(t-s)$th iteration, only LBUs are performed. At the end of the $(t-s)$th iteration, the vector is filled with all nonzero elements. Starting from the $(t-s+1)$st iteration, FBUs are applied to every element. This is depicted in Figure 5.13b, in which $K = 16$, $C = 4$. If we use the notation employed in Algorithm 5.10, then

$$\begin{cases} \hat{C}_q = \hat{C}_q^L = 2^s,\ \hat{C}_q^F = \hat{C}_q^U = 0, & q = 1, \ldots, t-s, \\ \hat{C}_q = \hat{C}_q^F = 2^{t-q},\ \hat{C}_q^L = \hat{C}_q^U = 0, & q = t-s+1, \ldots, t. \end{cases} \tag{5.2.23}$$

In total,

$$\begin{aligned} \sum_{q=1}^{t-s} \hat{C}_q^L L_q^* &= \sum_{q=1}^{t-s} 2^s \cdot 2^{q-1} \\ &= K - C \end{aligned}$$

Figure 5.13: Sparse FFT method ($t = 4, s = 2$)



LBUs and

$$\sum_{q=t-s+1}^{t} \hat{C}_q^F L_q^* = \sum_{q=t-s+1}^{t} 2^{t-q} \cdot 2^{q-1}$$
$$= K \log_2 C/2$$

FBUs are used.

We can extend this analysis to the computation of the loss probabilities $\tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}}$. First, both the best case and the worse case are not achievable when the sparse FFT algorithm is applied to compute $\tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}}$. In the best case, $\hat{p}_n^{\boldsymbol{z}}[1]$ is nonzero for $n = 0, \ldots, N-1$, where $\hat{\boldsymbol{p}}_n^{\boldsymbol{z}}$ is the vector obtained after bit-reversing $\boldsymbol{p}_n^{\boldsymbol{z}}$. Hence, $p_n^{\boldsymbol{z}}[K/2]$ is nonzero for $n = 0, \ldots, N-1$. Therefore, $\boldsymbol{p}_0^{\boldsymbol{z}} * \boldsymbol{p}_1^{\boldsymbol{z}}$ has length at least $K+1$ since a nonzero element occurs at $K/2+K/2 = K$, indicating that there is nonzero possibility that the portfolio loss is l_K. However, this contradicts our assumption that the largest portfolio loss is $l_{K-1} = (K-1)\delta$, which is enforced by the the way we choose K. A similar argument can be applied to the worst case,

in which nonzero $\hat{p}_n^{\boldsymbol{z}}[K-1]$ implies nonzero $p_n^{\boldsymbol{z}}[K-1]$. If we denote Ψ_{sfft}^b and Ψ_{sfft}^w as the amount of computation needed in Algorithm 5.11 in the best case and in the worst case, respectively, then

$$\Psi_{sfft}^b \leq \Psi_{sfft} \leq \Psi_{sfft}^w,$$

where Ψ_{sfft} is the amount of computation needed in Algorithm 5.11 to compute the loss probabilities $\tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}}$in any feasible case.

It is very difficult to obtain the achievable lower bound and upper bound for Ψ_{sfft} because of the bit reversal operations. Therefore, we analyze Ψ_{sfft}^b and Ψ_{sfft}^w instead. For the best case, we can let the permuted conditional loss probability vectors have the same structure as in (5.2.20). That is,

$$\begin{cases} \hat{q}_{n,0}^{\boldsymbol{z}}[k] \neq 0, & k = 0, \ldots, 2^s - 1, \\ \hat{q}_{n,0}^{\boldsymbol{z}}[k] = 0, & k = 2^s, \ldots, 2^t - 1, \end{cases}$$

for all obligors $n = 0, \ldots, N-1$. In this case, by(5.2.21), Line 8 in Algorithm 5.11 costs

$$\begin{aligned} &\sum_{n=0}^{N-1} \sum_{q=1}^{t} \left(5\hat{C}_{n,q} + 3(L_q^* + 1)\hat{C}_{n,q}^F + (3L_q^* + 2)\hat{C}_{n,q}^L + 2L_q^* \hat{C}_{n,q}^U + 3 \right) \\ &= N \left(\sum_{q=1}^{s} \left(5 \cdot 2^{s-q} + 3(2^{q-1} + 1)2^{s-q} + 3 \right) + \sum_{q=s+1}^{t} \left(5 \cdot 1 + 2 \cdot 2^{q-1} \cdot 1 + 3 \right) \right) \\ &= N \left(2K + \frac{3}{2} C \log_2 C + 6C + 7 \log_2 K - 5 \log_2 C - 8 \right) \end{aligned}$$

INOPs and

$$\sum_{n=0}^{N-1} \sum_{q=1}^{t} \left(10L_q^* \hat{C}_{n,q}^F + 8L_q^* \hat{C}_{n,q}^L \right) = N \left(\sum_{q=1}^{s} 10 \cdot 2^{q-1} \cdot 2^{s-q} \right) = 5NC \log_2 C$$

FLOPs. Therefore, in the best case, the total amount of computation needed in Algorithm 5.11 is

- TROPs: $\Psi_{sfft}^{TROP,b} = 2(K-1)$;

- INOPs:

$$\begin{aligned}\Psi_{sfft}^{INOP,b} &= (K + 2\log_2 K - 1) + 5NC\log_2 K + NC\log_2 C \\ &\quad + N\left(2K + \frac{3}{2}C\log_2 C + 6C + 7\log_2 K - 5\log_2 C - 8\right) \\ &\quad + 5K\log_2 K + \sum_{q=1}^{t}\left(3\frac{K}{2^q} + \frac{3}{2}K + 3\right) \\ &= 2NK + \left(\frac{13}{2}\log_2 K + 4\right)K \\ &\quad + \left(5C\log_2 K + \frac{5}{2}C\log_2 C + 6C + 7\log_2 K - 5\log_2 C - 8\right)N \\ &\quad + (5\log_2 K - 4)\,;\end{aligned}$$

- FLOPs:

$$\begin{aligned}\Psi_{sfft}^{FLOP,b} &= (K + 2\log_2 K - 1) + 5NC\log_2 C + 6(N-1)K \\ &\quad + (K-1) + 5K\log_2 K + K \\ &= 6NK + (5\log_2 K - 3)K + 5NC\log_2 C + 2\log_2 K - 2.\end{aligned}$$

In the best case, all $\overline{\boldsymbol{p}}_n^{\boldsymbol{z}}$ have the same length, $\bar{K} = \frac{K-1}{N} + 1$. Therefore, by (5.1.36), the speedup of the sparse FFT method over the full convolution method in this best case is

$$\begin{aligned}\frac{\Psi_{conv}}{\Psi_{fft}^b} &= \frac{\left(1 - \frac{1}{N}\right)K^2 + \left(\frac{N-3}{2} + \frac{1}{N}\right)K + \frac{N-1}{2}}{\beta\Psi_{fft}^{TROP,b} + \Psi_{fft}^{INOP,b} + \Psi_{fft}^{FLOP,b}} \\ &= \frac{\left(1 - \frac{1}{N}\right)K^2 + \left(\frac{N-3}{2} + \frac{1}{N}\right)K + \frac{N-1}{2}}{8NK + f_{sfft}^b(K,\beta)K + g_{sfft}^b(K,C)N + h_{sfft}^b(K,\beta)}\end{aligned}$$

where

$$\begin{aligned}f_{sfft}^b(K,\beta) &= \frac{23}{2}\log_2 K + 2\beta + 1, \\ g_{sfft}^b(K,C) &= 5C\log_2 K + \frac{15}{2}C\log_2 C + 6C + 7\log_2 K - 5\log_2 C - 8, \\ h_{sfft}^b(K,\beta) &= 7\log_2 K - 2\beta - 6).\end{aligned}$$

Hence,

$$\begin{aligned}\frac{\Psi_{conv}}{\Psi^b_{fft}} &= \frac{K^2\left(\left(1-\frac{1}{N}\right)+\left(\frac{N-3}{2}+\frac{1}{N}\right)\frac{1}{K}+\frac{N-1}{2}\frac{1}{K^2}\right)}{NK\left(8+\frac{f^b_{sfft}(K,\beta)}{N}+\frac{g^b_{sfft}(K,C)}{K}+\frac{h^b_{sfft}(K,\beta)}{NK}\right)} \\ &\overset{K\gg N}{\approx} \frac{K}{8N} \\ &= \Omega\left(K/N\right). \end{aligned} \tag{5.2.24}$$

Similarly, for the worst case,

$$\begin{cases} \hat{q}^{\mathbf{z}}_{n,0}\left[j2^{t-s}-1\right]\neq 0, & j=1,\ldots,2^s, \\ \hat{q}^{\mathbf{z}}_{n,0}[k]=0, & \text{otherwise}, \end{cases}$$

for all obligors $n=0,\ldots,N-1$. In this case, by (5.2.23), Line 8 in Algorithm 5.11 costs

$$\begin{aligned} &\sum_{n=0}^{N-1}\sum_{q=1}^{t}\left(5\hat{C}_{n,q}+3(L^*_q+1)\hat{C}^F_{n,q}+(3L^*_q+2)\hat{C}^L_{n,q}+2L^*_q\hat{C}^U_{n,q}+3\right) \\ &= N\left(\sum_{q=1}^{t-s}\left(5\cdot 2^s+(3\cdot 2^{q-1}+2)2^s+3\right)+\sum_{q=t-s+1}^{t}\left(5\cdot 2^{t-q}+3\cdot\left(2^{q-1}+1\right)\cdot 2^{t-q}+3\right)\right) \\ &= N\left(\left(\frac{3}{2}\log_2 C+3\right)K+(7C+3)\log_2 K+(-7C\log_2 C+5C-8)\right) \end{aligned}$$

INOPs and

$$\begin{aligned} &\sum_{n=0}^{N-1}\sum_{q=1}^{t}\left(10L^*_q\hat{C}^F_{n,q}+8L^*_q\hat{C}^L_{n,q}\right) \\ &= N\left(\sum_{q=1}^{t-s}8\cdot 2^{q-1}\cdot 2^s+\sum_{q=t-s+1}^{t}10\cdot 2^{q-1}\cdot 2^{t-q}\right) \\ &= 8N(K-C)+5KN\log_2 C \end{aligned}$$

FLOPs. Therefore, the total amount of computation needed in Algorithm 5.11 is

- TROPs: $\Psi^{TROP,w}_{sfft}=2(K-1)$;

- INOPs:

$$\begin{aligned}\Psi_{sfft}^{INOP,w} &= (K + 2\log_2 K - 1) + 5NC\log_2 K + NC\log_2 C \\ &\quad + N\left(\left(\frac{3}{2}\log_2 C + 3\right)K + (7C+3)\log_2 K + (-7C\log_2 C + 5C - 8)\right) \\ &\quad + 5K\log_2 K + \sum_{q=1}^{t}\left(3\frac{K}{2^q} + \frac{3}{2}K + 3\right) \\ &= \left(\frac{3}{2}\log_2 C + 3\right)NK + (5\log_2 K + 4)K \\ &\quad + (12C\log_2 K - 6C\log_2 C + 5C + 3\log_2 K - 8)N + (5\log_2 K - 4);\end{aligned}$$

- FLOPs:

$$\begin{aligned}\Psi_{sfft}^{FLOP,w} &= (K + 2\log_2 K - 1) + (8N(K-C) + 5KN\log_2 C) + 6(N-1)K \\ &\quad + (K-1) + 5K\log_2 K + K \\ &= (5\log_2 C + 14)NK + (5\log_2 K - 3)K - 8CN + 2(\log_2 K - 1).\end{aligned}$$

Thus, the speedup of the sparse FFT method over the full convolution method in the best case is

$$\begin{aligned}\frac{\Psi_{conv}}{\Psi_{fft}^{w}} &= \frac{\left(1-\frac{1}{N}\right)K^2 + \left(\frac{N-3}{2} + \frac{1}{N}\right)K + \frac{N-1}{2}}{\beta\Psi_{fft}^{TROP,w} + \Psi_{fft}^{INOP,w} + \Psi_{fft}^{FLOP,w}} \\ &= \frac{\left(1-\frac{1}{N}\right)K^2 + \left(\frac{N-3}{2} + \frac{1}{N}\right)K + \frac{N-1}{2}}{\left(17 + \frac{13}{2}\log_2 C\right)NK + f_{sfft}^{w}(K,\beta)K + g_{sfft}^{w}(K,C)N + h_{sfft}^{w}(K,\beta)}\end{aligned}$$

where

$$\begin{aligned}f_{sfft}^{w}(K,\beta) &= 10\log_2 K + 2\beta + 1, \\ g_{sfft}^{w}(K,C) &= 12C\log_2 K - 6C\log_2 C - 3C + 3\log_2 K - 8, \\ h_{sfft}^{w}(K,\beta) &= 7\log_2 K - 2\beta - 6.\end{aligned}$$

Hence,

$$\begin{aligned}\frac{\Psi_{conv}}{\Psi_{fft}^{w}} &= \frac{K^2\left(\left(1-\frac{1}{N}\right) + \left(\frac{N-3}{2} + \frac{1}{N}\right)\frac{1}{K} + \frac{N-1}{2}\frac{1}{K^2}\right)}{NK\left(\left(17 + \frac{13}{2}\log_2 C\right) + \frac{f_{sfft}^{w}(K,\beta)}{N} + \frac{g_{sfft}^{w}(K,C)}{K} + \frac{h_{sfft}^{w}(K,\beta)}{NK}\right)} \\ &\overset{K \gg N}{\approx} \frac{K}{N\left(17 + \frac{13}{2}\log_2 C\right)} \\ &= \Omega\left(K/\left(N\log_2 C\right)\right). \qquad (5.2.25)\end{aligned}$$

Comparing (5.2.24) and (5.2.25) with (5.2.15), we see that the speedup of the sparse FFT method over the full FFT method is

$$\frac{\Psi_{fft}}{\Psi^b_{sfft}} \sim \Omega(\log_2 K) \tag{5.2.26}$$

in the best case and

$$\frac{\Psi_{fft}}{\Psi^w_{sfft}} \sim \Omega(\log_2 K / \log_2 C) \tag{5.2.27}$$

in the worst case.

5.2.4 Truncated Sparse FFT Method

5.2.4.1 Truncated N-fold Convolution

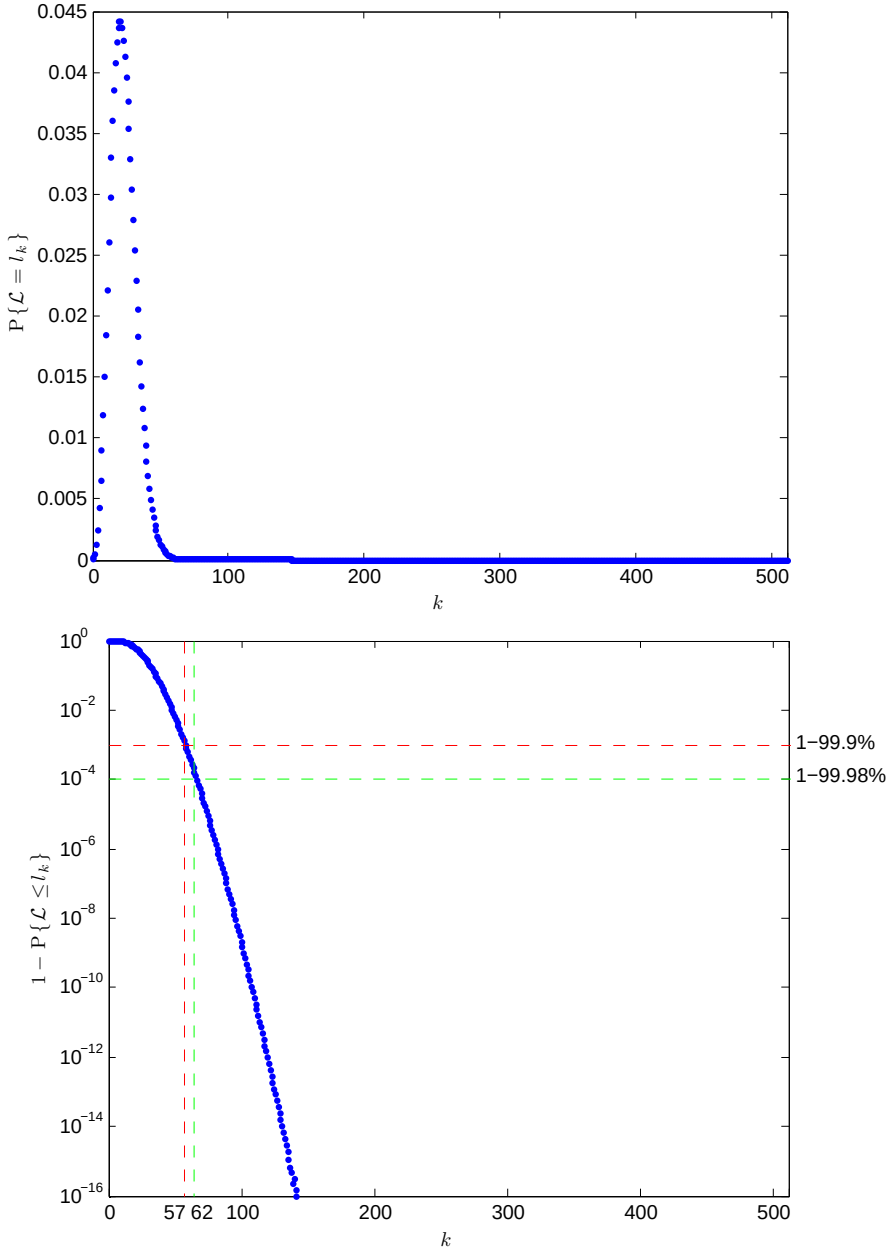
The distribution of portfolio credit losses often has a very fat tail, which means the distribution is quite skewed and there is a small, but not insignificant, probability for the occurrence of a very large loss. Figure 5.14 shows a typical distribution of portfolio credit losses. In this example, we set $K = 2^9 = 512$, $l_0 = L_{min}$ and $l_{K-1} = L_{max}$, where L_{min} and L_{max} are the minimum and maximum portfolio losses, respectively. With this discretization scheme, each vector $\boldsymbol{p}_n^{\boldsymbol{z}}$ has a length of K, with many zeros padded on the end of $\boldsymbol{p}_n^{\boldsymbol{z}}$. From Figure 5.14, the range of the portfolio losses is from l_0 to l_{511}, but the probabilities $\mathbb{P}\{\mathcal{L} = l_k\}$ are very small for $k \geq 100$, and the tail probabilities $\mathbb{P}\{\mathcal{L} > l_k\}$ are less than 10^{-8} for $k \geq 100$. In practice, risk managers are primarily concerned about calculating the 99.9% credit VaR for regulation capital, and the 99.98% credit VaR for economic capital to maintain an AA rating. In this example, the 99.9% credit VaR corresponds to l_{57} and 99.98% credit VaR is between l_{61} and l_{62}. Thus both the 99.9% and 99.98% credit VaR are much less than the maximum loss l_{K-1}. Moreover, in the computation of VaR, we are not concerned with the probabilities of portfolio losses exceeding VaR. Therefore, if we truncate $\boldsymbol{p}_n^{\boldsymbol{z}}$ by dropping many zeros in its tail such that $l_{\overline{K}-1} \geq \text{VaR}$, where $\overline{K}$ is the length of the truncated vectors $\overline{\boldsymbol{p}_n^{\boldsymbol{z}}}$, then we can compute the N-fold convolution of $\overline{\boldsymbol{p}_n^{\boldsymbol{z}}}$ to approximate the conditional probabilities $\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} = l_k\}$ for $k = 0, \ldots, \overline{K} - 1$.

We let $\overline{K} = 2^{\bar{t}}$, where

$$\bar{t} = \min\left\{ t \in \mathbb{N} \,\middle|\, l_{2^t - 1} \geq \max\left(\text{VaR}_\alpha, L_0^0, \ldots, L_{N-1}^0\right) \right\}, \tag{5.2.28}$$

and VaR_α is the credit VaR with a confidence level α. Construct the truncated vectors $\overline{\boldsymbol{p}_n^{\boldsymbol{z}}}$ by $\overline{p_n^{\boldsymbol{z}}}[k] = p_n^{\boldsymbol{z}}[k]$ for $k = 0, \ldots, \overline{K} - 1$. The reason we put L_n^0 in the maximum function in (5.2.28) is that we do not want to drop nonzero probabilities in $\boldsymbol{p}_n^{\boldsymbol{z}}$. Since $L_n^0 = w_n \text{LGC}_n^0$ is the largest loss for obligor n, then the

Figure 5.14: Fat-tailed distribution
Top: probability mass Bottom: 1-CDF



definition of $\bar{t}$ in (5.2.28) makes $p_n^{\boldsymbol{z}}[k] = 0$ for $k \geq \overline{K}$, which guarantees the truncated vectors $\overline{\boldsymbol{p}_n^{\boldsymbol{z}}}$ include all nonzero conditional loss probabilities for obligor n.

Next, the conditional portfolio loss probabilities can be approximated by the N-fold circular convolution of $\overline{\boldsymbol{p}_n^{\boldsymbol{z}}}$:

$$\mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} = l_k\right\} \approx \left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_n^{\boldsymbol{z}}}\right)[k] \tag{5.2.29}$$

for $k = 0, \ldots, \overline{K} - 1$.

The benefit of computing the truncated N-fold convolution lies in its efficiency. From the previous subsection, we know that the length of vectors $\boldsymbol{p}_n^{\boldsymbol{z}}$, K, is the most significant parameter influencing the computational cost for the FFT methods. Since the truncated vectors $\overline{\boldsymbol{p}_n^{\boldsymbol{z}}}$ are much shorter than the full vectors $\boldsymbol{p}_n^{\boldsymbol{z}}$, the computation of the N-fold convolution of $\overline{\boldsymbol{p}_n^{\boldsymbol{z}}}$ is expected to be much faster than the computation of the N-fold convolution of $\boldsymbol{p}_n^{\boldsymbol{z}}$. For instance, in the example shown in Figure 5.14, to compute the 99.9% or 99.98% credit VaR, if we use the full FFT method or the sparse FFT method, the length of the vectors is $K = 2^9 = 512$, while, if we use the approximation (5.2.29), then we can choose $\overline{K} = 2^6 = 64$. Therefore, instead of computing an N-fold convolution of vectors with 512 elements, we only need to compute an N-fold convolution of vectors with 64 elements. The improvement of performance is significant.

5.2.4.2 Aliasing Error

As we mentioned, (5.2.29) is an approximation to the conditional loss probability. The difference between $\mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} = l_k\right\}$ and $\left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_n^{\boldsymbol{z}}}\right)[k]$ in (5.2.29) is called the aliasing error. The following theorem gives a formula for the aliasing error.

Theorem 5.7. *Let $\overline{K} = 2^{\bar{t}}$, where $\bar{t}$ is defined in* (5.2.28)*, and construct $\overline{\boldsymbol{p}_n^{\boldsymbol{z}}}$ by $\overline{p_n^{\boldsymbol{z}}}[k] = p_n^{\boldsymbol{z}}[k]$ for $k = 0, \ldots, \overline{K} - 1$, then*

$$\mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} = l_k\right\} = \left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_n^{\boldsymbol{z}}}\right)[k] - \delta_k^a$$

where

$$\begin{aligned} \delta_k^a &= \sum_{u=1}^{U_k^{N-1}} \mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} = l_{u\overline{K}+k}\right\}, \\ U_k^{N-1} &= \max\left\{u \in \mathbb{N} \,\middle|\, 0 \leq u\overline{K} + k \leq N(\overline{K} - 1)\right\}. \end{aligned} \tag{5.2.30}$$

Proof. To begin, note that $\overline{p_n^{\boldsymbol{z}}}[k] = p_n^{\boldsymbol{z}}[k]$ for $k = 0, \ldots, \overline{K} - 1$ and $p_n^{\boldsymbol{z}}[k] = 0$ for $k \geq \overline{K}$. Therefore,

$$\left(\circledast_{n=0}^{N-1} \boldsymbol{p}_n^{\boldsymbol{z}} \right)[k] = \left(\circledast_{n=0}^{N-1} \overline{\boldsymbol{p}_n^{\boldsymbol{z}}} \right)[k] \tag{5.2.31}$$

for $k = 0, \ldots, N(\overline{K} - 1)$. We have shown that

$$\mathbb{P}\left\{ \mathcal{L}^{\boldsymbol{z}} = l_k \right\} = \left(\circledast_{n=0}^{N-1} \boldsymbol{p}_n^{\boldsymbol{z}} \right)[k] \tag{5.2.32}$$

for $k = 0, \ldots, N(K - 1)$ in Subsection 5.1.1. Thus, (5.2.31) and (5.2.32) imply

$$\mathbb{P}\left\{ \mathcal{L}^{\boldsymbol{z}} = l_k \right\} = \left(\circledast_{n=0}^{N-1} \overline{\boldsymbol{p}_n^{\boldsymbol{z}}} \right)[k] \tag{5.2.33}$$

for $k = 0, \ldots, N(\overline{K} - 1)$. By Theorem 5.4, we have

$$\left(\circledast_{n=0}^{N-1} \overline{\boldsymbol{p}_n^{\boldsymbol{z}}} \right)[k] = \left(\circledast_{n=0}^{N-1} \overline{\boldsymbol{p}_n^{\boldsymbol{z}}} \right)[k] - \sum_{u=1}^{U_k^{N-1}} \left(\circledast_{n=0}^{N-1} \overline{\boldsymbol{p}_n^{\boldsymbol{z}}} \right)[u\overline{K} + k]. \tag{5.2.34}$$

Thus, (5.2.33) and (5.2.34) imply

$$\mathbb{P}\left\{ \mathcal{L}^{\boldsymbol{z}} = l_k \right\} = \left(\circledast_{n=0}^{N-1} \overline{\boldsymbol{p}_n^{\boldsymbol{z}}} \right)[k] - \sum_{u=1}^{U_k^{N-1}} \mathbb{P}\left\{ \mathcal{L}^{\boldsymbol{z}} = l_{u\overline{K}+k} \right\},$$

which completes the proof. □

It should be noted that, Theorem 5.7 shows that

$$\mathbb{P}\left\{ \mathcal{L}^{\boldsymbol{z}} = l_k \right\} \leq \left(\circledast_{n=0}^{N-1} \overline{\boldsymbol{p}_n^{\boldsymbol{z}}} \right)[k],$$

hence

$$\mathbb{P}\left\{ \mathcal{L}^{\boldsymbol{z}} \leq l_k \right\} = \sum_{k \leq m} \mathbb{P}\left\{ \mathcal{L}^{\boldsymbol{z}} = l_k \right\} \leq \sum_{k \leq m} \left(\circledast_{n=0}^{N-1} \overline{\boldsymbol{p}_n^{\boldsymbol{z}}} \right)[k].$$

Consequently, VaR calculated with truncated probability vectors, $\overline{\boldsymbol{p}_n^{\boldsymbol{z}}}$, would be smaller than VaR calculated with probability vectors without truncating, $\boldsymbol{p}_n^{\boldsymbol{z}}$. Therefore, VaR calculated with truncated probability vectors would underestimate the risk.

5.2.4.3 Aliasing Reduction

Theorem 5.7 shows that the truncated N-fold circular convolution may incur aliasing errors. One way to reduce the aliasing errors is to apply an exponential window to decrease the value of the elements at the end of each vector $\overline{\boldsymbol{p}_n^{\boldsymbol{z}}}$ and to recover the values after the N-fold circular convolution is completed. The exponential window was originally applied in the field of signal processing [26, 47], and Schaller and Temnov [53] used this idea in the loss aggregation problem in insurance.

To begin, consider a sequence of vectors, each of length K, $\{\boldsymbol{x}_n\}_{n=0}^{N-1}$. We construct a sequence of transformed vectors $\{\boldsymbol{x}_{n,\tau}\}_{n=0}^{N-1}$ by applying an exponential window to each $\boldsymbol{x}_n$:

$$x_{n,\tau}[k] = e^{-k/\tau} x_n[k] \tag{5.2.35}$$

with $\tau > 0$. The following theorem shows how to recover the N-fold linear convolution of $\boldsymbol{x}_n$ from the N-fold linear convolution of $\boldsymbol{x}_{n,\tau}$.

Theorem 5.8. *Let $N \geq 2$, $\{\boldsymbol{x}_n\}_{n=0}^{N-1}$ be a sequence of vectors, each of length K, and $\{\boldsymbol{x}_{n,\tau}\}_{n=0}^{N-1}$ be the associated sequence of transformed vectors satisfying* (5.2.35)*. Then*

$$\left(\circledast_{n=0}^{N-1} \boldsymbol{x}_n \right)[k] = e^{k/\tau} \left(\circledast_{n=0}^{N-1} \boldsymbol{x}_{n,\tau} \right)[k] \tag{5.2.36}$$

for $k = 0, \ldots, K-1$.

Proof. We prove this theorem by induction. For the base case, let $N = 2$. We need to prove

$$e^{k/\tau} \boldsymbol{x}_{0,\tau} * \boldsymbol{x}_{1,\tau}[k] = \boldsymbol{x}_0 * \boldsymbol{x}_1[k].$$

By Lemma D.1 in Appendix D, we have

$$
\begin{aligned}
&e^{k/\tau}\boldsymbol{x}_{0,\tau} * \boldsymbol{x}_{1,\tau}[k] \\
&= \begin{cases} e^{k/\tau}\left(\displaystyle\sum_{j=0}^{k} x_{0,\tau}[j]x_{1,\tau}[k-j]\right), & k = 0,\dots,K-1, \\ e^{k/\tau}\left(\displaystyle\sum_{j=k-(K-1)}^{K-1} x_{0,\tau}[j]x_{1,\tau}[k-j]\right), & k = K,\dots,2(K-1), \end{cases} \\
&= \begin{cases} e^{k/\tau}\left(\displaystyle\sum_{j=0}^{k} e^{-j/\tau}x_0[j]e^{-(k-j)/\tau}x_1[k-j]\right), & k = 0,\dots,K-1, \\ e^{k/\tau}\left(\displaystyle\sum_{j=k-(K-1)}^{K-1} e^{-j/\tau}x_0[j]e^{-(k-j)/\tau}x_1[k-j]\right), & k = K,\dots,2(K-1), \end{cases} \\
&= \begin{cases} \displaystyle\sum_{j=0}^{k} x_0[j]x_1[k-j], & k = 0,\dots,K-1, \\ \displaystyle\sum_{j=k-(K-1)}^{K-1} x_0[j]x_1[k-j], & k = K,\dots,2(K-1), \end{cases} \\
&= \boldsymbol{x}_0 * \boldsymbol{x}_1[k].
\end{aligned}
$$

Therefore, (5.2.36) is true for $N = 2$. For the induction step, assume (5.2.36) is true for some $N \geq 2$. Then

$$
e^{k/\tau}\left(\circledast_{n=0}^{(N+1)-1} \boldsymbol{x}_{n,\tau}\right)[k] = e^{k/\tau}\left(\left(\circledast_{n=0}^{N-1} \boldsymbol{x}_{n,\tau}\right) * \boldsymbol{x}_{N,\tau}\right)[k].
$$

Applying Lemma D.1 again, we have

$$
e^{k/\tau}\left(\circledast_{n=0}^{(N+1)-1} \boldsymbol{x}_{n,\tau}\right)[k] = e^{k/\tau}\left(\sum_{j=0}^{k}\left(\circledast_{n=0}^{N-1} \boldsymbol{x}_{n,\tau}\right)[j]x_{N,\tau}[k-j]\right),
$$

for $k = 0,\dots,K-1$,

$$
e^{k/\tau}\left(\circledast_{n=0}^{(N+1)-1} \boldsymbol{x}_{n,\tau}\right)[k] = e^{k/\tau}\left(\sum_{j=k-(N(K-1)-1)}^{k}\left(\circledast_{n=0}^{N-1} \boldsymbol{x}_{n,\tau}\right)[j]x_{N,\tau}[k-j]\right),
$$

for $k = K,\dots,N(K-1)$, and

$$
e^{k/\tau}\left(\circledast_{n=0}^{(N+1)-1} \boldsymbol{x}_{n,\tau}\right)[k] = e^{k/\tau}\left(\sum_{j=k-(N(K-1)-1)}^{K-1}\left(\circledast_{n=0}^{N-1} \boldsymbol{x}_{n,\tau}\right)[j]x_{N,\tau}[k-j]\right),
$$

for $k = N(K-1)+1, \ldots, (N+1)(K-1)$. Since (5.2.36) is true for N,

$$\left(\overset{N-1}{\underset{n=0}{\circledast}} \boldsymbol{x}_{n,\tau}\right)[k] = e^{-j/\tau}\left(\overset{N-1}{\underset{n=0}{\circledast}} \boldsymbol{x}_{n}\right)[k].$$

Thus,

$$\begin{aligned} e^{k/\tau}\left(\overset{(N+1)-1}{\underset{n=0}{\circledast}} \boldsymbol{x}_{n,\tau}\right)[k] &= e^{k/\tau}\left(\sum_{j=0}^{k} e^{-j/\tau}\left(\overset{N-1}{\underset{n=0}{\circledast}} \boldsymbol{x}_{n}\right)[j] e^{-(k-j)/\tau} x_N[k-j]\right), \\ &= \sum_{j=0}^{k}\left(\overset{N-1}{\underset{n=0}{\circledast}} \boldsymbol{x}_{n}\right)[j] x_N[k-j], \end{aligned}$$

for $k = 0, \ldots, K-1$,

$$\begin{aligned} e^{k/\tau}\left(\overset{(N+1)-1}{\underset{n=0}{\circledast}} \boldsymbol{x}_{n,\tau}\right)[k] &= e^{k/\tau}\left(\sum_{j=k-(N(K-1)-1)}^{k} e^{-j/\tau}\left(\overset{N-1}{\underset{n=0}{\circledast}} \boldsymbol{x}_{n}\right)[j] e^{-(k-j)/\tau} x_N[k-j]\right), \\ &= \sum_{j=k-(N(K-1)-1)}^{k}\left(\overset{N-1}{\underset{n=0}{\circledast}} \boldsymbol{x}_{n}\right)[j] x_N[k-j], \end{aligned}$$

for $k = K, \ldots, N(K-1)$, and

$$\begin{aligned} e^{k/\tau}\left(\overset{(N+1)-1}{\underset{n=0}{\circledast}} \boldsymbol{x}_{n,\tau}\right)[k] &= e^{k/\tau}\left(\sum_{j=k-(N(K-1)-1)}^{K-1} e^{-j/\tau}\left(\overset{N-1}{\underset{n=0}{\circledast}} \boldsymbol{x}_{n}\right)[j] e^{-(k-j)/\tau} x_N[k-j]\right), \\ &= \sum_{j=k-(N(K-1)-1)}^{K-1}\left(\overset{N-1}{\underset{n=0}{\circledast}} \boldsymbol{x}_{n}\right)[j] x_N[k-j], \end{aligned}$$

for $k = N(K-1)+1, \ldots, (N+1)(K-1)$. Hence,

$$e^{k/\tau}\left(\overset{(N+1)-1}{\underset{n=0}{\circledast}} \boldsymbol{x}_{n,\tau}\right)[k] = \left(\overset{N-1}{\underset{n=0}{\circledast}} \boldsymbol{x}_{n} * \boldsymbol{x}_N\right)[k] = \left(\overset{(N+1)-1}{\underset{n=0}{\circledast}} \boldsymbol{x}_{n}\right)[k]$$

for $k = 0, \ldots, (N+1)(K-1)$. Therefore, (5.2.36) is true for $N+1$. This completes the proof. □

Combining Theorem 5.4 and Theorem 5.8, we obtain the following corollary.

Corollary 5.9. *Let $N \geq 2$, $\{\overline{\boldsymbol{p}_n^{\boldsymbol{z}}}\}_{n=0}^{N-1}$ be a sequence of truncated vectors, each of length $\overline{K}$ defined in Theorem 5.7. Define a sequence of transformed vectors $\{\overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}}\}_{n=0}^{N-1}$ by*

$$\overline{p_{n,\tau}^{\boldsymbol{z}}}[k] = e^{-k/\tau} p_{n,\tau}^{\boldsymbol{z}}[k]$$

for $k = 0, \ldots, \overline{K} - 1$. *Then*

$$\mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} = l_k\right\} = e^{k/\tau} \left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}} \right)[k] - \delta_{k,\tau}^{a}$$

where

$$\begin{aligned} \delta_{k,\tau}^{a} &= \sum_{u=1}^{U_k^{N-1}} e^{-u\overline{K}/\tau}\, \mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} = l_{u\overline{K}+k}\right\}, \\ U_k^{N-1} &= \max\left\{ u \in \mathbb{N} \,\middle|\, 0 \le u\overline{K} + k \le N(\overline{K} - 1) \right\}. \end{aligned} \tag{5.2.37}$$

Proof. To begin, by Theorem 5.8, we have

$$\left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_{n}^{\boldsymbol{z}}} \right)[k] = e^{k/\tau} \left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}} \right)[k]. \tag{5.2.38}$$

In addition, Theorem 5.4 shows that

$$\left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}} \right)[k] = \left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}} \right)[k] - \sum_{u=1}^{U_k^{N-1}} \left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}} \right)[u\overline{K} + k]. \tag{5.2.39}$$

Substituting (5.2.39) into (5.2.38), we obtain

$$\left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_{n}^{\boldsymbol{z}}} \right)[k] = e^{k/\tau} \left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}} \right)[k] - \sum_{u=1}^{U_k^{N-1}} e^{k/\tau} \left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}} \right)[u\overline{K} + k].$$

Applying Theorem 5.8 again to $\left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}} \right)[u\overline{K} + k]$, we have

$$\left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}} \right)[u\overline{K} + k] = e^{-(u\overline{K}+k)/\tau} \left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_{n}^{\boldsymbol{z}}} \right)[u\overline{K} + k],$$

which leads to

$$\begin{aligned} \left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_{n}^{\boldsymbol{z}}} \right)[k] &= e^{k/\tau} \left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}} \right)[k] - \sum_{u=1}^{U_k^{N-1}} e^{k/\tau} e^{-(u\overline{K}+k)/\tau} \left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_{n}^{\boldsymbol{z}}} \right)[u\overline{K} + k] \\ &= e^{k/\tau} \left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}} \right)[k] - \sum_{u=1}^{U_k^{N-1}} e^{-u\overline{K}/\tau} \left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_{n}^{\boldsymbol{z}}} \right)[u\overline{K} + k]. \end{aligned} \tag{5.2.40}$$

As discussed in the proof of Theorem 5.7, we know

$$\mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} = l_k\right\} = \left(\circledast_{n=0}^{N-1} \overline{\boldsymbol{p}_n^{\boldsymbol{z}}}\right)[k], \tag{5.2.41}$$

for $k = 0, \ldots, \overline{K} - 1$. Therefore, substituting (5.2.41) into (5.2.40) gives

$$\mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} = l_k\right\} = e^{k/\tau}\left(\circledast_{n=0}^{N-1} \overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}}\right)[k] - \sum_{u=1}^{U_k^{N-1}} e^{-u\overline{K}/\tau}\mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} = l_{u\overline{K}+k}\right\}$$

for $k = 0, \ldots, \overline{K} - 1$, which concludes the proof. □

Corollary 5.9 suggests that we can approximate the conditional portfolio loss probabilities by

$$\mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} = l_k\right\} \approx e^{k/\tau}\left(\circledast_{n=0}^{N-1} \overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}}\right)[k]. \tag{5.2.42}$$

As we discussed in 5.2.4.2, VaR calculated by (5.2.42) would underestimate the risk as well. However, comparing the aliasing errors $\delta_{k,\tau}^a$ in (5.2.37) and δ_k^a in (5.2.30), it is obvious that $\delta_{k,\tau}^a < \delta_k^a$ since $\tau > 0$. Therefore, the truncated approximation with the exponential window in (5.2.42) reduces the aliasing errors incurred in the truncated approximation in (5.2.29).

5.2.4.4 Optimal τ

Formula (5.2.37) for $\delta_{k,\tau}^a$ shows that $\delta_{k,\tau}^a$ is a strictly increasing function of the parameter τ. Thus the aliasing error increases as τ increases, which suggests that we should make τ as small as possible. However, the aliasing error is not the only source of errors incurred in the approximation (5.2.42). The exponential window makes the value of the elements in the tail of the vector $\overline{\boldsymbol{p}_n^{\boldsymbol{z}}}$ decrease exponentially to zero. Notice that the elements in the tail of $\overline{\boldsymbol{p}_n^{\boldsymbol{z}}}$ are probabilities of big losses from the obligor n, which are already very small. When we compute (5.2.42), we need to handle extremely small numbers. Due to the finite precision of the computer, rounding errors play a very significant role when operating with extremely small numbers.

We give an example to illustrate the importance of the rounding errors in the approximation (5.2.42). Consider $K = 2^{11} = 2048$, and, for all $n = 0, \ldots, 99$, let $p_n^{\boldsymbol{z}}[0] = 0.9$, $p_n^{\boldsymbol{z}}[8] = 0.05$, $p_n^{\boldsymbol{z}}[16] = 0.03$, $p_n^{\boldsymbol{z}}[24] = 0.02$, and $p_n^{\boldsymbol{z}}[k] = 0$, for all other $k = 0, \ldots, K - 1$. Let $\overline{K} = 2^8 = 256$, and truncate $\boldsymbol{p}_n^{\boldsymbol{z}}$ at $\overline{K} - 1$ to obtain $\overline{\boldsymbol{p}_n^{\boldsymbol{z}}}$, where $\mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} \leq l_{\overline{K}-1}\right\} \approx 0.9897$. Then we compute the following:

1. Apply the sparse convolution method using double-precision floating-point arithmetic to compute

the N-fold linear convolution $\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_n^{\boldsymbol{z}}}[k]$ for $k = 0, \ldots, K-1$, denoting the results by $\zeta\left(\left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_n^{\boldsymbol{z}}}\right)[k]\right)$;

2. Apply the sparse FFT method using single-precision[15] floating-point arithmetic to compute the truncated N-fold circular convolution without exponential windowing, $\left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_n^{\boldsymbol{z}}}\right)[k]$, and the truncated N-fold circular convolution with exponential windowing, $e^{k/\tau}\left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}}\right)[k]$ for $k = 0, \ldots, \overline{K}-1$ and $\tau = 20,\ 30,\ 40,\ 50$ and 60, denoting the results by $\zeta\left(\left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_n^{\boldsymbol{z}}}\right)[k]\right)$ and $\zeta\left(e^{k/\tau}\left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}}\right)[k]\right)$, respectively;

3. Treat $\zeta\left(\overset{N-1}{\underset{n=0}{\circledast}} \boldsymbol{p}_n^{\boldsymbol{z}}[k]\right)$ as the benchmark for $\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} = l_k\}$, and approximate the error incurred in the approximation (5.2.29) and in the approximation (5.2.42) for different values of τ by

$$\delta_k = \left|\zeta\left(\left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_n^{\boldsymbol{z}}}\right)[k]\right) - \zeta\left(\left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_n^{\boldsymbol{z}}}\right)[k]\right)\right|,$$
$$\delta_{k,\tau} = \left|\zeta\left(\left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_n^{\boldsymbol{z}}}\right)[k]\right) - \zeta\left(e^{k/\tau}\left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}}\right)[k]\right)\right|;$$

4. Compute the theoretical aliasing error by

$$\delta_k^a = \sum_{u=1}^{U_k^{N-1}} \mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} = l_{u\overline{K}+k}\right\} \approx \sum_{u=1}^{U_k^{N-1}} \zeta\left(\left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_n^{\boldsymbol{z}}}\right)[u\overline{K}+k]\right),$$

$$\delta_{k,\tau}^a = \sum_{u=1}^{U_k^{N-1}} e^{-u\overline{K}/\tau}\mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} = l_{u\overline{K}+k}\right\} \approx \sum_{u=1}^{U_k^{N-1}} e^{-u\overline{K}/\tau}\zeta\left(\left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_n^{\boldsymbol{z}}}\right)[u\overline{K}+k]\right).$$

The numerical results for δ_k^a and $\delta_{k,\tau}^a$ are presented in Figure 5.15a, and the numerical results of δ_k and $\delta_{k,\tau}$ are given in in Figure 5.15b. If the rounding errors were not significant, then $\zeta\left(e^{k/\tau}\left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}}\right)[k]\right)$ would be very close to $e^{k/\tau}\left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}}\right)[k]$. Consequently, by Theorem 5.7 and Corollary 5.9, the total error δ_k and $\delta_{k,\tau}$ should be very close to the aliasing error δ_k^a and $\delta_{k,\tau}^a$ for all k and τ. However, numerical results show that this is not true. Comparing the curves of δ_k^a and $\delta_{k,\tau}^a$ in Figure 5.15a with the curves of δ_k and $\delta_{k,\tau}$ in Figure 5.15b, we see that, for small k, the actual errors δ_k and $\delta_{k,\tau}$ are close to the aliasing errors δ_k^a and $\delta_{k,\tau}^a$, which decrease as k increases, indicating that the rounding errors are not significant for small k. However, as k increases, unlike the aliasing errors δ_k^a and $\delta_{k,\tau}^a$, which decrease, the actual errors δ_k and $\delta_{k,\tau}$ begin to increase, and the actual error $\delta_{k,\tau}$ with small τ starts to increase

[15] By using single precision here, we aim to amplify the impact of rounding errors incurred in the approximations (5.2.29) and (5.2.42), and to ignore the impact of rounding errors in our benchmark, which is computed with double precision.

earlier than that with large τ. This implies that the rounding errors increase with k. We also observe that, for some τ, the errors incurred in computing $\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} = l_k\}$ by the truncated sparse FFT method with an exponential window can be even larger than the errors for the truncated sparse FFT method without an exponential window. This indicates that the reduction in the aliasing errors associated with using the exponential window can be offset by the increase in the rounding errors if an inappropriate τ is used.

Moreover, to compute VaR_α, we need to find the smallest l such that $\mathbb{P}\{\mathcal{L} \le l\} \ge \alpha$, which, in our approach, requires the computation of $\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} \le l\}$, since $\mathbb{P}\{\mathcal{L} \le l\} = \mathbb{E}[\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} \le l\}]$. To compute $\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} \le l_m\}$, $m = 0, \ldots, \overline{K} - 1$, we need to compute the sum

$$\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} \le l_m\} = \sum_{k \le m} \mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} = l_k\}. \tag{5.2.43}$$

The aliasing error in computing $\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} \le l_m\}$ is the sum of the aliasing errors in computing $\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} = l_k\}$: $\Delta_m^a = \sum_{k \le m} \delta_k^a$ and $\Delta_{m.\tau}^a = \sum_{k \le m} \delta_{k,\tau}^a$. There are rounding errors involved in the summation in (5.2.43) as well. For the example outlined above, we also compute the aliasing error and the actual error in computing $\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} \le l_m\}$:

$$\begin{aligned}
\Delta_m^a &\approx \sum_{k \le m} \sum_{u=1}^{U_k^{N-1}} \zeta\left(\left(\circledast_{n=0}^{N-1} \overline{\boldsymbol{p}_n^{\boldsymbol{z}}}\right)[u\bar{K} + k]\right), \\
\Delta_{m.\tau}^a &\approx \sum_{k \le m} \sum_{u=1}^{U_k^{N-1}} e^{-u\overline{K}/\tau} \zeta\left(\left(\circledast_{n=0}^{N-1} \overline{\boldsymbol{p}_n^{\boldsymbol{z}}}\right)[u\bar{K} + k]\right), \\
\Delta_m &= \left|\zeta\left(\sum_{k \le m} \left(\circledast_{n=0}^{N-1} \overline{\boldsymbol{p}_n^{\boldsymbol{z}}}\right)[k]\right) - \zeta\left(\sum_{k \le m} \left(\circledast_{n=0}^{N-1} \overline{\boldsymbol{p}_n^{\boldsymbol{z}}}\right)[k]\right)\right|, \\
\Delta_{m.\tau} &= \left|\zeta\left(\sum_{k \le m} \left(\circledast_{n=0}^{N-1} \overline{\boldsymbol{p}_n^{\boldsymbol{z}}}\right)[k]\right) - \zeta\left(\sum_{k \le m} e^{k/\tau} \left(\circledast_{n=0}^{N-1} \overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}}\right)[k]\right)\right|.
\end{aligned}$$

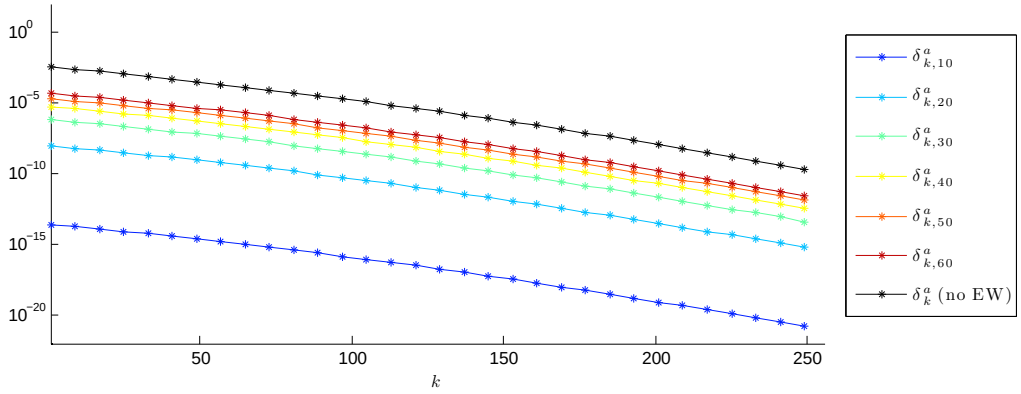
The numerical results are shown in Figure 5.16a and Figure 5.16b. As we can see, when we compute $\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} \le l_m\}$ with the truncated FFT method with exponential windowing, the choice of τ plays a significant role in both aliasing errors and rounding errors. Large τ is ineffective in reducing aliasing errors, while small τ makes rounding errors explode. Therefore, we need to find an appropriate τ to balance aliasing errors and rounding errors.

Denote the rounding error in the computation of $\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} \leq l_m\}$ by

$$\Delta^r_{m,\tau} = \sum_{k \leq m} \left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}^{\boldsymbol{z}}_n} \right)[k] - \zeta \left(\sum_{k \leq m} \left(\overset{N-1}{\underset{n=0}{\circledast}} \overline{\boldsymbol{p}^{\boldsymbol{z}}_n} \right)[k] \right).$$

Figure 5.15: The impact of rounding errors on the computation of $\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} = l_k\}$

(a) Aliasing errors, δ^a_k and $\delta^a_{k,\tau}$



(b) Actual errors, δ_k and $\delta_{k,\tau}$

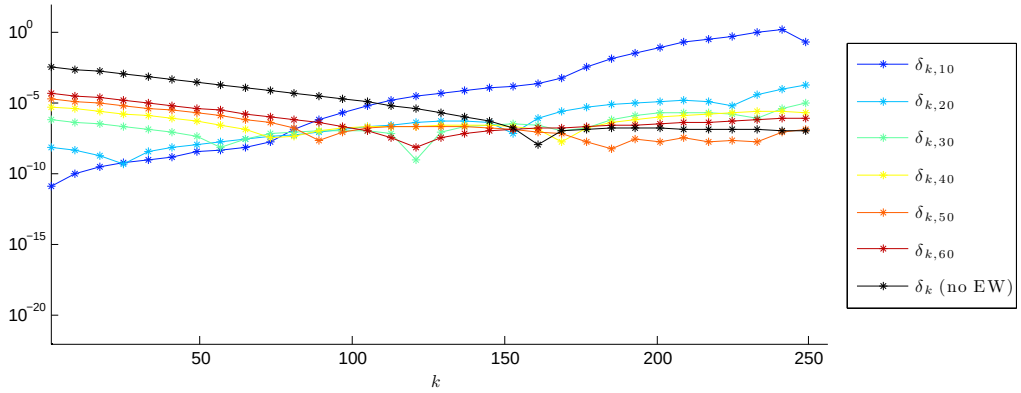
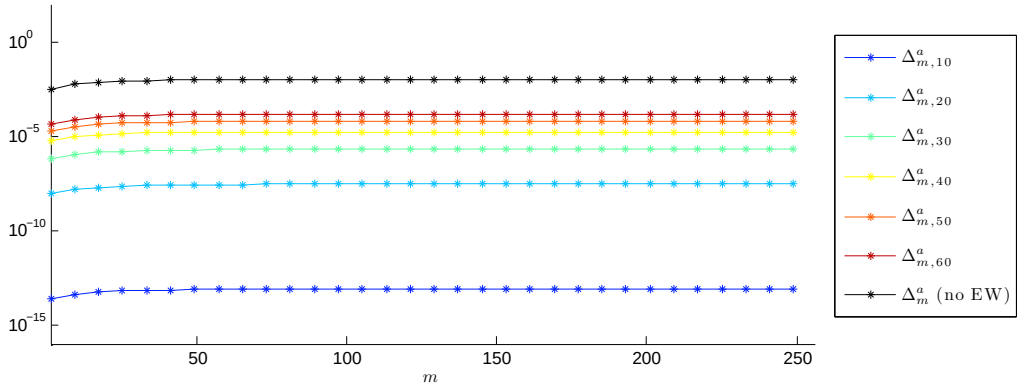
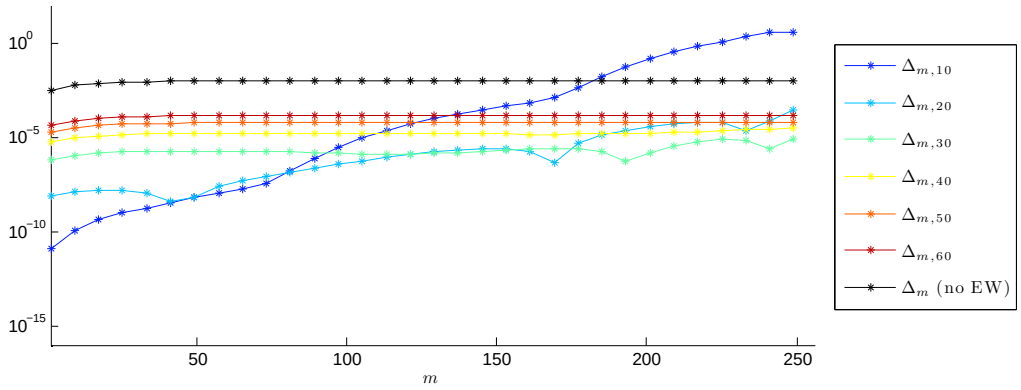


Figure 5.16: The impact of rounding errors on the computation of $\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} \leq l_m\}$

(a) Aliasing errors, Δ_m^a and $\Delta_{m,\tau}^a$



(b) Actual errors, Δ_m and $\Delta_{m,\tau}$



Then

$$\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} \leq l_m\} = \zeta\left(\sum_{k\leq m}\left(\circledast_{n=0}^{N-1}\overline{\boldsymbol{p}_n^{\boldsymbol{z}}}\right)[k]\right) - \Delta_{m,\tau}^a + \Delta_{m,\tau}^r.$$

To obtain the optimal τ, we wish to solve the following optimization problem:

$$\min_{\tau} \quad \left|\Delta_{m,\tau}^r - \Delta_{m,\tau}^a\right|. \tag{5.2.44}$$

Though we know how the aliasing errors behave, it is more difficult to analyze the rounding errors. Most rounding error models ignore the signs of the rounding errors and thereby produce the worst-case bound of the rounding errors. If we consider the rounding errors in every operation, then we get a rounding error bound at each operation. By applying appropriate inequalities to the bound obtained at each operation, we obtain a bound for $\Delta_{m,\tau}^r$. However, this approach is normally too pessimistic, and the

bound for $\Delta^r_{m,\tau}$ is usually much larger than $\Delta^r_{m,\tau}$. Consequently, the optimal τ obtained by minimizing the distance between this bound and $\Delta^a_{m,\tau}$ is not particularly helpful.

Instead, we apply a method suggested in [53] to estimate the optimal τ. For the aliasing error, instead of using formula (5.2.37), we amplify it by

$$\begin{aligned}
\Delta^a_{m,\tau} &= \sum_{k\le m}\sum_{u=1}^{U_k^{N-1}} e^{-u\overline{K}/\tau}\mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} = l_{u\overline{K}+k}\right\} \\
&\le e^{-m/\tau}\sum_{k\le m}\sum_{u=1}^{U_k^{N-1}} \mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} = l_{u\overline{K}+k}\right\} \\
&\le e^{-m/\tau}\mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} > l_m\right\} \\
&= e^{-m/\tau}\left(1-\mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} \le l_m\right\}\right) \\
&\doteq \tilde{\Delta}^a_{m,\tau}, \qquad (5.2.45)
\end{aligned}$$

where $m \le \bar{K}$ is used in the inequalities above.

For the rounding error, due to the finite precision of the floating number system, each element, $\overline{p^{\boldsymbol{z}}_{n,\tau}}[k]$, in $\overline{\boldsymbol{p}^{\boldsymbol{z}}_{n,\tau}}$ is stored in a computer as $fl\left(\overline{p^{\boldsymbol{z}}_{n,\tau}}[k]\right)$, where $fl(x)$ is the nearest floating number to x in a floating point number system. Thus, there is a rounding error

$$\epsilon^{\boldsymbol{z}}_{n,\tau}[k] = fl\left(\overline{p^{\boldsymbol{z}}_{n,\tau}}[k]\right) - \overline{p^{\boldsymbol{z}}_{n,\tau}}[k].$$

In the truncated sparse FFT method with exponential windowing, if we consider only the propagation of rounding errors $\boldsymbol{\epsilon}^{\boldsymbol{z}}_{n,\tau}$, then the total rounding error $\Delta^r_{m,\tau}$ can be approximated by

$$\Delta^r_{m,\tau} \approx \sum_{k\le m} e^{k/\tau}\mathcal{F}^{-1}\left(\prod_{n=0}^{N-1}\mathcal{F}\left(fl\left(\overline{\boldsymbol{p}^{\boldsymbol{z}}_{n,\tau}}\right)\right)\right)[k] - \sum_{k\le m} e^{k/\tau}\mathcal{F}^{-1}\left(\prod_{n=0}^{N-1}\mathcal{F}\left(\overline{\boldsymbol{p}^{\boldsymbol{z}}_{n,\tau}}\right)\right)[k]. \qquad (5.2.46)$$

In any floating point number system, we have $\epsilon^{\boldsymbol{z}}_{n,\tau}[k] = u^{\boldsymbol{z}}_{n,\tau}[k]\overline{p^{\boldsymbol{z}}_{n,\tau}}[k]$ for $k = 0,\dots,\overline{K}-1$, where $u^{\boldsymbol{z}}_{n,\tau}[k] \in [-u,u]$, and u is the unit roundoff in the floating point number system. Therefore,

$$\begin{aligned}
\mathcal{F}\left(fl\left(\overline{\boldsymbol{p}^{\boldsymbol{z}}_{n,\tau}}\right)\right)[k] &= \sum_{j=0}^{\overline{K}-1} fl\left(\overline{p^{\boldsymbol{z}}_{n,\tau}}[j]\right) w^{jk} \\
&= \sum_{j=0}^{\overline{K}-1}\left(p^{\boldsymbol{z}}_{n,\tau}[j] + u^{\boldsymbol{z}}_{n,\tau}[j]p^{\boldsymbol{z}}_{n,\tau}[j]\right) w^{jk} \\
&= \mathcal{F}\left(\overline{\boldsymbol{p}^{\boldsymbol{z}}_{n,\tau}}\right)[k] + \sum_{j=0}^{\overline{K}-1} u^{\boldsymbol{z}}_{n,\tau}[j]p^{\boldsymbol{z}}_{n,\tau}[j]w^{jk}.
\end{aligned}$$

Denote $\mu_{n,\tau}^{\boldsymbol{z}}[k] \doteq \sum_{j=0}^{K-1} u_{n,\tau}^{\boldsymbol{z}}[j] p_{n,\tau}^{\boldsymbol{z}}[j] w^{jk}$, then

$$\mathcal{F}\left(fl\left(\overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}}\right)\right)[k] - \mathcal{F}\left(\overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}}\right)[k] = \mu_{n,\tau}^{\boldsymbol{z}}[k],$$

and

$$\begin{aligned}
\left|\mu_{n,\tau}^{\boldsymbol{z}}[k]\right| &\leq \sum_{j=0}^{\overline{K}-1} \left|u_{n,\tau}^{\boldsymbol{z}}[j]\right| p_{n,\tau}^{\boldsymbol{z}}[j] \left|w^{jk}\right| \\
&= \sum_{j=0}^{\overline{K}-1} \left|u_{n,\tau}^{\boldsymbol{z}}[j]\right| p_{n,\tau}^{\boldsymbol{z}}[j] \\
&\leq u_{n,\tau}^{\boldsymbol{z}}[k] \\
&\leq u.
\end{aligned}$$

Therefore, $|\mu_n^{\boldsymbol{z}}[k]| \sim O(u)$.

Consequently,

$$\begin{aligned}
\prod_{n=0}^{N-1} \mathcal{F}\left(fl\left(\overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}}\right)\right)[k] &= \prod_{n=0}^{N-1} \left(\mathcal{F}\left(\overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}}\right)[k] + \mu_{n,\tau}^{\boldsymbol{z}}[k]\right) \\
&= \prod_{n=0}^{N-1} \mathcal{F}\left(\overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}}\right)[k] + \sum_{n=0}^{N-1} \left(\mu_{n,\tau}^{\boldsymbol{z}}[k] \prod_{v\neq n} \mathcal{F}\left(\overline{\boldsymbol{p}_{v,\tau}^{\boldsymbol{z}}}\right)[k]\right) + O\left(u^2\right). \quad (5.2.47)
\end{aligned}$$

Since

$$\left|\sum_{n=0}^{N-1} \left(\mu_{n,\tau}^{\boldsymbol{z}}[k] \prod_{v\neq n} \mathcal{F}\left(\overline{\boldsymbol{p}_{v,\tau}^{\boldsymbol{z}}}\right)[k]\right)\right| \leq \sum_{n=0}^{N-1} \left(\left|\mu_{n,\tau}^{\boldsymbol{z}}[k]\right| \prod_{v\neq n} \left|\mathcal{F}\left(\overline{\boldsymbol{p}_{v,\tau}^{\boldsymbol{z}}}\right)[k]\right|\right),$$

and

$$\left|\mathcal{F}\left(\overline{\boldsymbol{p}_{v,\tau}^{\boldsymbol{z}}}\right)[k]\right| = \left|\sum_{j=0}^{\overline{K}-1} p_{v,\tau}^{\boldsymbol{z}}[j] w^{jk}\right| \leq \sum_{j=0}^{\overline{K}-1} e^{-j/\tau} p_v^{\boldsymbol{z}}[j] \left|w^{jk}\right| \leq 1,$$

we have

$$\left|\sum_{n=0}^{N-1} \left(\mu_{n,\tau}^{\boldsymbol{z}}[k] \prod_{v\neq n} \mathcal{F}\left(\overline{\boldsymbol{p}_{v,\tau}^{\boldsymbol{z}}}\right)[k]\right)\right| \leq \sum_{n=0}^{N-1} \left|\mu_{n,\tau}^{\boldsymbol{z}}[k]\right| \leq Nu. \quad (5.2.48)$$

Denote $\nu_{n,\tau}^{\boldsymbol{z}}[k] \doteq \sum_{n=0}^{N-1} \left(\mu_{n,\tau}^{\boldsymbol{z}}[k] \prod_{v\neq n} \mathcal{F}\left(\overline{\boldsymbol{p}_{v,\tau}^{\boldsymbol{z}}}\right)[k]\right) + O\left(u^2\right)$, then (5.2.47) and (5.2.48) show that

$$\prod_{n=0}^{N-1} \mathcal{F}\left(fl\left(\overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}}\right)\right)[k] - \prod_{n=0}^{N-1} \mathcal{F}\left(\overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}}\right)[k] = \nu_{n,\tau}^{\boldsymbol{z}}[k]$$

and $\left|\nu^{\boldsymbol{z}}_{n,\tau}[k]\right| \sim O(Nu)$ for $k = 0, \ldots, \overline{K} - 1$.

For the inverse Fourier transform, we could perform the analysis as we did for the Fourier transform. However, as we mentioned, this would give a rather pessimistic estimation. Instead, a statistical error analysis is applied. The basic idea of the statistical error analysis is to assume the rounding errors

$$\delta = fl\,(x) - x$$

are i.i.d. random variables with a uniform distribution on the interval $[-ux, ux]$, and to use the root mean square error (RMSE) as a measure for the rounding error, where RMSE of a random variable $\mathcal{Y}$ is defined by

$$\mathrm{RMSE}(\mathcal{Y}) = \sqrt{\mathbb{E}\left[(\mathcal{Y} - \mathbb{E}\left[\mathcal{Y}\right])^2\right]}.$$

Usually the statistical error analysis gives a more realistic estimate of the rounding errors than does the worst-case error bound. One can find more details about the statistical error analysis in [31, 32, 33, 34, 39].

In our problem, since $\nu^{\boldsymbol{z}}_{n,\tau}[k] \sim O(Nu)$ for $k = 0, \ldots, \overline{K} - 1$, we assume $\mathrm{Re}\left(\nu^{\boldsymbol{z}}_{n,\tau}[k]\right)$ and $\mathrm{Im}\left(\nu^{\boldsymbol{z}}_{n,\tau}[k]\right)$ are i.i.d. random variables with a uniform distribution on the interval $[-Nu, Nu]$.[16] Hence,

$$\begin{aligned}
\mathbb{E}\left[\mathrm{Re}\left(\nu^{\boldsymbol{z}}_{n,\tau}[k]\right)\right] &= \mathbb{E}\left[\mathrm{Im}\left(\nu^{\boldsymbol{z}}_{n,\tau}[k]\right)\right] = 0, \\
\mathbb{V}\left[\mathrm{Re}\left(\nu^{\boldsymbol{z}}_{n,\tau}[k]\right)\right] &= \mathbb{V}\left[\mathrm{Im}\left(\nu^{\boldsymbol{z}}_{n,\tau}[k]\right)\right] = \frac{N^2u^2}{3}.
\end{aligned}$$

Note that

$$\begin{aligned}
\mathcal{F}^{-1}\left(\prod_{n=0}^{N-1} \mathcal{F}\left(fl\left(\overline{\boldsymbol{p}^{\boldsymbol{z}}_{n,\tau}}\right)\right)\right)[k] &= \frac{1}{\overline{K}} \sum_{j=0}^{\overline{K}-1} \left(\prod_{n=0}^{N-1} \mathcal{F}\left(fl\left(\overline{\boldsymbol{p}^{\boldsymbol{z}}_{n,\tau}}\right)\right)[j]\right) w^{-jk} \\
&= \frac{1}{\overline{K}} \sum_{j=0}^{\overline{K}-1} \left(\prod_{n=0}^{N-1} \mathcal{F}\left(\overline{\boldsymbol{p}^{\boldsymbol{z}}_{n,\tau}}\right)[j] + \nu^{\boldsymbol{z}}_{n,\tau}[j]\right) w^{-jk} \\
&= \mathcal{F}^{-1}\left(\prod_{n=0}^{N-1} \mathcal{F}\left(\overline{\boldsymbol{p}^{\boldsymbol{z}}_{n,\tau}}\right)[k]\right) + \frac{1}{\overline{K}} \sum_{j=0}^{\overline{K}-1} \nu^{\boldsymbol{z}}_{n,\tau}[j] w^{-jk}.
\end{aligned}$$

[16] For a complex number x, $\mathrm{Re}(x)$ and $\mathrm{Im}(x)$ are the real part and imaginary part of x, respectively.

Denote $\xi_\tau^{\boldsymbol{z}}[k] \doteq \frac{1}{\overline{K}} \sum_{j=0}^{\overline{K}-1} \nu_{n,\tau}^{\boldsymbol{z}}[j] w^{-jk}$, then

$$\begin{aligned}
\mathbb{E}\left[\mathrm{Re}\left(\xi_\tau^{\boldsymbol{z}}[k]\right)\right] &= \mathbb{E}\left[\frac{1}{\overline{K}} \sum_{j=0}^{\overline{K}-1} \left(\mathrm{Re}\left(\nu_{n,\tau}^{\boldsymbol{z}}[j]\right) \cos\left(\frac{-2\pi}{\bar{K}}(-jk)\right) - \mathrm{Im}\left(\nu_{n,\tau}^{\boldsymbol{z}}[j]\right) \sin\left(\frac{-2\pi}{\bar{K}}(-jk)\right)\right)\right] \\
&= \frac{1}{\overline{K}} \sum_{j=0}^{\overline{K}-1} \left(\mathbb{E}\left[\mathrm{Re}\left(\nu_{n,\tau}^{\boldsymbol{z}}[j]\right)\right] \cos\left(\frac{-2\pi}{\bar{K}}(-jk)\right) - \mathbb{E}\left[\mathrm{Im}\left(\nu_{n,\tau}^{\boldsymbol{z}}[j]\right)\right] \sin\left(\frac{-2\pi}{\bar{K}}(-jk)\right)\right) \\
&= 0,
\end{aligned}$$

since $\mathbb{E}\left[\mathrm{Re}\left(\nu_{n,\tau}^{\boldsymbol{z}}[k]\right)\right] = \mathbb{E}\left[\mathrm{Im}\left(\nu_{n,\tau}^{\boldsymbol{z}}[k]\right)\right] = 0$, for $k = 0, \ldots, \bar{K} - 1$. Moreover,

$$\begin{aligned}
\mathbb{V}\left[\mathrm{Re}\left(\xi_\tau^{\boldsymbol{z}}[k]\right)\right] &= \frac{1}{\overline{K}^2} \sum_{j=0}^{\overline{K}-1} \left(\mathbb{V}\left[\mathrm{Re}\left(\nu_{n,\tau}^{\boldsymbol{z}}[j]\right)\right] \cos^2\left(\frac{-2\pi}{\bar{K}}(-jk)\right) + \mathbb{V}\left[\mathrm{Im}\left(\nu_{n,\tau}^{\boldsymbol{z}}[j]\right)\right] \sin^2\left(\frac{-2\pi}{\bar{K}}(-jk)\right)\right) \\
&= \frac{1}{\overline{K}^2} \sum_{j=0}^{\overline{K}-1} \frac{N^2u^2}{3} \\
&= \frac{N^2u^2}{3\overline{K}},
\end{aligned}$$

since $\mathrm{Re}\left(\nu_{n,\tau}^{\boldsymbol{z}}[k]\right)$ and $\mathrm{Im}\left(\nu_{n,\tau}^{\boldsymbol{z}}[k]\right)$ are independent and $\mathbb{V}\left[\mathrm{Re}\left(\nu_{n,\tau}^{\boldsymbol{z}}[k]\right)\right] = \mathbb{V}\left[\mathrm{Im}\left(\nu_{n,\tau}^{\boldsymbol{z}}[k]\right)\right] = \frac{N^2u^2}{3}$. Therefore, the RMSE of $\mathrm{Re}\left(\xi_\tau^{\boldsymbol{z}}[k]\right)$ is

$$\begin{aligned}
\mathrm{RMSE}\left(\mathrm{Re}\left(\xi_\tau^{\boldsymbol{z}}[k]\right)\right) &= \sqrt{\mathbb{E}\left[\left(\mathrm{Re}\left(\xi_\tau^{\boldsymbol{z}}[k]\right) - \mathbb{E}\left[\mathrm{Re}\left(\xi_\tau^{\boldsymbol{z}}[k]\right)\right]\right)^2\right]} \\
&= \sqrt{\mathbb{V}\left[\mathrm{Re}\left(\xi_\tau^{\boldsymbol{z}}[k]\right)\right]} \\
&= \frac{Nu}{\sqrt{3\overline{K}}}.
\end{aligned}$$

Similarly, $\mathrm{RMSE}\left(\mathrm{Im}\left(\xi_\tau^{\boldsymbol{z}}[k]\right)\right) = Nu/\sqrt{3\overline{K}}$. Therefore,

$$\mathcal{F}^{-1}\left(\prod_{n=0}^{N-1} \mathcal{F}\left(fl\left(\overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}}\right)\right)\right)[k] - \mathcal{F}^{-1}\left(\prod_{n=0}^{N-1} \mathcal{F}\left(\overline{\boldsymbol{p}_{n,\tau}^{\boldsymbol{z}}}\right)[k]\right) = \xi_\tau^{\boldsymbol{z}}[k], \tag{5.2.49}$$

where $|\xi_\tau^{\boldsymbol{z}}[k]| \sim O\left(Nu/\sqrt{\overline{K}}\right)$.

Finally, by (5.2.46) and (5.2.49), we have

$$\Delta_{m,\tau}^r \approx \sum_{k=0}^{m} e^{k/\tau} \xi_\tau^{\boldsymbol{z}}[k].$$

Again, assuming $\mathrm{Re}\left(\xi_\tau^{\boldsymbol{z}}[k]\right)$ and $\mathrm{Im}\left(\xi_\tau^{\boldsymbol{z}}[k]\right)$ are i.i.d. random variables with a uniform distribution on the

interval $\left[-Nu/\sqrt{\overline{K}}, Nu/\sqrt{\overline{K}}\right]$, we have

$$\begin{aligned}\mathbb{E}\left[\mathrm{Re}\left(\xi_\tau^{\boldsymbol{z}}[k]\right)\right] &= \mathbb{E}\left[\mathrm{Im}\left(\xi_\tau^{\boldsymbol{z}}[k]\right)\right] = 0,\\ \mathbb{V}\left[\mathrm{Re}\left(\xi_\tau^{\boldsymbol{z}}[k]\right)\right] &= \mathbb{V}\left[\mathrm{Im}\left(\xi_\tau^{\boldsymbol{z}}[k]\right)\right] = \frac{N^2u^2}{3\overline{K}}.\end{aligned}$$

By similar arguments, we have

$$\begin{aligned}\mathbb{E}\left[\mathrm{Re}\left(\Delta_{m,\tau}^r\right)\right] &= \sum_{k=0}^{m} e^{k/\tau}\mathbb{E}\left[\mathrm{Re}\left(\xi_\tau^{\boldsymbol{z}}[k]\right)\right] = 0,\\ \mathbb{E}\left[\mathrm{Im}\left(\Delta_{m,\tau}^r\right)\right] &= \sum_{k=0}^{m} e^{k/\tau}\mathbb{E}\left[\mathrm{Im}\left(\xi_\tau^{\boldsymbol{z}}[k]\right)\right] = 0,\\ \mathbb{V}\left[\mathrm{Re}\left(\Delta_{m,\tau}^r\right)\right] &= \sum_{k=0}^{m} e^{2k/\tau}\mathbb{V}\left[\mathrm{Re}\left(\xi_\tau^{\boldsymbol{z}}[k]\right)\right] = \frac{N^2u^2}{3\overline{K}}\frac{e^{\frac{2(m+1)}{\tau}}-1}{e^{\frac{2}{\tau}}-1},\\ \mathbb{V}\left[\mathrm{Im}\left(\Delta_{m,\tau}^r\right)\right] &= \sum_{k=0}^{m} e^{2k/\tau}\mathbb{V}\left[\mathrm{Im}\left(\xi_\tau^{\boldsymbol{z}}[k]\right)\right] = \frac{N^2u^2}{3\overline{K}}\frac{e^{\frac{2(m+1)}{\tau}}-1}{e^{\frac{2}{\tau}}-1},\end{aligned}$$

and

$$\begin{aligned}\mathrm{RMSE}\left(\mathrm{Re}\left(\Delta_{m,\tau}^r\right)\right) &= \sqrt{\mathbb{V}\left[\mathrm{Re}\left(\Delta_{m,\tau}^r\right)\right]} = \frac{Nu}{\sqrt{3\overline{K}}}\sqrt{\frac{e^{\frac{2(m+1)}{\tau}}-1}{e^{\frac{2}{\tau}}-1}},\\ \mathrm{RMSE}\left(\mathrm{Im}\left(\Delta_{m,\tau}^r\right)\right) &= \sqrt{\mathbb{V}\left[\mathrm{Im}\left(\Delta_{m,\tau}^r\right)\right]} = \frac{Nu}{\sqrt{3\overline{K}}}\sqrt{\frac{e^{e^{2(m+1)/\tau}}-1}{e^{2/\tau}-1}},\end{aligned}$$

which indicates

$$\left|\Delta_{m,\tau}^r\right| \sim O\left(\frac{Nu}{\sqrt{\overline{K}}}\sqrt{\frac{e^{2m/\tau}-1}{e^{2/\tau}-1}}\right).$$

Therefore, we can approximate the solution to the problem (5.2.44) by solving

$$\tilde{\Delta}_{m,\tau}^a = \tilde{\Delta}_{m,\tau}^r,$$

where

$$\tilde{\Delta}_{m,\tau}^r \doteq \frac{Nu}{\sqrt{K}}\sqrt{\frac{e^{2m/\tau}-1}{e^{2/\tau}-1}}.$$

That is,

$$\frac{Nu}{\sqrt{K}}\sqrt{\frac{e^{2m/\tau}-1}{e^{2/\tau}-1}} = e^{-m/\tau}\left(1-\mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} \le l_m\right\}\right). \tag{5.2.50}$$

However, the conditional loss probabilities $\mathbb{P}\left\{\mathcal{L}^{\boldsymbol{z}} \le l_m\right\}$, which is what we aim to compute, is in the

equation (5.2.44). To solve this problem, we can apply the asymptotic approximation based on the central limit theorem to approximate $\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} \leq l_m\}$. Applying the CLT approximation described in Section (4.2), the conditional loss probability can be approximated by

$$\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} \leq l_m\} \approx \Phi\left(\frac{l_m - \mu(\boldsymbol{z})}{\sigma(\boldsymbol{z})}\right),$$

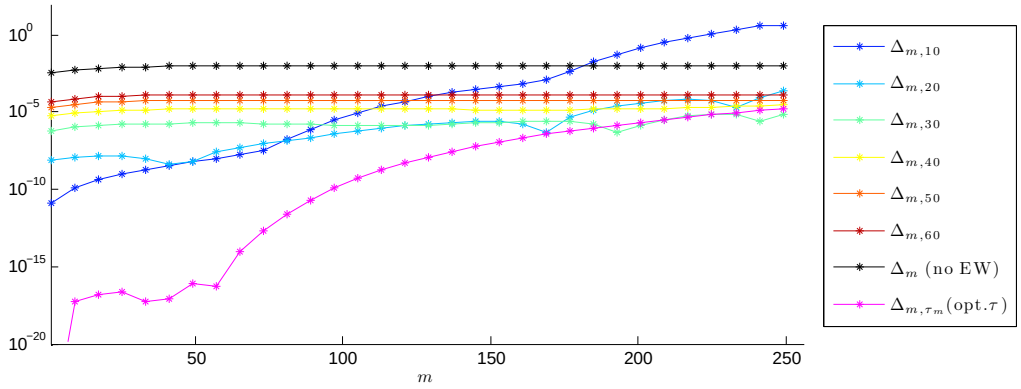
where $\mu(\boldsymbol{z}) = \mathbb{E}[\mathcal{L}^{\boldsymbol{z}}]$ and $\sigma(\boldsymbol{z}) = \sqrt{\mathbb{V}[\mathcal{L}^{\boldsymbol{z}}]}$. Therefore we can solve the following equation to obtain an approximation to the optimal τ.

$$\frac{Nu}{\sqrt{K}}\sqrt{\frac{e^{2m/\tau}-1}{e^{2/\tau}-1}} = e^{-m/\tau}\left(1 - \Phi\left(\frac{l_m - \mu(\boldsymbol{z})}{\sigma(\boldsymbol{z})}\right)\right). \tag{5.2.51}$$

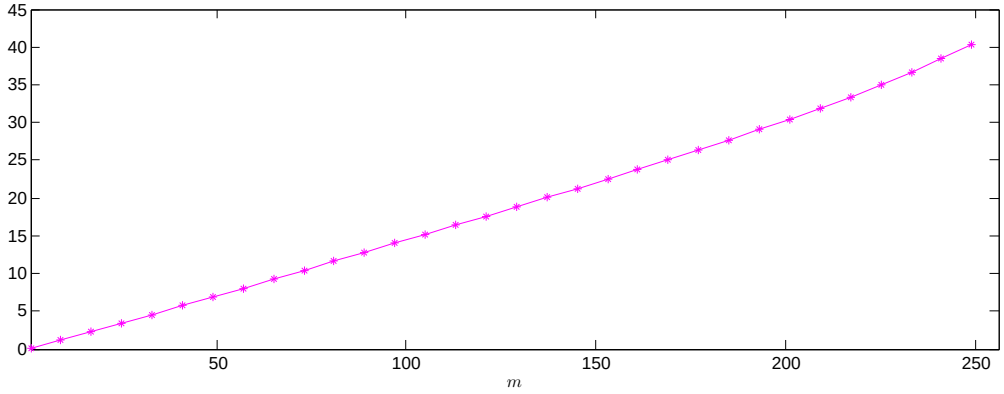
Notice that, the optimal τ obtained by (5.2.51) is dependent on m and $\boldsymbol{z}$. If we need to compute $\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} \leq l_m\}$ for only one quantile l_m, then we need to solve (5.2.51) once to get τ_m. If we need to compute the distribution $\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} \leq l_m\}$ for $m = 0, \ldots, \overline{K} - 1$, then a more accurate approximation to the distribution can be achieved by solving (5.2.51) to obtain τ_m for each m, and applying the truncated sparse FFT method plus exponential windowing with τ_m to compute $\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} \leq l_m\}$. For the example we discussed in the beginning of this subsubsection, we compare the total errors in computing $\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} \leq l_m\}$ for $m = 0, \ldots, \overline{K} - 1$ using different optimal τ_m to that using other values of τ. The results are presented in Figure 5.17a. Figure 5.17b shows the optimal τ_m for different m.

Figure 5.17: Comparison of actual errors in computing $\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} \leq l_m\}$ with different τ

(a) Actual errors, Δ_m and $\Delta_{m,\tau}$



(b) Optimal τ_m



We conclude this section by presenting Algorithm 5.12, which is based on the truncated sparse FFT method plus exponential windowing with optimal τ.

Algorithm 5.12 Truncated sparse FFT method with exponential window to compute $\mathbb{P}\{\mathcal{L}^{\boldsymbol{z}} \leq l_m\}$

Input: $\boldsymbol{p}_n^{\boldsymbol{z}}$, K, t, m;

Output: $p_m^{\boldsymbol{z}}$;

$\bar{t} \leftarrow \lceil \log_2(m) \rceil$; $\overline{K} \leftarrow 2^{\bar{t}}$;
$\overline{\boldsymbol{p}_n^{\boldsymbol{z}}} \leftarrow \boldsymbol{p}_n^{\boldsymbol{z}}(0 : \overline{K} - 1)$;

Compute $\boldsymbol{w}_{long}$ by Algorithm 5.5 with $\bar{t}$;
$\tilde{\boldsymbol{p}} \leftarrow \boldsymbol{p}_0^{\boldsymbol{z}}$;

Compute τ by solving (5.2.51);
for $k = 1 : \overline{K} - 1$ **do**
    $\overline{p_n^{\boldsymbol{z}}[k]} \leftarrow e^{-k/\tau} \overline{p_n^{\boldsymbol{z}}[k]}$;
end for

for $n = 0 : N - 1$ **do**
    $\overline{\boldsymbol{p}_n^{\boldsymbol{z}}} \leftarrow \mathbf{P}_K^T \overline{\boldsymbol{p}_n^{\boldsymbol{z}}}$ by Algorithm 5.9;
    $\texttt{sort}(\hat{\boldsymbol{d}})$;
    for $q = 1 : t$ **do**
        $\overline{\boldsymbol{p}_n^{\boldsymbol{z}}} \leftarrow \mathbf{A}_q \overline{\boldsymbol{p}_n^{\boldsymbol{z}}}$ by Algorithm 5.10;
    end for
    if $n > 0$ **then**
        $\tilde{\boldsymbol{p}} \leftarrow \tilde{\boldsymbol{p}} \odot \overline{\boldsymbol{p}_n^{\boldsymbol{z}}}$;
    end if
end for

$\boldsymbol{w}_{long} \leftarrow \overline{\boldsymbol{w}_{long}}$;
$\tilde{\boldsymbol{p}} \leftarrow \mathbf{P}_K^T \tilde{\boldsymbol{p}}$ by Algorithm 5.4;
for $q = 1 : t$ **do**
    $\tilde{\boldsymbol{p}} \leftarrow \overline{\mathbf{A}_q} \tilde{\boldsymbol{p}}$ by Algorithm 5.6;
end for
$\tilde{\boldsymbol{p}} \leftarrow \tilde{\boldsymbol{p}}/K$;

for $k = 0 : \overline{K} - 1$ **do**
    $\tilde{\boldsymbol{p}}[k] \leftarrow e^{k/\tau} \tilde{\boldsymbol{p}}[k]$;
end for

$p_m^{\boldsymbol{z}} \leftarrow 0$;
for $k = 0 : m$ **do**
    $p_m^{\boldsymbol{z}} \leftarrow p_m^{\boldsymbol{z}} + \tilde{p}[k]$;
end for
return $p_m^{\boldsymbol{z}}$;

5.3 Numerical Results and Comparisons

In this section, we conduct numerical experiments to compare different exact methods. In Subsection 5.3.1, we compare the sparse convolution (SCONV) method and the truncated sparse convolution (TR SCONV) method to the benchmark full convolution (FCONV) method in the best/worse case discussed in Subsection 5.1.2. The purpose of this comparison is to provide numerical support for the speedups we derived in (5.1.38) and (5.1.42), and to check the error estimation in (5.1.50). In Subsection 5.3.2, we compare the sparse FFT method (SFFT) to the benchmark full FFT (FFFT) method in the best/worst case described in Subsection 5.2.3.4 to provide numerical support for the speedups we obtained in (5.2.26) and (5.2.27). As we discussed before, the best/worse case may not be achievable for real portfolios, while we are more interested in the performance of different methods when they are used for real portfolios. Therefore, in Subsection 5.3.3, we build several test portfolios, and compare the efficiency and accuracy of the SCONV method, TR SCONV method, SFFT method, the truncated sparse FFT (TR SFFT) method, and the truncated sparse FFT method with exponential windowing (EW TR SFFT) to those of the benchmark FCONV method. Finally, in Subsection 5.3.4, we compare the efficiency and accuracy of the exact methods to MC method for some synthetic portfolios. All methods are implemented in C++ and the computation was performed on a workstation with a 2.8GHz Intel Core 2 Duo CPU and 6GB 667 MHz DDR2 SDRAM.

5.3.1 Best/Worst Cases: (Truncated) Sparse Convolution Methods

As shown in (5.1.38) and (5.1.42), the speedup of the SCONV method over the FCONV method is $\Omega(\alpha^2)$ in the best case and $\Omega\left(\alpha^2 C^N\right)$ in the worst case. Therefore, we build different test cases to determine the impact of the parameters N, α, and C, respectively, in both best and worst cases.

To provide numerical support for Theorem 5.2 and Corollary 5.3, we let the threshold $Tol = 10^{-16}$, and set ϵ by (5.1.50). Then we compare Tol and the maximum absolute difference of $\tilde{p}^{\mathbf{z}}_{N-1}[k]$ computed by the TR SCONV method and by the FCONV method.

5.3.1.1 Best Case

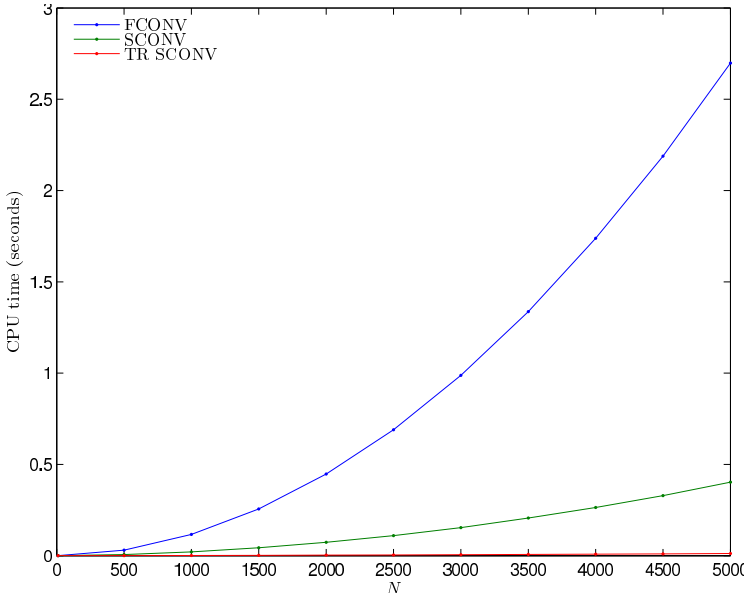
Testing N. The nonzero index vector $\boldsymbol{d}_n$ is constructed by (5.1.28). To test the impact of the parameter N, we set $\alpha = 5$, $C = 2$, $\boldsymbol{q}_n^{\boldsymbol{z}} = [0.99, 0.01]^T$ for $n = 0, \ldots, N-1$, and compute $\tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}}$ for N =10, 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000 by FCONV, SCONV and TR SCONV method. The CPU time, speedup and maximum absolute error in $\tilde{p}_{N-1}^{\boldsymbol{z}}[k]$ are listed in Table 5.3. The CPU time and speedup are given in Figure 5.18.

Table 5.3: Numerical results for testing N in the best case for the SCONV and TR SCONV methods

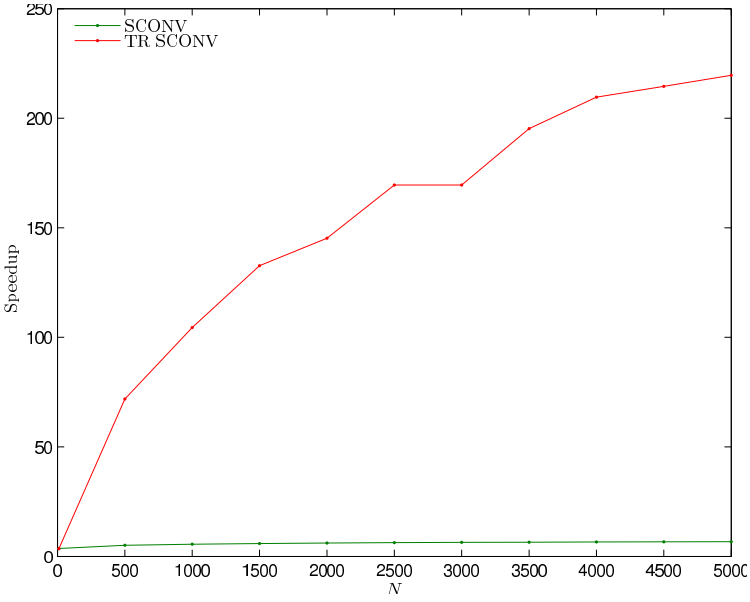
N	FCONV	SCONV		TR SCONV		
	CPU time	CPU time	Speedup	CPU time	Speedup	Max error
10	1.3424E-05	3.7019E-06	3.6262E+00	3.8064E-06	3.5267E+00	9.9000E-19
500	3.0142E-02	5.9117E-03	5.0987E+00	4.1962E-04	7.1832E+01	1.5590E-18
1000	1.1617E-01	2.0850E-02	5.5717E+00	1.1123E-03	1.0444E+02	1.2533E-18
1500	2.5580E-01	4.3483E-02	5.8828E+00	1.9280E-03	1.3268E+02	1.0603E-18
2000	4.4709E-01	7.3202E-02	6.1076E+00	3.0786E-03	1.4523E+02	9.8600E-19
2500	6.8955E-01	1.0992E-01	6.2732E+00	4.0676E-03	1.6952E+02	9.8813E-19
3000	9.8746E-01	1.5380E-01	6.4204E+00	5.8243E-03	1.6954E+02	8.8315E-19
3500	1.3369E+00	2.0607E-01	6.4876E+00	6.8472E-03	1.9525E+02	7.7952E-19
4000	1.7383E+00	2.6373E-01	6.5912E+00	8.2931E-03	2.0961E+02	7.2973E-19
4500	2.1886E+00	3.2870E-01	6.6584E+00	1.0200E-02	2.1457E+02	7.0412E-19
5000	2.6983E+00	4.0255E-01	6.7030E+00	1.2288E-02	2.1959E+02	6.7083E-19

Figure 5.18: CPU time and speedup for testing N in the best case for the SCONV and TR SCONV methods

(a) CPU time



(b) Speedup



The numerical results show that, as N increases, the CPU time required by each convolution method increases. The SCONV method and the TR SCONV method run much faster than the FCONV method, and the TR SCONV method has a much larger speedup than the SCONV method. The speedup of the SCONV method over the FCONV method increases with N for small N, but, when N is larger than 2000, the increase of the speedup is rather slow, with the speedup remaining between 6 and 7. This

implies that, when N is large enough, it has no significant impact on the speedup of the SCONV method. This result agrees with our theoretical speedup in (5.1.38), where there is no N in the expression.

However, for the TR SCONV method, the speedup is much more significant and depends strongly on N. As N increases from 10 to 5000, the speedup of TR SCONV method increases steadily from 3.57 to 219.59. The reason for this is that, when N is large, more extremely small $\tilde{p}_{n-1}^{\boldsymbol{z}}[i]p_n^{\boldsymbol{z}}[j]$ are truncated to zero, which saves computational time.

The last column in Table 5.3 indicates that the maximum absolute error in all cases tested is around $10^{-18} \sim 10^{-19}$, which is less than our pre-defined threshold $Tol = 10^{-16}$. These numerical results support the way we choose ϵ in (5.1.50), which follows from Theorem 5.2 and Corollary 5.3.

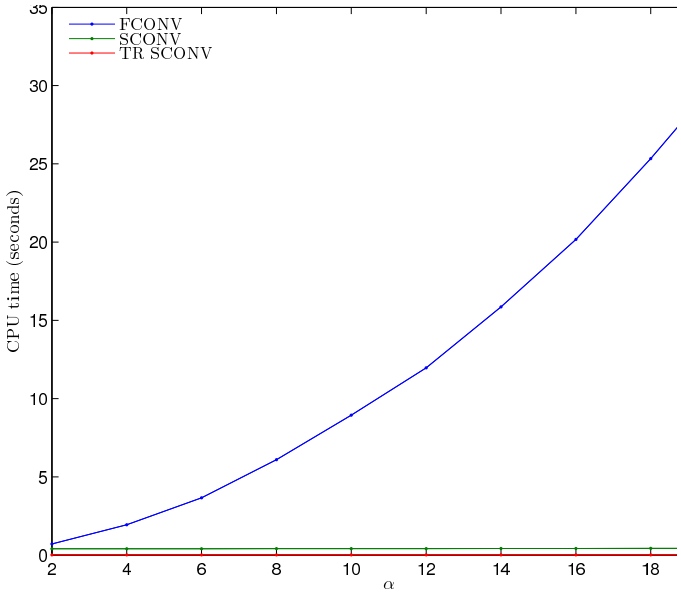
Testing α. The nonzero index vector $\boldsymbol{d}_n$ is constructed by (5.1.28). To test the impact of the parameter α, we set $N = 5000$, $C = 2$, $\boldsymbol{q}_n^{\boldsymbol{z}} = [0.99, 0.01]^T$ for $n = 0, \ldots, N-1$, and compute $\tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}}$ for α =2, 4, 6, 8, 10, 12, 14, 16, 18, 20 by the FCONV, SCONV and TR SCONV methods. The CPU time, speedup and maximum absolute error in $\tilde{p}_{N-1}^{\boldsymbol{z}}[k]$ are listed in Table 5.4. The CPU time and speedup are given in Figure 5.19.

Table 5.4: Numerical results for testing α in the best case for the SCONV and TR SCONV methods

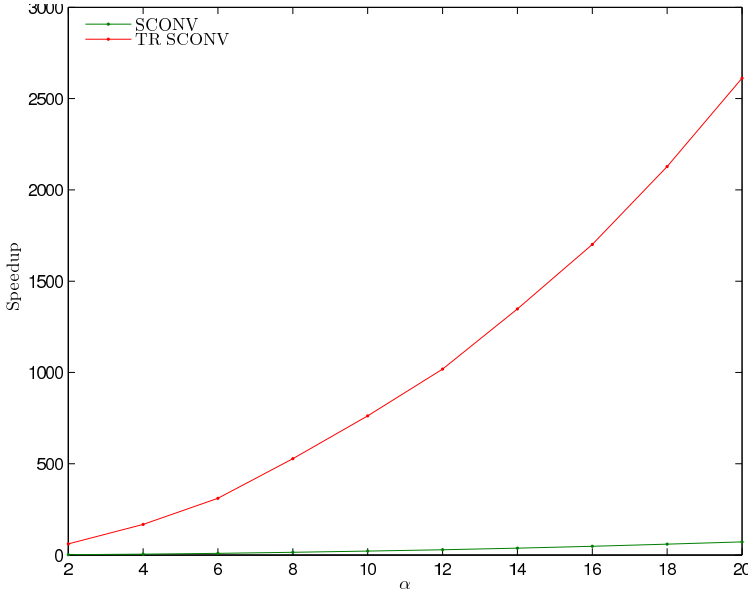
α	FCONV	SCONV		TR SCONV		
	CPU time	CPU time	Speedup	CPU time	Speedup	Max error
2	7.1225E-01	4.0126E-01	1.7750E+00	1.1616E-02	6.1316E+01	6.7084E-19
4	1.9379E+00	4.0220E-01	4.8182E+00	1.1545E-02	1.6786E+02	6.7084E-19
6	3.6616E+00	4.0488E-01	9.0437E+00	1.1773E-02	3.1102E+02	6.7084E-19
8	6.0961E+00	4.1231E-01	1.4785E+01	1.1561E-02	5.2730E+02	6.7084E-19
10	8.9407E+00	4.1099E-01	2.1754E+01	1.1734E-02	7.6195E+02	6.7084E-19
12	1.1965E+01	4.1276E-01	2.8988E+01	1.1745E-02	1.0187E+03	6.7084E-19
14	1.5858E+01	4.2376E-01	3.7422E+01	1.1764E-02	1.3480E+03	6.7084E-19
16	2.0172E+01	4.2474E-01	4.7493E+01	1.1857E-02	1.7013E+03	6.7084E-19
18	2.5335E+01	4.2623E-01	5.9440E+01	1.1907E-02	2.1277E+03	6.7084E-19
20	3.0947E+01	4.3193E-01	7.1648E+01	1.1849E-02	2.6118E+03	6.7084E-19

Figure 5.19: CPU time and speedup for testing α in the best case for the SCONV and TR SCONV methods

(a) CPU time



(b) Speedup



The numerical results show that, as α increases, the CPU time required by all convolution methods increases. The SCONV and TR SCONV methods are much faster than the FCONV method, and the TR SCONV method has a much larger speedup than the SCONV method. The speedup of the SCONV method over the FCONV method increases with α. We performed a linear least squares fit to the

speedup of the SCONV method to

$$f_{best,\alpha}^{SCONV}(\alpha) = p_2\alpha^2,$$
$$\tilde{f}_{best,\alpha}^{SCONV}(\alpha) = p_3\alpha^3 + p_2\alpha^2 + p_1\alpha + p_0.$$

The results are

$$f_{best,\alpha}^{SCONV}(\alpha) = 0.19\alpha^2,$$
$$\tilde{f}_{best,\alpha}^{SCONV}(\alpha) = 8.67\text{E-}04 \cdot \alpha^3 + 0.12 \cdot \alpha^2 + 0.95 \cdot \alpha - 0.75. \tag{5.3.1}$$

Figure 5.20 compares the fitted curves $f_{best,\alpha}^{SCONV}(\alpha)$ and $\tilde{f}_{best,\alpha}^{SCONV}(\alpha)$ to the actual SCONV speedup.

Figure 5.20: Comparison of the fitted curves $f_{best,\alpha}^{SCONV}(\alpha)$ and $\tilde{f}_{best,\alpha}^{SCONV}(\alpha)$ to actual speedup for the SCONV method

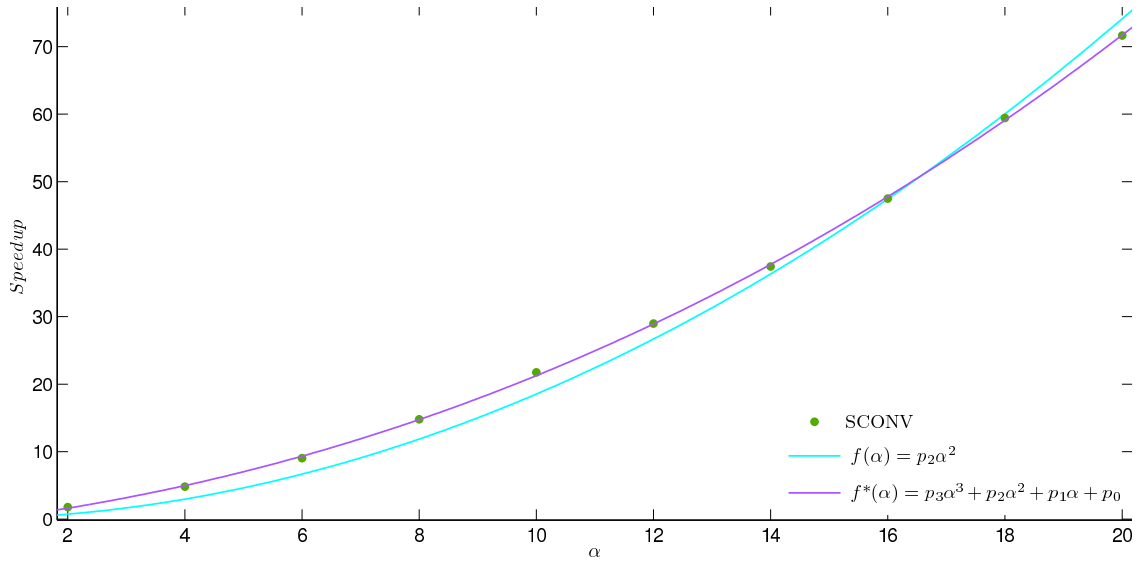


Figure 5.20 shows $\tilde{f}_{best,\alpha}^{SCONV}(\alpha)$ and $f_{best,\alpha}^{SCONV}(\alpha)$ are good fits to the SCONV speedup. Moreover, the coefficient of α^3 in $\tilde{f}_{best,\alpha}^{SCONV}(\alpha)$ is insignificant compared to the coefficient of α^2. Therefore, we conclude that, in the best case, the speedup of the SCONV method grows quadratically in α, which agrees with our theoretical speedup in (5.1.38).

For the TR SCONV method, there is a rapid increase of the speedup as α grows. More specifically, as α increases from 2 to 20, the speedup of the TR SCONV method increases from 61.32 to 2611.82, which shows that α has a significant impact on the speedup of the TR SCONV method.

The last column in Table 5.4 indicates that the maximum absolute error in all cases tested is around 6.71×10^{-19}, which is less than our pre-defined threshold $Tol = 10^{-16}$. These numerical results support

the way we choose ϵ in (5.1.50), which follows from Theorem 5.2 and Corollary 5.3.

Testing C. The nonzero index vector $\boldsymbol{d}_n$ is constructed by (5.1.28). To test the impact of the parameter C, we set $N = 5000$, $\alpha = 5$, and vary C =2, 3, 4, 5, 6, 7, 8, 9, 10. For each C, the nonzero probability vector $\boldsymbol{q}_n^{\boldsymbol{z}}$ is set to

$$q_n^{\boldsymbol{z}}[c] = \begin{cases} 1 - 0.01 \sum_{k=1}^{C-1} k, & c = 0, \\ 0.01(C - c), & c = 1, \ldots, C-1. \end{cases} \tag{5.3.2}$$

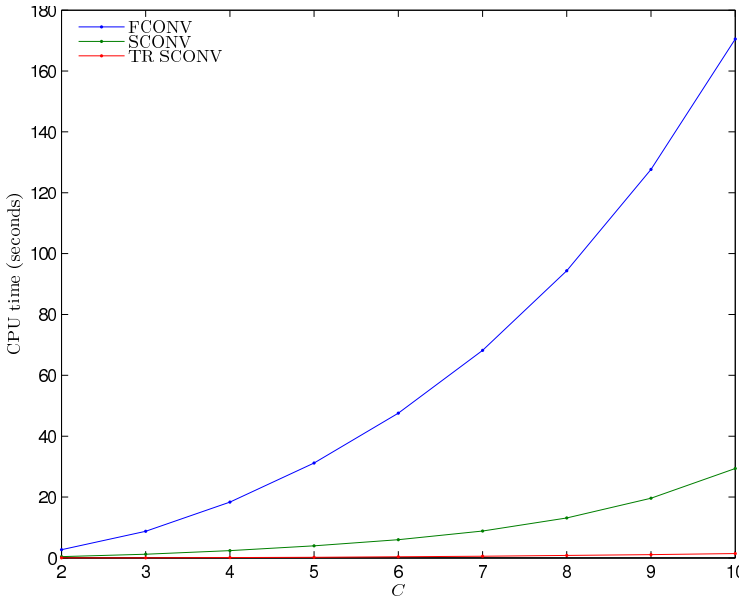
for $n = 0, \ldots, N-1$. For example, we have $\boldsymbol{q}_n^{\boldsymbol{z}} = [0.99, 0.01]^T$ for $C = 2$, and $\boldsymbol{q}_n^{\boldsymbol{z}} = [0.97, 0.02, 0.01]^T$ for $C = 3$. We compute $\tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}}$ for C =2, 3, 4, 5, 6, 7, 8, 9, 10 by the FCONV, SCONV and TR SCONV methods. The CPU time, speedup and maximum absolute errors are listed in Table 5.5. The CPU time and speedup are given in Figure 5.21.

Table 5.5: Numerical results for testing C in the best case for the SCONV and TR SCONV methods

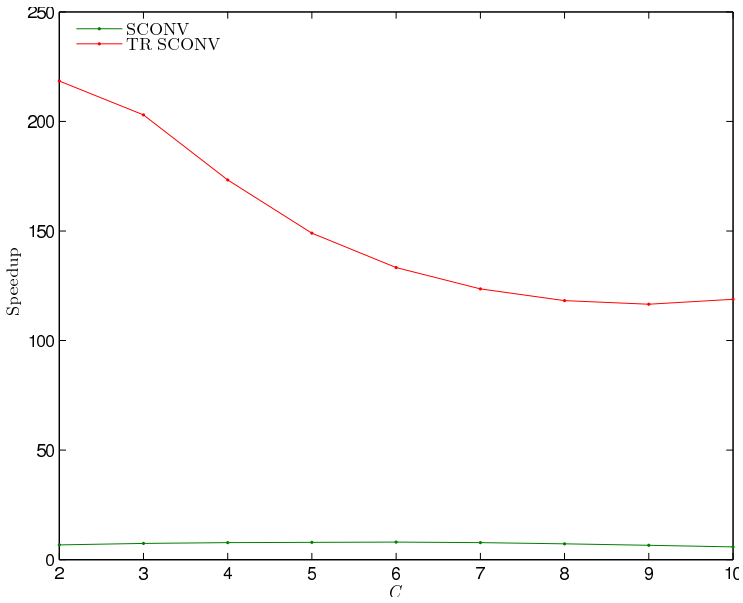
C	FCONV	SCONV		TR SCONV		
	CPU time	CPU time	Speedup	CPU time	Speedup	Max error
2	2.7020E+00	4.0286E-01	6.7070E+00	1.2369E-02	2.1845E+02	6.7083E-19
3	8.7557E+00	1.1851E+00	7.3882E+00	4.3140E-02	2.0296E+02	7.1550E-19
4	1.8338E+01	2.3692E+00	7.7402E+00	1.0581E-01	1.7331E+02	6.5983E-19
5	3.1189E+01	3.9493E+00	7.8973E+00	2.0932E-01	1.4900E+02	5.9852E-19
6	4.7585E+01	5.9644E+00	7.9782E+00	3.5692E-01	1.3332E+02	5.4234E-19
7	6.8201E+01	8.8232E+00	7.7297E+00	5.5200E-01	1.2355E+02	7.3922E-19
8	9.4349E+01	1.3111E+01	7.1962E+00	7.9805E-01	1.1822E+02	4.5567E-19
9	1.2767E+02	1.9604E+01	6.5124E+00	1.0951E+00	1.1658E+02	4.1625E-19
10	1.7050E+02	2.9386E+01	5.8021E+00	1.4350E+00	1.1882E+02	3.8501E-19

Figure 5.21: CPU time and speedup for testing C in the best case for the SCONV and TR SCONV methods

(a) CPU time



(b) Speedup



The numerical results show that, as C increases, the CPU time required by each convolution method increases. The SCONV method and the TR SCONV method run much faster than the FCONV method, and the TR SCONV method has a much larger speedup than the SCONV method. The speedup of the SCONV method over the FCONV method varies in a narrow range between 5.8 and 7.8 as C increases

from 2 to 10. This implies that C has no significant impact on the speedup of the SCONV method. This result agrees with our theoretical speedup in (5.1.28), where there is no C in the expression.

However, for the TR SCONV method, as C increases from 2 to 6, the speedup of the TR SCONV method decreases rapidly from 218.45 to 133.32, while for $C > 7$, the change in the speedup is much smaller (within $[116, 118]$). We believe this is related to the way that we construct the nonzero probability vector $\boldsymbol{q}_n^{\boldsymbol{z}}$.

The last column in Table 5.5 indicates that the maximum absolute error in all test cases is around 10^{-19}, which is less than our pre-defined threshold $Tol = 10^{-16}$. These numerical results support the way we choose ϵ in (5.1.50), which follows from Theorem 5.2 and Corollary 5.3.

5.3.1.2 Worst Case

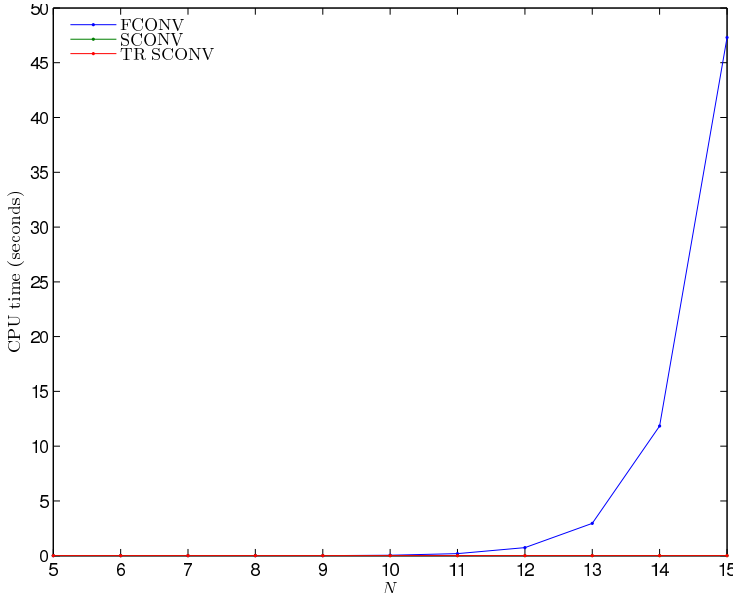
Testing N. The nonzero index vector $\boldsymbol{d}_n$ is constructed by (5.1.32). To test the impact of the parameter N, we set $\alpha = 5$, $C = 2$, $\boldsymbol{q}_n^{\boldsymbol{z}} = [0.999, 0.001]^T$ for $n = 0, \ldots, N-1$, and compute $\tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}}$ for N =5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 by FCONV, SCONV and TR SCONV method. Unlike our testing for the best case, we can not choose very large test parameters N for the worst case, because the length of the resultant vector, as shown in (5.1.40), is too large, so it is impractical to use the benchmark FCONV method. Since N in our tests is not large, most $\tilde{p}_{n-1}^{\boldsymbol{z}}[i]p_n^{\boldsymbol{z}}[j]$ are not truncated in the TR SCONV method. Hence, to differentiate between the SCONV method and the TR SCONV method, we set $\boldsymbol{q}_n^{\boldsymbol{z}} = [0.999, 0.001]^T$ instead of $\boldsymbol{q}_n^{\boldsymbol{z}} = [0.99, 0.01]^T$ as we did in the best case. The CPU time, speedup and maximum absolute error in $\tilde{p}_{N-1}^{\boldsymbol{z}}[k]$ are listed in Table 5.6. The CPU time and speedup are given in Figure 5.22.

Table 5.6: Numerical results for testing N in the worst case for the SCONV and TR SCONV methods

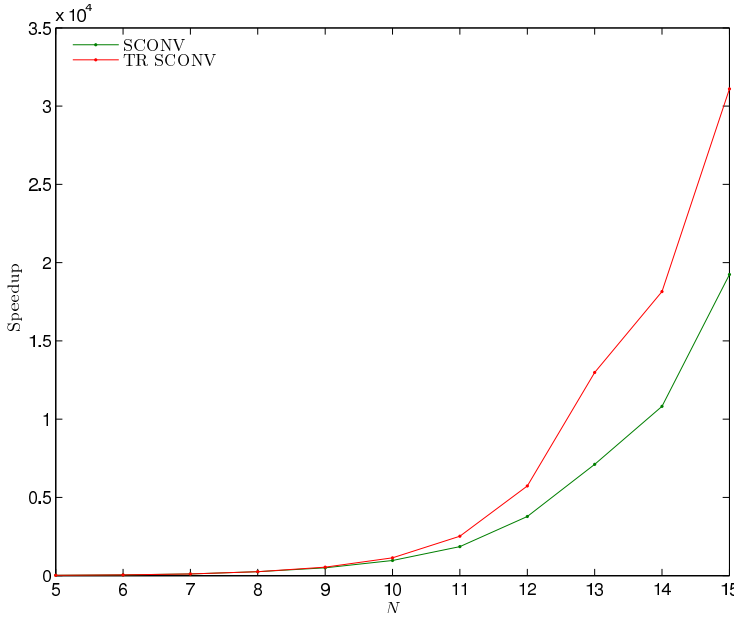
N	FCONV	SCONV		TR SCONV		
	CPU time	CPU time	Speedup	CPU time	Speedup	Max error
5	4.8879E-05	2.8456E-06	1.7177E+01	2.8084E-06	1.7405E+01	0.0000E+00
6	1.8328E-04	4.1232E-06	4.4451E+01	4.1147E-06	4.4543E+01	1.0000E-18
7	7.1740E-04	6.4959E-06	1.1044E+02	6.4813E-06	1.1069E+02	9.9900E-19
8	2.8440E-03	1.1487E-05	2.4758E+02	1.1287E-05	2.5197E+02	9.9800E-19
9	1.1339E-02	2.2779E-05	4.9778E+02	2.1028E-05	5.3923E+02	9.9700E-19
10	4.5390E-02	4.6737E-05	9.7118E+02	4.0010E-05	1.1345E+03	9.9600E-19
11	1.8338E-01	9.8854E-05	1.8551E+03	7.2842E-05	2.5175E+03	9.9500E-19
12	7.3656E-01	1.9506E-04	3.7761E+03	1.2859E-04	5.7280E+03	9.9400E-19
13	2.9516E+00	4.1519E-04	7.1090E+03	2.2741E-04	1.2979E+04	9.9300E-19
14	1.1838E+01	1.0949E-03	1.0812E+04	6.5229E-04	1.8148E+04	9.9202E-19
15	4.7299E+01	2.4584E-03	1.9240E+04	1.5208E-03	3.1101E+04	9.9104E-19

Figure 5.22: CPU time and speedup for testing N in the worst case for the SCONV and TR SCONV methods

(a) CPU time



(b) Speedup



The numerical results show that, as N increases, the CPU time required by each convolution method increases. The SCONV method and the TR SCONV method run much faster than the FCONV method, and the TR SCONV method has a larger speedup than the SCONV method. The speedup of the SCONV method over the FCONV method increases with N. Setting $\alpha = 5$ and $C = 2$, we perform a least squares

fit to the speedup of the SCONV method to

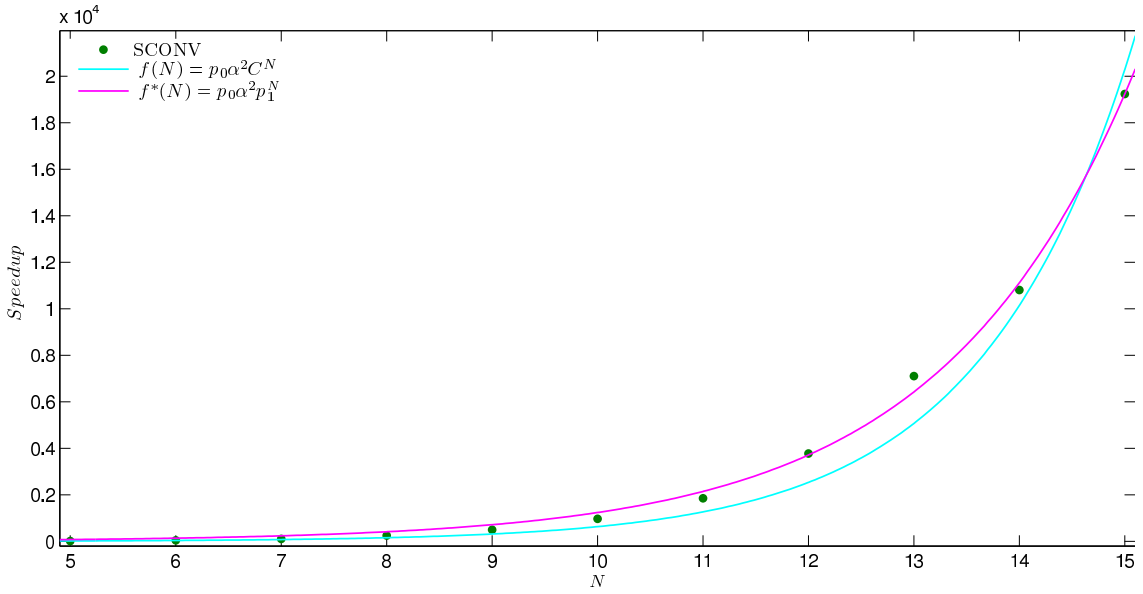
$$f_{worst,N}^{SCONV}(N) = p_0\alpha^2 C^N,$$
$$\tilde{f}_{worst,N}^{SCONV}(N) = p_0\alpha^2 p_1^N.$$

The results are

$$f_{worst,N}^{SCONV}(N) = 2.48\text{E-}02 \cdot \alpha^2 \cdot C^N, \tag{5.3.3}$$
$$\tilde{f}_{worst,N}^{SCONV}(N) = 0.26 \cdot \alpha^2 \cdot 1.73^N. \tag{5.3.4}$$

Figure 5.23 compares the fitted curves $f_{worst,N}^{SCONV}(N)$ and $\tilde{f}_{worst,N}^{SCONV}(N)$ to the actual SCONV speedup.

Figure 5.23: Comparison of the curves $f_{worst,N}^{SCONV}(N)$ and $\tilde{f}_{worst,N}^{SCONV}(N)$ to the actual speedup for the SCONV method



Comparing (5.3.3) and (5.3.4), the fitted parameter $p_2 = 1.73$ in (5.3.4) is close to $C = 2$ in (5.3.3), and Figure 5.23 shows $\tilde{f}_{worst,N}^{SCONV}(N)$ and $f_{worst,N}^{SCONV}(N)$ are good fits to the SCONV speedup. Therefore, we conclude that, in the worst case, given C, the speedup of the SCONV method grows with N at a rate approximately proportional to C^N, which agrees with our theoretical speedup in (5.1.42).

For the TR SCONV method, we also see an increase of the speedup with N. More specifically, as N increases from 5 to 15, the speedup for the TR SCONV method increases from 1.7405E+01 to 3.1101E+04. Moreover, Figure 5.22 shows that the speedup for the TR SCONV method increases faster

than for the SCONV method.

The last column in Table 5.6 indicates that the maximum absolute error in all test cases is around 10^{-18}, which is less than our pre-defined threshold $Tol = 10^{-16}$. These numerical results support the way we choose ϵ in (5.1.50), which follows from Theorem 5.2 and Corollary 5.3.

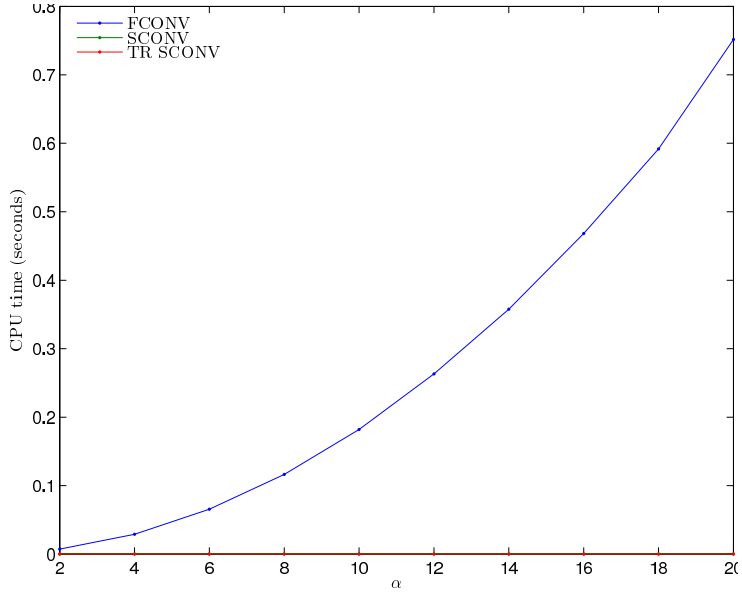
Testing α. The nonzero index vector $\boldsymbol{d}_n$ is constructed by (5.1.32). To test the impact of the parameter α, we set $N = 10$, $C = 2$, $\boldsymbol{q}_n^{\boldsymbol{z}} = [0.999, 0.001]^T$ for $n = 0, \ldots, N-1$, and compute $\tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}}$ for $\alpha = 2$, 4, 6, 8, 10, 12, 14, 16, 18, 20 by the FCONV, SCONV and TR SCONV methods. The CPU time, speedup and maximum absolute error in $\tilde{p}_{N-1}^{\boldsymbol{z}}[k]$ are listed in Table 5.7. The CPU time and speedup are given in Figure 5.24.

Table 5.7: Numerical results for testing α in the worst case for the SCONV and TR SCONV methods

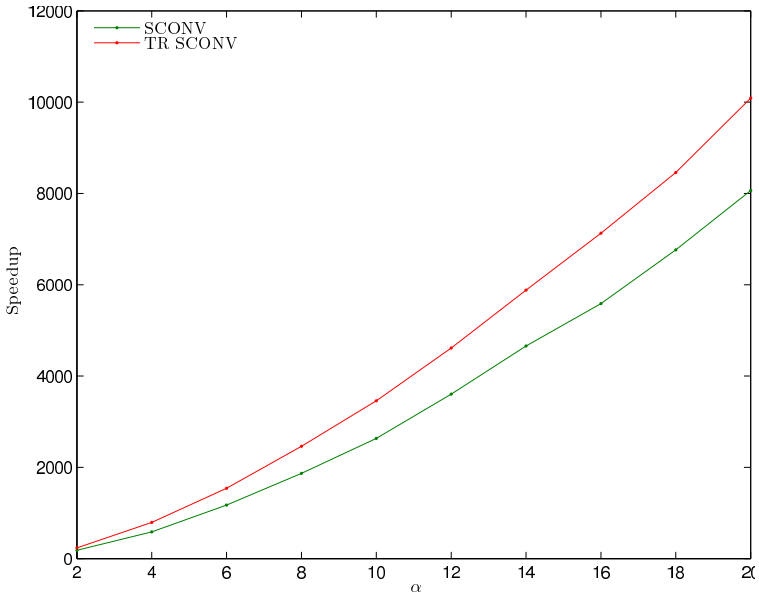
α	FCONV	SCONV		TR SCONV		
	CPU time	CPU time	Speedup	CPU time	Speedup	Max error
2	7.2563E-03	4.0056E-05	1.8115E+02	3.0984E-05	2.3420E+02	9.9600E-19
4	2.8978E-02	4.9368E-05	5.8698E+02	3.6531E-05	7.9324E+02	9.9600E-19
6	6.5528E-02	5.5913E-05	1.1720E+03	4.2510E-05	1.5415E+03	9.9600E-19
8	1.1636E-01	6.2294E-05	1.8679E+03	4.7268E-05	2.4617E+03	9.9600E-19
10	1.8198E-01	6.9105E-05	2.6334E+03	5.2628E-05	3.4579E+03	9.9600E-19
12	2.6328E-01	7.3040E-05	3.6046E+03	5.7074E-05	4.6130E+03	9.9600E-19
14	3.5772E-01	7.6822E-05	4.6565E+03	6.0807E-05	5.8829E+03	9.9600E-19
16	4.6826E-01	8.3836E-05	5.5854E+03	6.5697E-05	7.1276E+03	9.9600E-19
18	5.9164E-01	8.7474E-05	6.7636E+03	6.9958E-05	8.4571E+03	9.9600E-19
20	7.5155E-01	9.3215E-05	8.0625E+03	7.4498E-05	1.0088E+04	9.9600E-19

Figure 5.24: CPU time and speedup for testing α in the worst case for the SCONV and TR SCONV methods

(a) CPU time



(b) Speedup



The numerical results show that, as α increases, the CPU time required by each convolution method increases. The SCONV method and the TR SCONV method are much faster than the FCONV method, and the TR SCONV method has a larger speedup than the SCONV method. Moreover, the speedup of the SCONV method over the FCONV method increases with α. Setting $N = 10$ and $C = 2$, we perform a least squares fit to the speedup of the SCONV method to

$$f^{SCONV}_{worst,\alpha}(\alpha) = p_0\alpha^2 C^N,$$
$$\tilde{f}^{SCONV}_{worst,\alpha}(\alpha) = p_2\alpha^2 C^N + p_1\alpha + p_0.$$

The results are

$$f^{SCONV}_{worst,\alpha}(\alpha) = 2.09\text{E-}02 \cdot \alpha^2 \cdot C^N, \tag{5.3.5}$$
$$\tilde{f}^{SCONV}_{worst,\alpha}(\alpha) = 1.29\text{E-}02 \cdot \alpha^2 \cdot C^N + 1.73\text{E}2 \cdot \alpha - 2.67\text{E}2. \tag{5.3.6}$$

Figure 5.25 compares the fitted curves $f^{SCONV}_{worst,\alpha}(\alpha)$ and $\tilde{f}^{SCONV}_{worst,\alpha}(\alpha)$ to the actual SCONV speedup.

Figure 5.25: Comparison of curves $f^{SCONV}_{worst,\alpha}(\alpha)$ and $\tilde{f}^{SCONV}_{worst,\alpha}(\alpha)$ to the actual speedup for the SCONV method

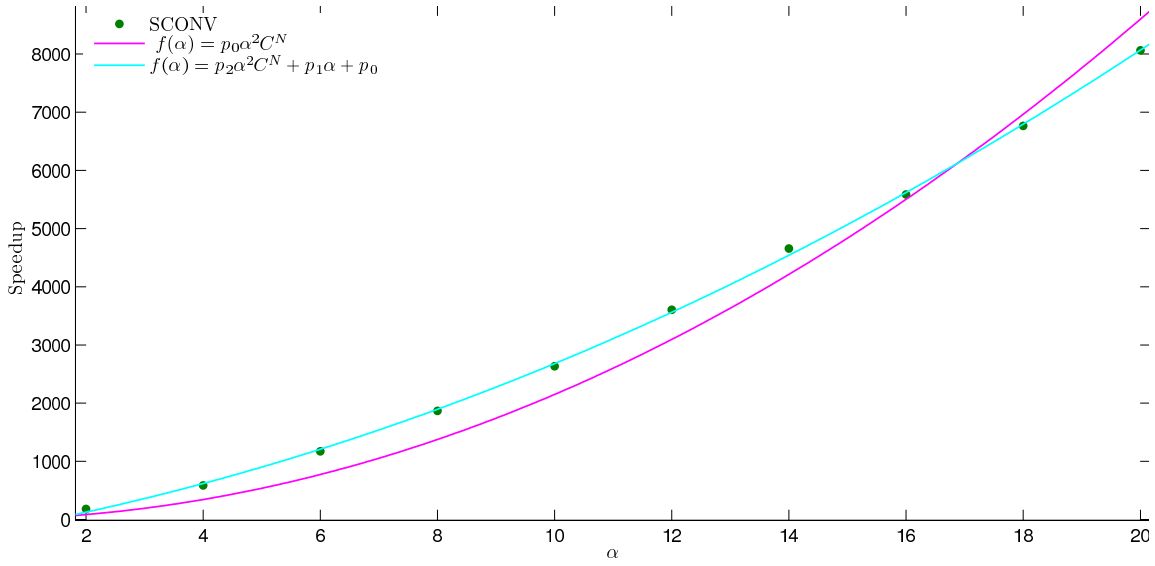


Figure 5.25 shows $\tilde{f}^{SCONV}_{worst,\alpha}(\alpha)$ and $f^{SCONV}_{worst,\alpha}(\alpha)$ are good fits to the SCONV speedup. Therefore, we conclude that, in the worst case, the speedup of the SCONV method grows with α at an approximately quadratic rate, which agrees with our theoretical speedup in (5.1.42).

For the TR SCONV method, we also see an increase of the speedup with α. As α increases from 2 to 20, the speedup of the TR SCONV method increases from 2.3420E+02 to 1.0088E+04. Moreover, Figure 5.25 shows that the speedup of the TR SCONV method increases faster than the SCONV method.

The last column in Table 5.6 shows that the maximum absolute error in all test cases is less than 1×10^{-18}, which is less than our pre-defined threshold $Tol = 10^{-16}$. These numerical results support the way we choose ϵ in (5.1.50), which follows from Theorem 5.2 and Corollary 5.3.

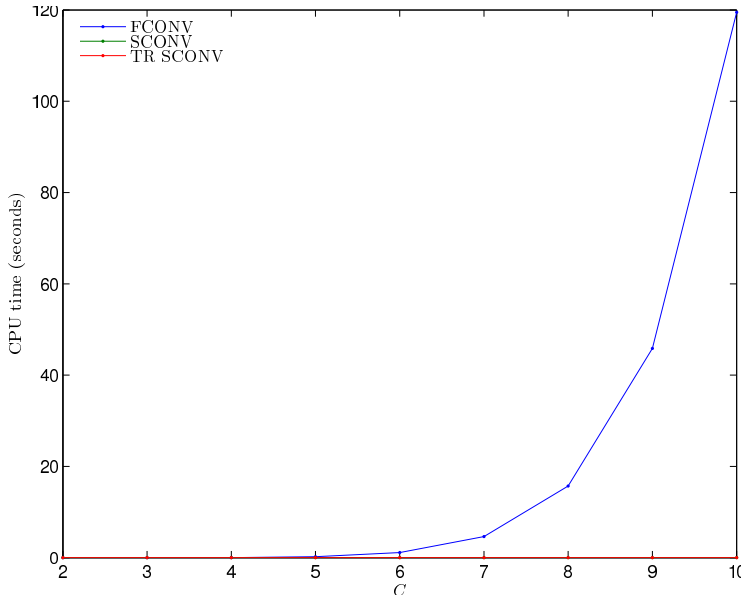
Testing C. The nonzero index vector $\boldsymbol{d}_n$ is constructed by (5.1.32). To test the impact of the parameter C, we set $N = 5$, $\alpha = 5$, and compute $\tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}}$ for $C = 2, 3, 4, 5, 6, 7, 8, 9, 10$ by the FCONV, SCONV and TR SCONV methods. The nonzero probability vector $\boldsymbol{q}_n^{\boldsymbol{z}}$ is set by (5.3.2). The CPU time, speedup and maximum absolute error in $\tilde{p}_{N-1}^{\boldsymbol{z}}[k]$ are listed in Table 5.8. The CPU time and speedup are given in Figure 5.26.

Table 5.8: Numerical results for testing C in the worst case for the SCONV and TR SCONV methods

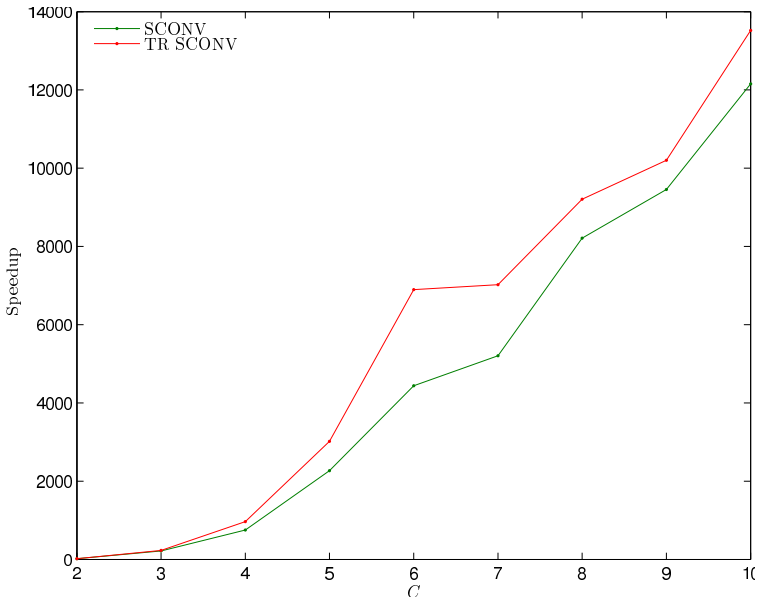
C	FCONV	SCONV		TR SCONV		
	CPU time	CPU time	Speedup	CPU time	Speedup	Max error
2	4.7681E-05	2.7492E-06	1.7344E+01	2.7615E-06	1.7266E+01	1.0000E-17
3	1.9000E-03	8.7902E-06	2.1615E+02	8.2259E-06	2.3098E+02	6.6667E-18
4	2.6975E-02	3.5919E-05	7.5100E+02	2.7927E-05	9.6591E+02	5.0000E-18
5	2.1314E-01	9.3964E-05	2.2683E+03	7.0636E-05	3.0174E+03	4.0000E-18
6	1.1339E+00	2.5541E-04	4.4395E+03	1.6440E-04	6.8972E+03	3.3333E-18
7	4.6423E+00	8.9170E-04	5.2061E+03	6.6096E-04	7.0236E+03	2.8571E-18
8	1.5689E+01	1.9104E-03	8.2124E+03	1.7044E-03	9.2050E+03	2.5000E-18
9	4.5850E+01	4.8498E-03	9.4540E+03	4.4943E-03	1.0202E+04	2.2222E-18
10	1.1949E+02	9.8298E-03	1.2156E+04	8.8366E-03	1.3522E+04	2.0000E-18

Figure 5.26: CPU time and speedup for testing C in the worst case for the SCONV and TR SCONV methods

(a) CPU time



(b) Speedup



The numerical results show that, as C increases, the CPU time required by each convolution method increases. The SCONV method and the TR SCONV method are much faster than the FCONV method, and the TR SCONV method has a larger speedup than the SCONV method. Moreover, the speedup of SCONV method over the FCONV method increases with C. Setting $N = 5$ and $\alpha = 2$, we perform a least square fit to the speedup of the SCONV method to

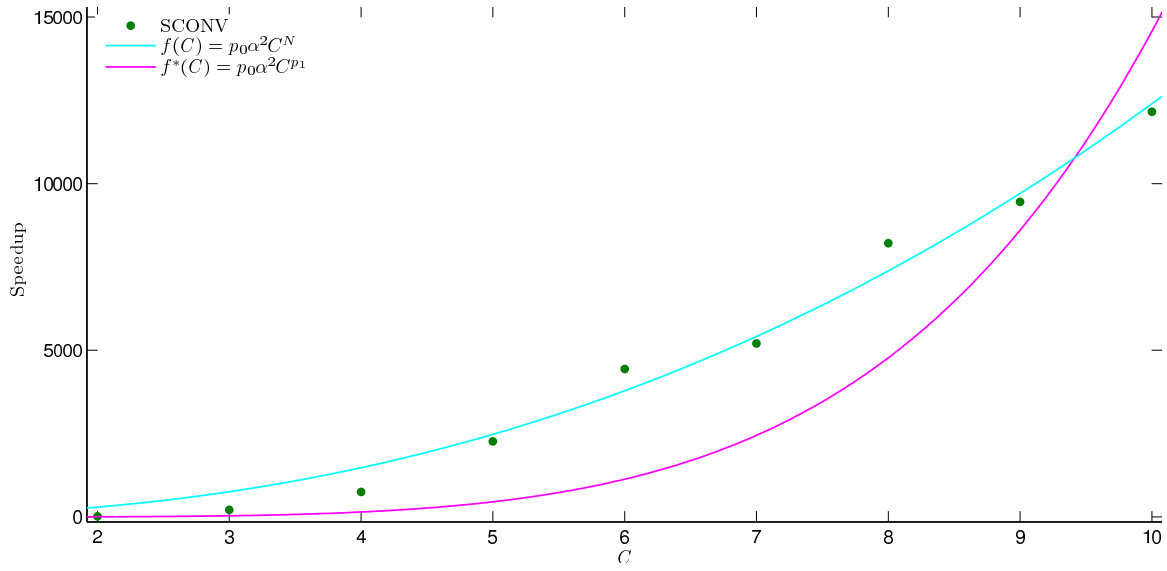
$$f_{worst,C}^{SCONV}(C) = p_0\alpha^2 C^N,$$
$$\tilde{f}_{worst,C}^{SCONV}(C) = p_0\alpha^2 C^{p_1}.$$

The results are

$$f_{worst,C}^{SCONV}(C) = 5.83\text{E-}03 \cdot \alpha^2 \cdot C^N, \tag{5.3.7}$$
$$\tilde{f}_{worst,C}^{SCONV}(C) = 2.36 \cdot \alpha^2 \cdot C^{2.32}. \tag{5.3.8}$$

Figure 5.27 compares the fitted curves $f_{worst,C}^{SCONV}(C)$ and $\tilde{f}_{worst,C}^{SCONV}(C)$ to the actual SCONV speedup.

Figure 5.27: Comparison of the curves $f_{worst,C}^{SCONV}(C)$ and $\tilde{f}_{worst,C}^{SCONV}(C)$ to the actual SCONV speedup for SCONV method



Comparing (5.3.7) and (5.3.8), the fitted parameter $p_1 = 2.32$ in (5.3.8) is significantly smaller than $N = 5$ in (5.3.7). Moreover, Figure 5.23 shows that $\tilde{f}_{worst,C}^{SCONV}(C)$ fits the actual SCONV speedup much better than $f_{worst,C}^{SCONV}(C)$. We believe the reason for this is that, $N = 5$ is fairly small in this example, while the theoretical speedup in (5.1.42) is obtained for a large N. Varying C from 2 to 10, we are not able to conduct an adequate test for large N, since as shown in (5.1.35) and (5.1.41), the total number of FLOPs in the worst case for both the TR SCONV method and the SCONV method grows with N at an exponential rate. Our test cases show that, for small N, given α, the speedup of SCONV method grows with C at a rate proportional to $\alpha^2 C^{\hat{N}}$, for some $\hat{N} < N$.

For the TR SCONV method, we also see an increase of the speedup with C. As C increases from 2 to 10, the speedup of the TR SCONV method increases from 1.7266E+01 to 1.3522E+04. Moreover, Figure 5.27 shows that the speedup of the TR SCONV method increases faster than that of the SCONV method.

The last column in Table 5.8 shows that the maximum absolute error in all test cases is around $10^{-17} \sim 10^{-18}$, which is less than our pre-defined threshold $Tol = 10^{-16}$. These numerical results support the way we choose ϵ in (5.1.50), which follows from Theorem 5.2 and Corollary 5.3.

5.3.2 Best/Worst Cases: Sparse FFT Method

As shown in (5.2.24) and (5.2.25), the speedup of the SFFT method over the FCONV method is $\Omega(\log_2 K)$ in the best case and $\Omega(\log_2 K/\log_2 C)$ in the worst case. Therefore, we build different test cases to determine the impact of the parameters N, K, and C respectively in both the best and worst cases.

5.3.2.1 Best Case

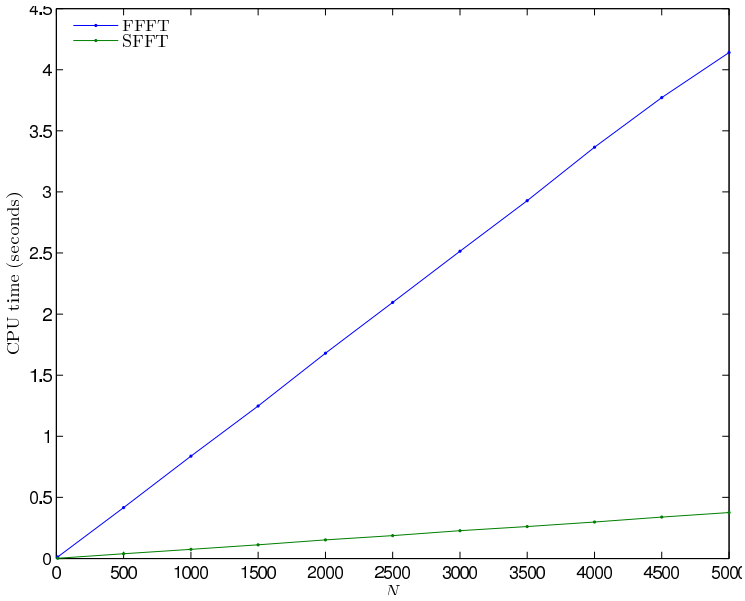
Testing N. For $n = 1, \ldots, N$, the nonzero index vector $\boldsymbol{d}_n$ is constructed such that the bit reversal of $\boldsymbol{p}_n^{\boldsymbol{z}}$ has the pattern shown in (5.2.20). To test the impact of the parameter N, we set $K = 2^{12} = 4096$, $C = 2$, $\boldsymbol{q}_n^{\boldsymbol{z}} = [0.99, 0.01]^T$ for $n = 0, \ldots, N-1$, and compute $\tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}}$ for N =10, 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000 by the FFFT and the SFFT methods. The CPU time and speedup for the SFFT method over the FFFT method are shown in Table 5.9 and Figure 5.28.

Table 5.9: Numerical results for testing N in the best case for the SFFT method

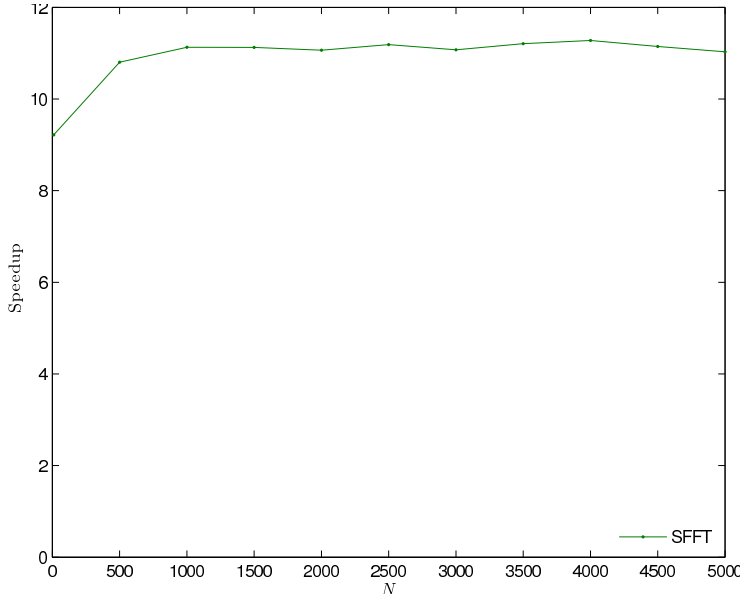
N	FFFT	SFFT	
	CPU time	CPU time	Speedup
10	1.3058E-02	1.4168E-03	9.2167E+00
500	4.1576E-01	3.8490E-02	1.0802E+01
1000	8.3717E-01	7.5225E-02	1.1129E+01
1500	1.2479E+00	1.1217E-01	1.1125E+01
2000	1.6801E+00	1.5183E-01	1.1065E+01
2500	2.0955E+00	1.8730E-01	1.1188E+01
3000	2.5143E+00	2.2704E-01	1.1074E+01
3500	2.9283E+00	2.6131E-01	1.1206E+01
4000	3.3653E+00	2.9840E-01	1.1278E+01
4500	3.7721E+00	3.3843E-01	1.1146E+01
5000	4.1405E+00	3.7541E-01	1.1029E+01

Figure 5.28: CPU time and speedup for testing N in the best case for the SFFT method

(a) CPU time



(b) Speedup



The numerical results show that, as N increases, the CPU time required by both the FFFT and SFFT methods increases. However, the SFFT method is much faster than the FFFT method. The speedup of the SFFT method over the FFFT method varies within the narrow range $[10.80, 11.28]$, for $N > 10$. This implies that when N is large, it has no significant impact on the speedup of the SFFT method. This result agrees with our theoretical speedup in (5.2.26), where there is no dependence on

N.

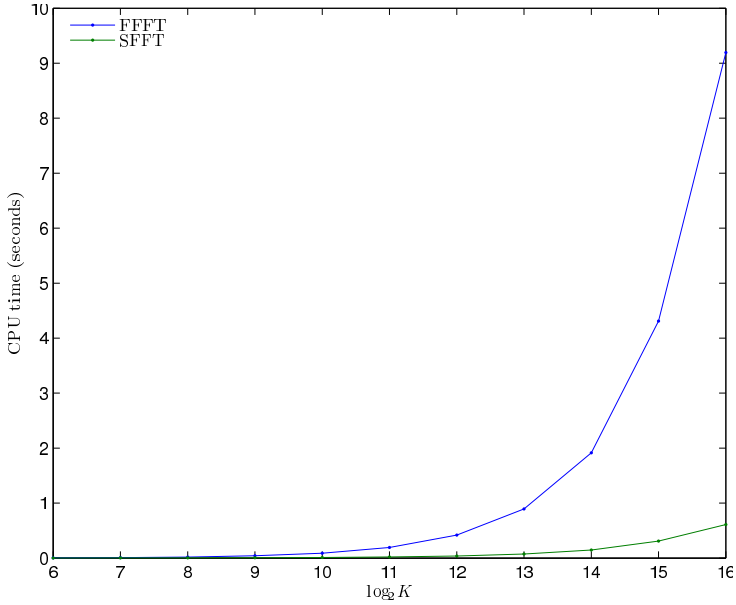
Testing K. For $n = 1, \ldots, N$, the nonzero index vector $\boldsymbol{d}_n$ is constructed such that the bit reversal of $\boldsymbol{p}_n^{\boldsymbol{z}}$ has the pattern as shown in (5.2.22). To test the impact of the parameter K, we set $N = 500$, $C = 2$, $\boldsymbol{q}_n^{\boldsymbol{z}} = [0.99, 0.01]^T$ for $n = 0, \ldots, N-1$, and compute $\tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}}$ for $K = 2^t$, $t = 6, \ldots, 16$, by the FFFT and SFFT methods. The CPU time and speedup for the SFFT method over the FFFT method are shown in Table 5.10 and Figure 5.29.

Table 5.10: Numerical results for testing K in the best case for the SFFT method

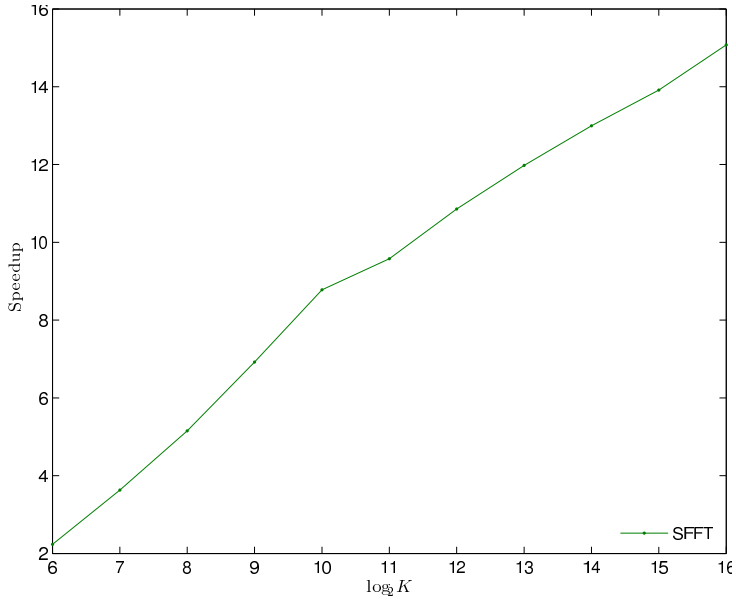
$\log_2 K$	FFFT	SFFT	
	CPU time	CPU time	Speedup
6	4.2739E-03	1.9121E-03	2.2351E+00
7	9.0629E-03	2.4977E-03	3.6285E+00
8	1.9059E-02	3.6985E-03	5.1531E+00
9	4.1372E-02	5.9792E-03	6.9193E+00
10	8.9180E-02	1.0159E-02	8.7787E+00
11	1.9197E-01	2.0042E-02	9.5784E+00
12	4.1730E-01	3.8440E-02	1.0856E+01
13	8.9428E-01	7.4671E-02	1.1976E+01
14	1.9138E+00	1.4725E-01	1.2997E+01
15	4.3107E+00	3.0983E-01	1.3913E+01
16	9.1930E+00	6.0983E-01	1.5075E+01

Figure 5.29: CPU time and speedup for testing K in the best case for the SFFT method

(a) CPU time



(b) Speedup



The numerical results show that, as K increases, the CPU time required by both FFT methods increases. However, the SFFT method is much faster than the FFFT method. Moreover, the speedup of the SFFT method over the FFFT method increases with K. We perform a least squares fit of the

speedup of the SFFT method to

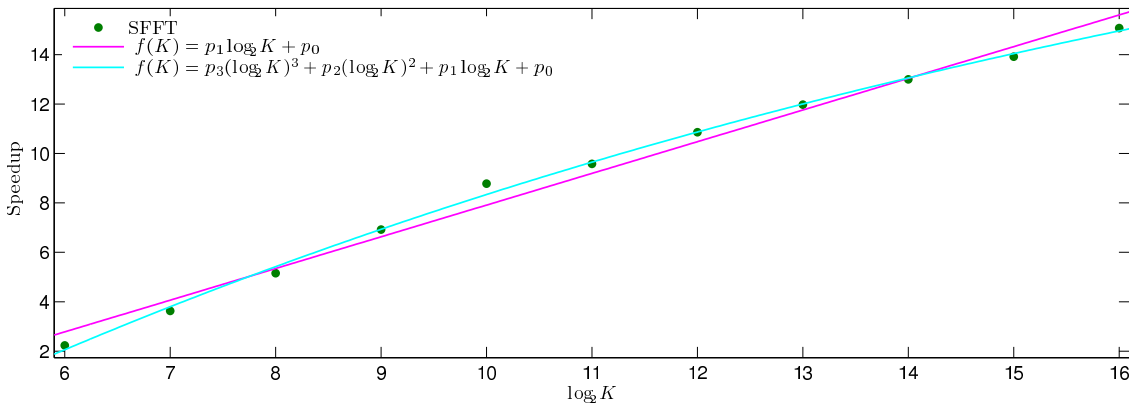
$$f_{best,K}^{SFFT}(K) = p_1 \log_2 K + p_0,$$
$$\tilde{f}_{best,K}^{SFFT}(K) = p_3(\log_2 K)^3 + p_2(\log_2 K)^2 + p_1 \log_2 K + p_0.$$

The results are

$$f_{best,K}^{SFFT}(K) = 1.28 \cdot \log_2 K - 4.91,$$
$$\tilde{f}_{best,K}^{SFFT}(K) = 1.07\text{E-}3 \cdot (\log_2 K)^3 - 8.10\text{E-}2 \cdot (\log_2 K)^2 + 2.66 \cdot \log_2 K - 11.19. \tag{5.3.9}$$

Figure 5.30 compares the fitted curves $f_{best,K}^{SFFT}(K)$ and $\tilde{f}_{best,K}^{SFFT}(K)$ to the actual SFFT speedup.

Figure 5.30: Comparison of the curves $f_{best,K}^{SFFT}(K)$ and $\tilde{f}_{best,K}^{SFFT}(K)$ to the actual speedup for the SFFT method



The coefficients for $\tilde{f}_{best,K}^{SFFT}(K)$ in (5.3.9) show that the speedup for the SFFT method has a strong dependence on $\log_2 K$, but a weak dependence on $(\log_2 K)^2$ and $(\log_2 K)^3$. Moreover, Figure 5.30 shows $\tilde{f}_{best,K}^{SFFT}(K)$ and $f_{best,K}^{SFFT}(K)$ are good fits to the SFFT speedup. Therefore, we conclude that, in the best case, the speedup of the SFFT method grows with $\log_2 K$ at a linear rate, which agrees with our theoretical speedup in (5.2.26).

Testing C. For $n = 1, \ldots, N$, the nonzero index vector $\boldsymbol{d}_n$ is constructed such that the bit reversal of $\boldsymbol{p}_n^{\boldsymbol{z}}$ has the pattern shown in (5.2.20). To test the impact of the parameter C, we set $K = 2^{15} = 32768$, $N = 500$, and vary $C = 2^t$, $t = 1, \ldots, 6$. For each C, the nonzero probability vector $\boldsymbol{q}_n^{\boldsymbol{z}}$ is set to

$$q_n^{\boldsymbol{z}}[c] = \begin{cases} 1 - 0.001 \sum_{k=1}^{C-1} k, & c = 0, \\ 0.001(C - c), & c = 1, \ldots, C-1. \end{cases} \tag{5.3.10}$$

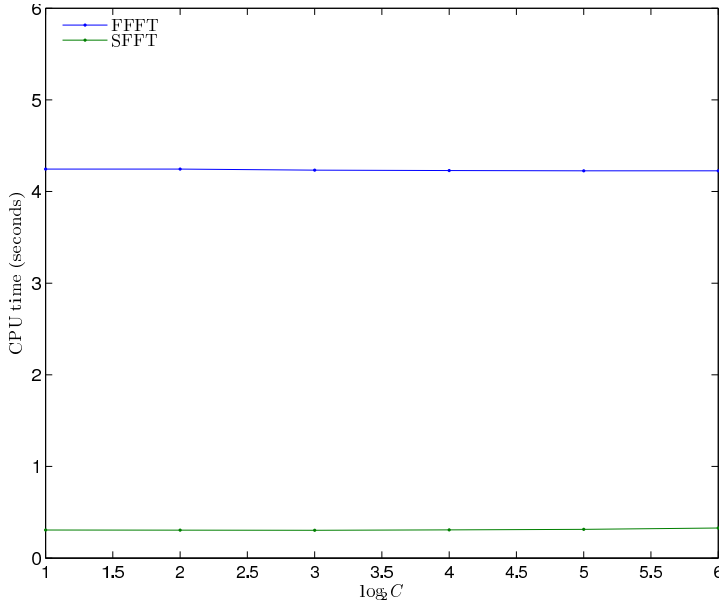
for $n = 0, \ldots, N-1$. We compute $\tilde{\boldsymbol{p}}^{\boldsymbol{z}}_{N-1}$ for $C = 2^t$, $t = 1, \ldots, 6$, by the FFFT and SFFT methods. The CPU time and speedup of the SFFT method over the FFFT method are shown in Table 5.11 and Figure 5.31.

Table 5.11: Numerical results for testing C in the best case for the SFFT method

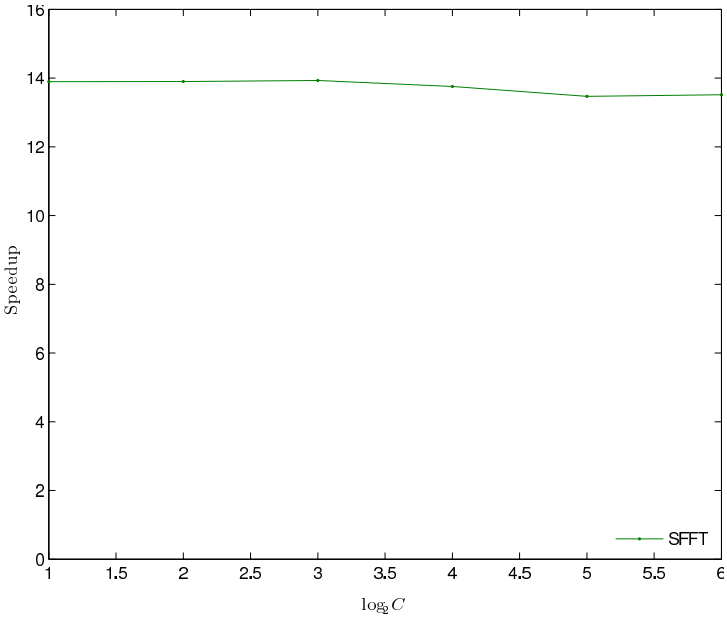
$\log_2 C$	FFFT	SFFT	
	CPU time	CPU time	Speedup
1	4.2453E+00	3.0555E-01	1.3894E+01
2	4.2444E+00	3.0541E-01	1.3898E+01
3	4.2331E+00	3.0392E-01	1.3929E+01
4	4.2280E+00	3.0733E-01	1.3757E+01
5	4.2252E+00	3.1371E-01	1.3468E+01
6	4.2264E+00	3.1270E-01	1.3515E+01

Figure 5.31: CPU time and speedup in testing C for the best case for the SFFT method

(a) CPU time



(b) Speedup



The numerical results show that, as C increases, the CPU time required by both FFT methods does not change significantly. However, the SFFT method is much faster than the FFFT method. The speedup of the SFFT method over the FFFT method varies within the narrow range $[13.47, 13.93]$, which implies that C has no significant impact on the speedup of the SFFT method. This result agrees with our theoretical speedup in (5.2.26), where there is no dependence on C.

5.3.2.2 Worst case

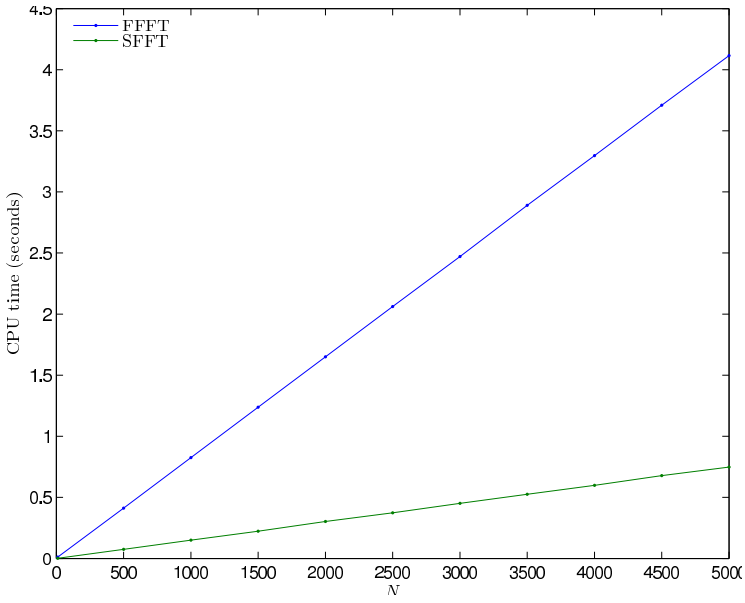
Testing N. For $n = 1, \ldots, N$, the nonzero index vector $\boldsymbol{d}_n$ is constructed such that the bit reversal of $\boldsymbol{p}_n^{\boldsymbol{z}}$ has the pattern shown in (5.2.22). To test the impact of the parameter N, we set $K = 2^{12} = 4096$, $C = 2$, $\boldsymbol{q}_n^{\boldsymbol{z}} = [0.99, 0.01]^T$ for $n = 0, \ldots, N-1$, and compute $\tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}}$ for N =10, 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000 by the FFFT and SFFT methods. The CPU time and speedup of the SFFT method over the FFFT method are shown in Table 5.12 and Figure 5.32.

Table 5.12: Numerical results for testing N in the worst case for the SFFT method

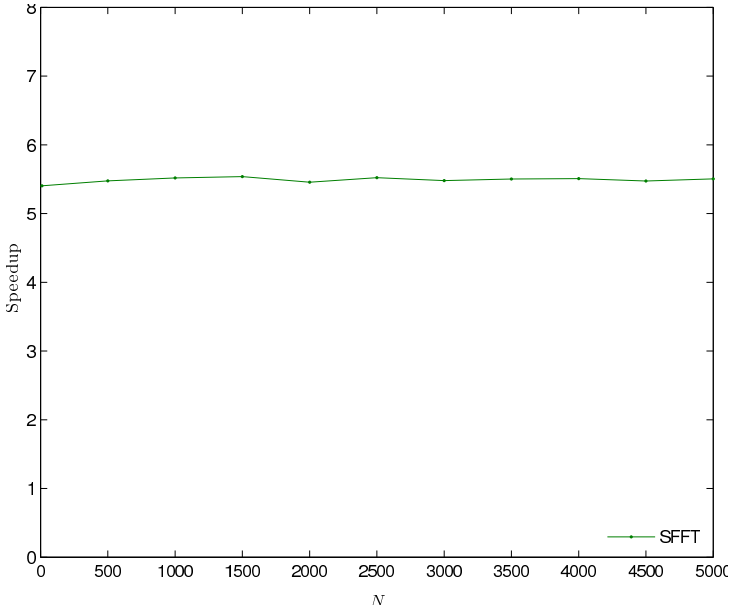
N	FFFT	SFFT	
	CPU time	CPU time	Speedup
10	1.0966E-02	2.0292E-03	5.4044E+00
500	4.1209E-01	7.5271E-02	5.4748E+00
1000	8.2536E-01	1.4959E-01	5.5175E+00
1500	1.2383E+00	2.2363E-01	5.5372E+00
2000	1.6509E+00	3.0255E-01	5.4566E+00
2500	2.0623E+00	3.7351E-01	5.5214E+00
3000	2.4708E+00	4.5089E-01	5.4799E+00
3500	2.8903E+00	5.2536E-01	5.5015E+00
4000	3.2969E+00	5.9861E-01	5.5075E+00
4500	3.7097E+00	6.7783E-01	5.4729E+00
5000	4.1149E+00	7.4766E-01	5.5037E+00

Figure 5.32: CPU time and speedup for testing N in the worst case for the SFFT method

(a) CPU time



(b) Speedup



The numerical results show that, as N increases, the CPU time required by both FFT methods increases. However, the SFFT method is much faster than the FFFT method. The speedup for the SFFT method over the FFFT method varies within the narrow range $[5.40, 5.54]$. This shows that N has no significant impact on the speedup of the SFFT method. This result agrees with our theoretical speedup in (5.2.27), where there is no dependence on N.

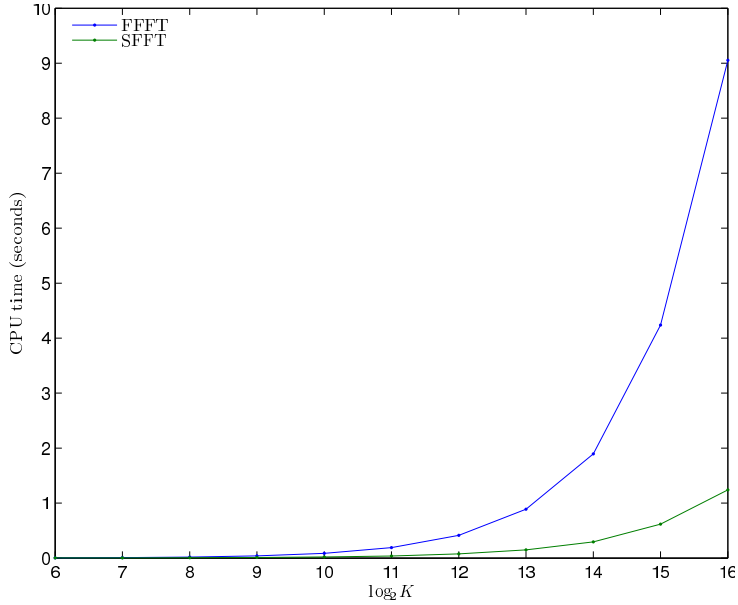
Testing K. For $n = 1, \ldots, N$, the nonzero index vector $\boldsymbol{d}_n$ is constructed such that the bit reversal of $\boldsymbol{p}_n^{\boldsymbol{z}}$ has the pattern as shown in (5.2.22). To test the impact of the parameter K, we set $N = 500$, $C = 2$, $\boldsymbol{q}_n^{\boldsymbol{z}} = [0.99, 0.01]^T$ for $n = 0, \ldots, N-1$, and compute $\tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}}$ for $K = 2^t$, $t = 6, \ldots, 16$, by the FFFT and SFFT methods. The CPU time and speedup for the SFFT method over the FFFT method are shown in Table 5.13 and Figure 5.33.

Table 5.13: Numerical results for testing K in the worst case for the SFFT method

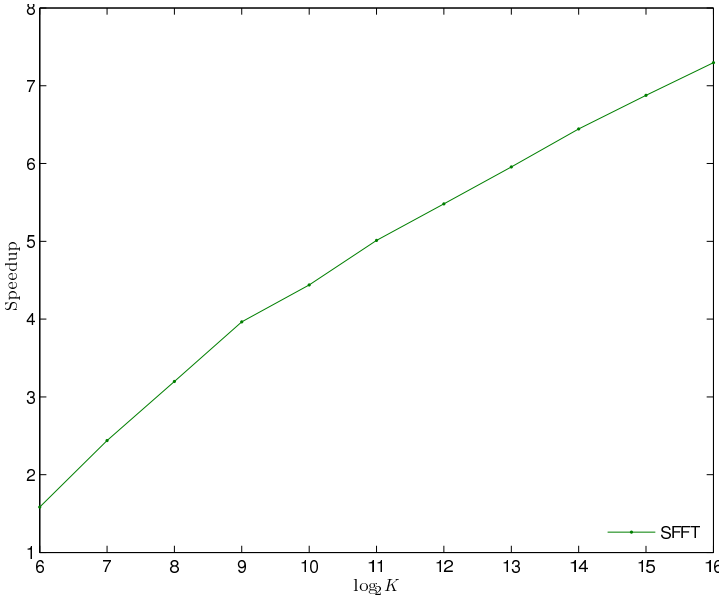
$\log_2 K$	FFFT	SFFT	
	CPU time	CPU time	Speedup
6	3.8460E-03	2.4288E-03	1.5835E+00
7	8.7737E-03	3.5982E-03	2.4384E+00
8	1.8969E-02	5.9309E-03	3.1982E+00
9	4.0623E-02	1.0245E-02	3.9653E+00
10	8.6628E-02	1.9518E-02	4.4383E+00
11	1.9047E-01	3.8004E-02	5.0119E+00
12	4.1286E-01	7.5320E-02	5.4815E+00
13	8.8957E-01	1.4934E-01	5.9566E+00
14	1.8956E+00	2.9417E-01	6.4437E+00
15	4.2369E+00	6.1615E-01	6.8764E+00
16	9.0529E+00	1.2406E+00	7.2973E+00

Figure 5.33: CPU time and speedup for testing K in the worst case for the SFFT method

(a) CPU time



(b) Speedup



The numerical results show that, as K increases, the CPU time required by each FFT method increases. The SFFT method is much faster than the FFFT method. The speedup of the SFFT method over the FFFT method increases with K. We perform a least squares fit of the speedup of the SFFT

method to

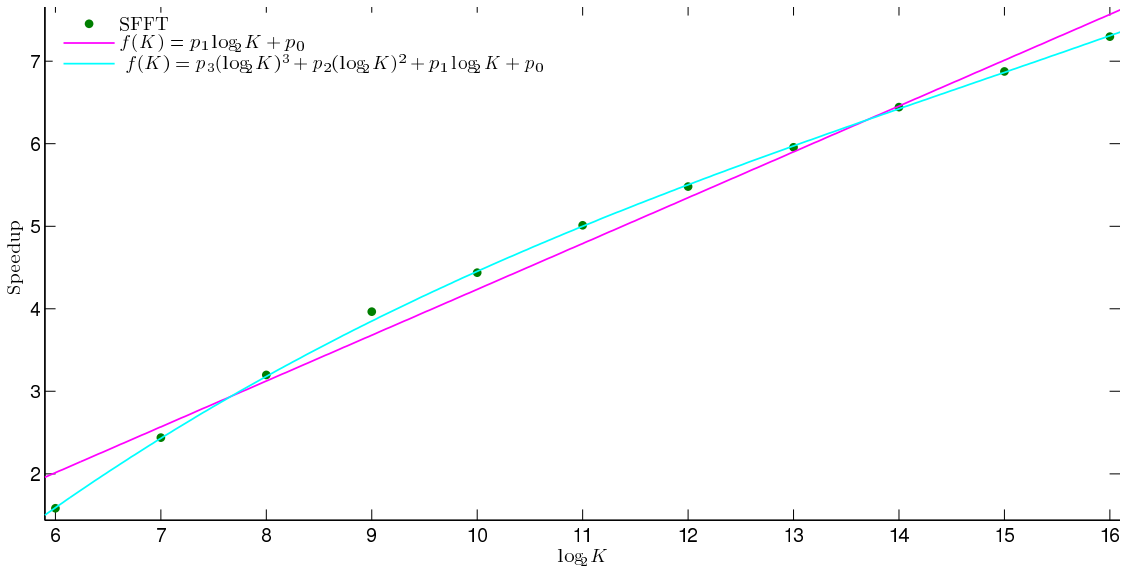
$$f^{SFFT}_{worst,K}(K) = p_1 \log_2 K + p_0,$$
$$\tilde{f}^{SFFT}_{worst,K}(K) = p_3(\log_2 K)^3 + p_2(\log_2 K)^2 + p_1 \log_2 K + p_0.$$

The results are

$$f^{SFFT}_{worst,K}(K) = 0.55 \cdot \log_2 K - 1.32,$$
$$\tilde{f}^{SFFT}_{worst,K}(K) = 1.97\text{E-}3 \cdot (\log_2 K)^3 - 8.68\text{E-}2 \cdot (\log_2 K)^2 + 1.71 \cdot \log_2 K - 6.03. \tag{5.3.11}$$

Figure 5.34 compares the fitted curve $f^{SFFT}_{worst,K}(K)$ and $\tilde{f}^{SFFT}_{worst,K}(K)$ to the actual SFFT speedup.

Figure 5.34: Comparison of the curves $f^{SFFT}_{worst,K}(K)$ and $\tilde{f}^{SFFT}_{worst,K}(K)$ to the actual speedup for the SFFT method



The coefficients of $\tilde{f}^{SFFT}_{worst,K}(K)$ in (5.3.11) show that the speedup for the SFFT method has a strong dependence on $\log_2 K$, but a weak dependence on $(\log_2 K)^2$ and $(\log_2 K)^3$. Figure 5.34 shows $\tilde{f}^{SFFT}_{worst,K}(K)$ and $f^{SFFT}_{worst,K}(K)$ are good fits to the SFFT speedup. Therefore, we conclude that, in the worst case, the speedup of SFFT method grows with $\log_2 K$ at a linear rate, which agrees with our theoretical speedup in (5.2.27).

Testing C. For $n = 1, \ldots, N$, the nonzero index vector $\boldsymbol{d}_n$ is constructed such that the bit reversal of $\boldsymbol{p}^{\boldsymbol{z}}_n$ has the pattern shown in (5.2.22). To test the impact of the parameter C, we set $K = 2^{15} = 32768$,

$N = 500$, and vary $C = 2^t$, $t = 1, \ldots, 6$. For each C, the nonzero probability vector $\boldsymbol{q}_n^{\boldsymbol{z}}$ is set to

$$q_n^{\boldsymbol{z}}[c] = \begin{cases} 1 - 0.001 \sum_{k=1}^{C-1} k, & c = 0, \\ 0.001(C - c), & c = 1, \ldots, C-1. \end{cases} \tag{5.3.12}$$

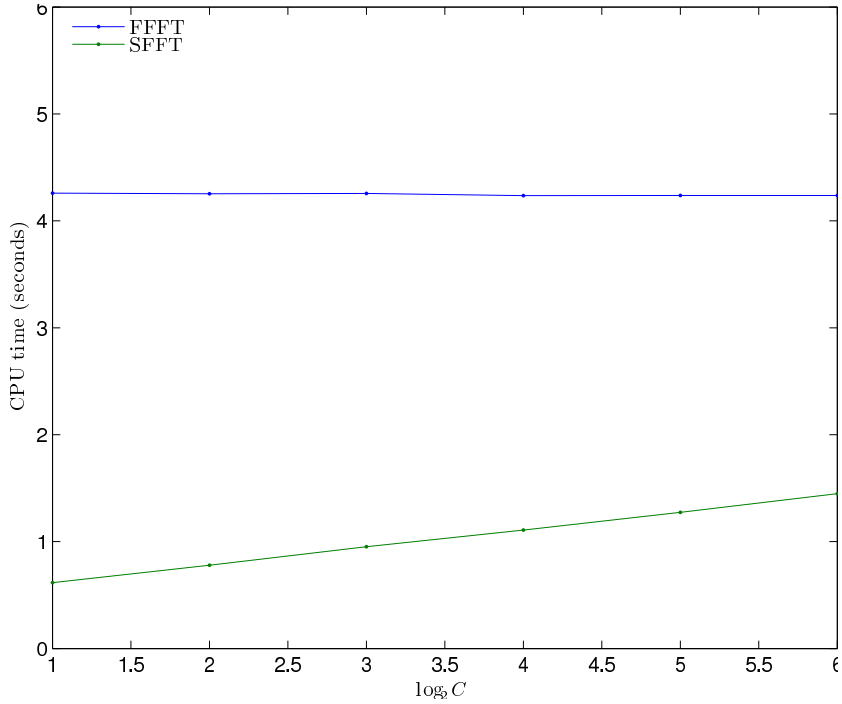
for $n = 0, \ldots, N-1$. We compute $\tilde{\boldsymbol{p}}_{N-1}^{\boldsymbol{z}}$ for $C = 2^t$, $t = 1, \ldots, 6$, by the FFFT and SFFT methods. The CPU time and speedup of the SFFT method over the FFFT method are shown in Table 5.14 and Figure 5.35.

Table 5.14: Numerical results for testing C in the worst case for the SFFT method

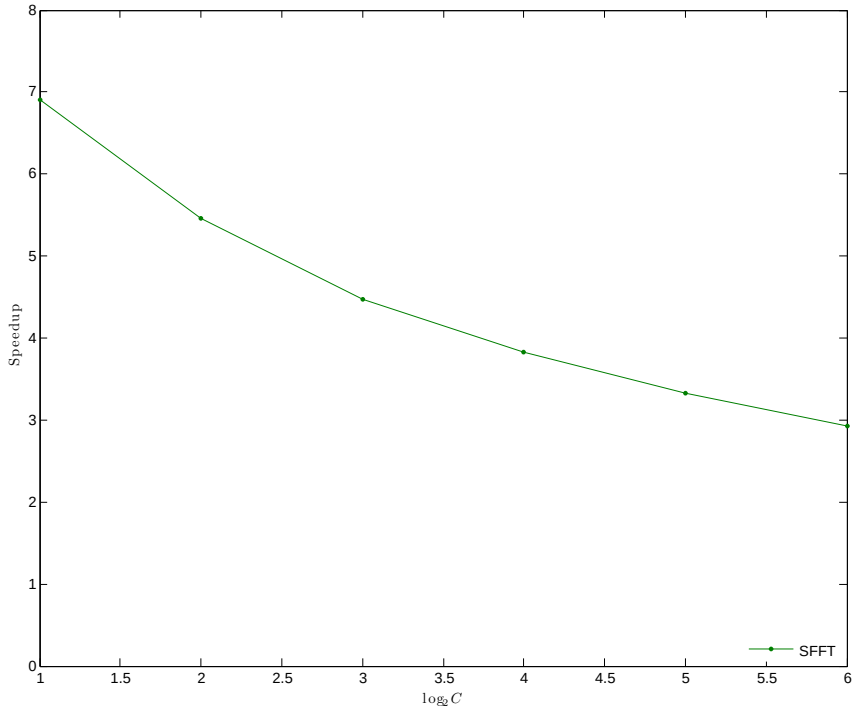
$\log_2 C$	FFFT	SCONV	
	CPU time	CPU time	Speedup
1	4.2592E+00	6.1680E-01	6.9054E+00
2	4.2533E+00	7.7951E-01	5.4564E+00
3	4.2556E+00	9.5273E-01	4.4667E+00
4	4.2354E+00	1.1072E+00	3.8255E+00
5	4.2371E+00	1.2736E+00	3.3268E+00
6	4.2371E+00	1.4482E+00	2.9259E+00

Figure 5.35: CPU time and speedup for testing C in the worst case for the SFFT method

(a) CPU time



(b) Speedup



The numerical results show that, as C increases, the CPU time required by the FFFT method does

not change significantly, but the CPU time for the SFFT method increases. The number of rating classes does have an impact on the efficiency of the FFFT method, as can be explained by the complexity analysis presented in Subsection 5.2.2.5. As C increases, the probability vectors become less sparse, causing the SFFT method to slow down. Nevertheless, the SFFT method is much faster than the FFFT method. The speedup of the SFFT method over the FFFT method decreases with C. We perform a least squares fit of the speedup for the SFFT method to

$$f^{SFFT}_{worst,C}(C) = \frac{p_1}{\log_2 C},$$
$$\tilde{f}^{SFFT}_{worst,C}(C) = \frac{p_1}{p_2 + p_3 \log_2 C}.$$

The results are

$$f^{SFFT}_{worst,C}(C) = \frac{8.87}{\log_2 C},$$
$$\tilde{f}^{SFFT}_{worst,C}(C) = \frac{15.52}{2.71 + \log_2 C}. \tag{5.3.13}$$

Figure 5.36 compares the fitted curves $f^{SFFT}_{worst,C}(C)$ and $\tilde{f}^{SFFT}_{worst,C}(C)$ to the actual SFFT speedup.

Figure 5.36: Comparison of the curves $f^{SFFT}_{worst,C}(C)$ and $\tilde{f}^{SFFT}_{worst,C}(C)$ to the actual speedup for the SFFT method

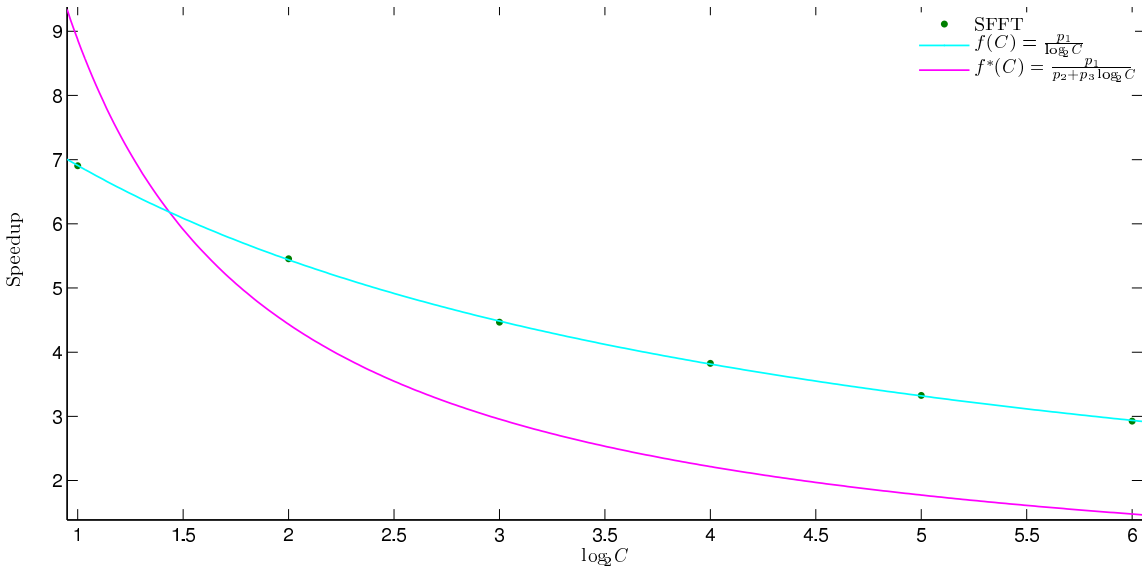


Figure 5.36 shows that $f^{SFFT}_{best,K}(K)$ does not fit the SFFT speedup very well, but $\tilde{f}^{SFFT}_{best,K}(K)$ does. Note that $\tilde{f}^{SFFT}_{worst,C}(C) \sim \Omega(1/\log_2 C)$. Therefore, as C increases, it is still true that, for fixed N and K, the speedup of the SFFT method varies approximately proportional to $1/\log_2 C$, which agrees with our

theoretical speedup in (5.2.27).

5.3.3 Testing on Synthetic Portfolios

In this subsection, we compare the SCONV, TR SCONV, FFFT, SFFT, TR SFFT and EW TR SFFT methods to the benchmark FCONV method in computing the loss distribution for realistic synthetic portfolios. We build six testing portfolios to test the efficiency and accuracy of different methods. For all portfolios, the number of obligors is $N = 500$, and all obligors are in the highest rating class. The synthetic testing portfolios are divided into two groups: homogeneous and inhomogeneous. The portfolios within each group differ in the the number of rating classes, C, which is also the number of nonzero elements in $\boldsymbol{p}_n^{\boldsymbol{z}}$. We test $C = 2$ and 18, where $C = 2$ corresponds to the simplest rating system - default and no default - and $C = 18$ corresponds to a typical rating system in the market (see Table 2.1). The following table summarizes the portfolios we use in our tests.

Table 5.15: Testing portfolios for exact methods

Portfolio		C	$c(n)$	EAD_n	LGC_n^c	β_n
Homo-geneous	Π_1	2	1	1	$\overline{\text{LGC}^1}_n^c$	0.5
	Π_2	18	17		$\overline{\text{LGC}^2}_n^c$	
Hetero-geneous	Π_1^*	2	1	$\text{Unif}(0.5, 1)$	$\widetilde{\text{LGC}^1}_n^c$	
	Π_2^*	18	17		$\widetilde{\text{LGC}^2}_n^c$	

The loss-given-credit-event, LGC_n^c, given in Table 5.15 are set to be

$$\overline{\mathbf{LGC^1}}_n = [0.9, 0]^T;$$

$$\overline{\mathbf{LGC^2}}_n = [0.9, \mathbf{LGC}^*]^T,\ \mathbf{LGC}^* = [0.8 : -0.05 : 0];$$

$$\widetilde{\text{LGC}^{1,2}}_n^c = \begin{cases} \text{Unif(0.9,1)}, & \text{if} \quad c = 0 \\ \text{Unif}\left(0, \widetilde{\text{LGC}^{1,2}}_n^{c-1}\right), & \text{if} \quad 0 < c < C-2 \cdot \\ 0, & \text{if} \quad c = C-1 \end{cases}$$

The unconditional credit migrating probabilities are listed in Table 5.16 for $C = 2$ and in Table 5.17 for

$C = 18$.

Table 5.16: Credit migration matrix for $C = 2$

c	1	0
1	0.9073	0.0927
0	0	1

Table 5.17: Credit migration matrix for $C = 18$

c	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
17	0.9062	0.0502	0.0186	0.0050	0.0049	0.0021	0.0009	0.0002	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0119
16	0.0324	0.7989	0.0841	0.0517	0.0137	0.0036	0.0015	0.0011	0.0002	0.0001	0.0003	0.0001	0.0004	0.0000	0.0000	0.0000	0.0000	0.0119
15	0.0146	0.0458	0.8073	0.0795	0.0287	0.0077	0.0027	0.0007	0.0002	0.0001	0.0000	0.0001	0.0003	0.0001	0.0003	0.0000	0.0000	0.0119
14	0.0025	0.0080	0.0367	0.8225	0.0778	0.0277	0.0068	0.0010	0.0009	0.0008	0.0001	0.0004	0.0001	0.0007	0.0001	0.0000	0.0000	0.0139
13	0.0008	0.0013	0.0067	0.0505	0.8157	0.0716	0.0250	0.0062	0.0018	0.0007	0.0014	0.0014	0.0005	0.0004	0.0001	0.0000	0.0000	0.0159
12	0.0004	0.0005	0.0024	0.0101	0.0503	0.8014	0.0751	0.0286	0.0077	0.0026	0.0007	0.0008	0.0008	0.0005	0.0001	0.0000	0.0003	0.0177
11	0.0004	0.0007	0.0012	0.0024	0.0154	0.0733	0.7589	0.0679	0.0364	0.0125	0.0036	0.0020	0.0015	0.0015	0.0005	0.0001	0.0003	0.0214
10	0.0004	0.0007	0.0011	0.0012	0.0031	0.0233	0.0692	0.7456	0.0816	0.0293	0.0088	0.0028	0.0033	0.0026	0.0013	0.0005	0.0002	0.0250
9	0.0001	0.0010	0.0008	0.0011	0.0022	0.0089	0.0281	0.0633	0.7446	0.0773	0.0233	0.0064	0.0068	0.0033	0.0027	0.0010	0.0005	0.0286
8	0.0004	0.0001	0.0004	0.0005	0.0022	0.0035	0.0070	0.0314	0.0871	0.6977	0.0675	0.0314	0.0175	0.0077	0.0036	0.0011	0.0015	0.0394
7	0.0003	0.0001	0.0007	0.0004	0.0014	0.0011	0.0033	0.0075	0.0285	0.0850	0.6877	0.0543	0.0467	0.0141	0.0108	0.0054	0.0028	0.0499
6	0.0000	0.0000	0.0004	0.0001	0.0004	0.0008	0.0010	0.0035	0.0068	0.0253	0.0810	0.6625	0.0810	0.0289	0.0304	0.0119	0.0057	0.0603
5	0.0000	0.0000	0.0001	0.0003	0.0004	0.0013	0.0011	0.0016	0.0028	0.0059	0.0230	0.0583	0.6699	0.0707	0.0511	0.0223	0.0073	0.0839
4	0.0000	0.0000	0.0003	0.0000	0.0004	0.0007	0.0010	0.0011	0.0013	0.0021	0.0054	0.0221	0.0601	0.6724	0.0650	0.0456	0.0155	0.1070
3	0.0000	0.0000	0.0003	0.0000	0.0006	0.0001	0.0004	0.0009	0.0012	0.0011	0.0026	0.0105	0.0245	0.0545	0.6320	0.1001	0.0417	0.1295
2	0.0000	0.0003	0.0003	0.0000	0.0000	0.0001	0.0006	0.0004	0.0014	0.0020	0.0013	0.0032	0.0086	0.0291	0.0466	0.6707	0.0840	0.1514
1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0001	0.0003	0.0037	0.0051	0.0032	0.0009	0.0045	0.0159	0.0154	0.0321	0.7307	0.1881
0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	1.0000

For each testing portfolio, we first compute the maximum portfolio loss L_{max} by (5.1.6), and set the minimum portfolio loss to be zero, since initially all obligors are in the highest credit rating state. Then we construct a discretization grid by (5.1.13) and (5.1.14) with $K = 2^{16} = 65536$. Notice that K is chosen to be a power of two for our FFT methods which are based on the Cooley-Tukey radix-2 FFT algorithm. Based on this discretization scheme, indices of nonzero elements are computed by

$$d_n[c] = \lfloor L_n^c/\delta \rfloor, \tag{5.3.14}$$

for $c = 0, \ldots, C-1$. The discretization errors associated with(5.3.14) have no impact on our comparisons of the accuracy of the different methods, since all methods use the same $\boldsymbol{d}_n$. Since the methods tested here only differ in how they compute the conditional loss probabilities, it is enough to sample once only for systematic risk factors $\boldsymbol{\mathcal{Z}}$. That is, in our testing, the sample size for the outer simulation is 1, and, to eliminate the impact of sampling error, all methods share the same sample of systematic risk factors $\boldsymbol{\mathcal{Z}}$. It should be mentioned that, changing the values of $\boldsymbol{\mathcal{Z}}$ does not change the location of the nonzero elements in the conditional probability vectors; hence different values of $\boldsymbol{\mathcal{Z}}$ would not have an impact on the efficiency of the sparse methods. Conditional on $\boldsymbol{\mathcal{Z}} = \boldsymbol{z}$, the nonzero conditional probabilities $\boldsymbol{q}_n^{\boldsymbol{z}} = \left[q_n^{\boldsymbol{z}}[0], \ldots, q_n^{\boldsymbol{z}}[C-1]\right]^T$ are computed by (5.1.1).

For the FCONV, SCONV, TR SCONV, FFFT, SFFT methods, we first compute the portfolio loss probabilities by

$$p[k] = \mathbb{P}\{\mathcal{L} = l_k\}, \tag{5.3.15}$$

where $l_k = k\delta$, for $k = 0, \ldots, K-1$, and then we compute the cumulative loss probabilities by

$$\hat{p}[k] \doteq \mathbb{P}\{\mathcal{L} \leq l_k\} = \sum_{j \leq k} p[j] \tag{5.3.16}$$

for $k = 0, \ldots, K-1$. To distinguish the results for different methods, we denote $\hat{\boldsymbol{p}}^{method}$ as the cumulative loss probabilities vector computed by method *method*. As mentioned above, we use the FCONV method as the benchmark, then based on $\hat{\boldsymbol{p}}^{FCONV}$ we compute VaR at the confidence level $\gamma = 95\%,\ 99\%,\ 99.9\%$ and 99.98% by $\text{VaR}_\gamma = k_\gamma\delta$, where $k_\gamma = \min\{k :\ \hat{p}^{FCONV}[k] \geq \gamma\}$. For the TR SFFT and EW TR SFFT

methods, we first determine the length, $\bar{K}$, of truncated vectors $\overline{\boldsymbol{p}_n^{\boldsymbol{z}}}$ by

$$\bar{K} = \min\left\{2^t \; 2^t \geq \mathrm{VaR}_\gamma\right\} \tag{5.3.17}$$

and then apply the TR SFFT and EW TR SFFT methods to (5.3.15) and (5.3.16) to compute the cumulative loss probabilities $\hat{p}^{TR\ SFFT}[k]$ and $\hat{p}^{EW\ TR\ SFFT}[k]$ for $k = 0, \ldots, \bar{K} - 1$. Notice that, for the TR SCONV method, we let the threshold for errors in cumulative loss probabilities be $Tol = 10^{-4}$, and set ϵ by (5.1.51). For the EW TR SFFT method, the optimal threshold is applied.

5.3.3.1 Accuracy

To assess the accuracy, we compare $\hat{p}^{method}[k_\gamma]$ to $\hat{p}^{FCONV}[k_\gamma]$ by computing the relative difference

$$\delta_\gamma^{method} = \left|\frac{\hat{p}^{method}[k_\gamma] - \hat{p}^{FCONV}[k_\gamma]}{\hat{p}^{FCONV}[k_\gamma]}\right|.$$

Numerical results are listed in Table 5.18.

Table 5.18: Relative difference in computing cumulative loss probabilities for the synthetic portfolios

Portfolio	γ	SCONV	TR SCONV	FFFT	SFFT	TR SFFT	EW TR SFFT
	95.00%	0.0000E+00	1.4462E-12	6.9944E-15	6.9944E-15	4.6121E-03	1.5212E-10
	99.00%	0.0000E+00	6.6760E-10	2.0203E-15	2.0203E-15	4.5668E-03	6.0558E-10
Π_1	99.50%	0.0000E+00	9.2452E-10	4.0185E-15	4.0185E-15	4.5419E-03	1.0029E-09
	99.90%	0.0000E+00	2.2267E-09	2.0005E-15	2.0005E-15	2.5059E-09	8.7274E-11
	99.98%	0.0000E+00	3.6096E-09	2.9982E-15	2.9982E-15	2.5038E-09	5.4503E-11
	95.00%	0.0000E+00	5.7029E-10	1.0529E-15	1.0529E-15	4.9932E-04	3.8893E-11
	99.00%	0.0000E+00	1.1255E-09	1.0093E-15	1.0093E-15	4.7867E-04	7.1638E-11
Π_2	99.50%	0.0000E+00	1.4293E-09	1.0042E-15	1.0042E-15	4.7626E-04	8.5115E-11
	99.90%	0.0000E+00	2.2645E-09	1.0002E-15	1.0002E-15	4.7434E-04	1.1316E-11
	99.98%	0.0000E+00	3.3572E-09	0.0000E+00	0.0000E+00	1.3372E-09	2.0055E-11
	95.00%	0.0000E+00	7.8558E-10	2.1038E-15	2.1038E-15	2.1308E-02	1.1229E-09
	99.00%	0.0000E+00	2.3471E-09	1.0093E-15	1.0093E-15	3.7618E-11	6.5662E-13
Π_1^*	99.50%	0.0000E+00	3.2772E-09	1.0042E-15	1.0042E-15	3.7425E-11	1.4614E-12
	99.90%	0.0000E+00	5.9515E-09	1.0002E-15	1.0002E-15	3.7277E-11	5.0555E-13
	99.98%	0.0000E+00	9.3155E-09	0.0000E+00	0.0000E+00	3.7248E-11	4.9315E-13
	95.00%	0.0000E+00	8.4618E-11	1.0521E-15	1.0521E-15	4.0012E-03	1.5583E-10
	99.00%	0.0000E+00	2.2055E-10	0.0000E+00	0.0000E+00	4.0256E-03	3.1577E-10
Π_2^*	99.50%	0.0000E+00	3.5677E-10	0.0000E+00	0.0000E+00	4.0054E-03	1.0353E-09
	99.90%	0.0000E+00	9.5019E-10	1.0002E-15	1.0002E-15	9.6766E-10	1.8468E-12
	99.98%	0.0000E+00	1.9823E-09	0.0000E+00	0.0000E+00	9.6689E-10	1.7900E-13

If we consider the cumulative loss probabilities computed by the FCONV method to be exact, the following observations can be made. First, the SCONV method generates exactly the same cumulative loss probabilities as the FCONV method. The relative errors for the TR SCONV method are about $10^{-9} - 10^{-12}$, which are much smaller than the specified threshold of $TOL = 10^{-4}$. For the FFT methods, the relative errors for the FFFT method are the same as those for the SFFT method: both are of order 10^{-15}. The TR SFFT method suffers from significant aliasing errors, which may be as large as 10^{-2}. On the other hand, thanks to the effective aliasing reduction, the EW TR SFFT method has errors of 10^{-9}–10^{-13}, which is much better than the TR SFFT method. Comparing the TR SCONV method and the EW TR SFFT method, for the selected parameters, both methods provide comparable

accuracy: the TR SCONV method is slightly more accurate when γ is small, while the EW TR SFFT method is more accurate when γ is very close to 1.

5.3.3.2 Efficiency

To compare the efficiency, we record the CPU time in seconds required by the different methods to compute the loss probabilities $p[k]$, and compute the speedup of the different methods over the benchmark FCONV method. The results are listed in Table 5.19, Table 5.20, Figure 5.37 and Figure 5.38.

Table 5.19: CPU time to compute the cumulative loss probabilities for the synthetic portfolios

Portfolio	γ	FCONV	SCONV	TR SCONV	FFFT	SFFT	TR SFFT	EW TR SFFT
	95.00%						7.2478E-02	7.5574E-02
	99.00%						7.3006E-02	7.5970E-02
Π_1	99.50%	1.1480E+01	1.0573E-02	2.4508E-03	8.3550E+00	1.2472E+00	1.4293E-01	1.4715E-01
	99.90%						1.4505E-01	1.4601E-01
	99.98%						1.4330E-01	1.4693E-01
	95.00%						4.6687E-02	4.8128E-02
	99.00%						4.6568E-02	4.8182E-02
Π_2	99.50%	2.3397E+01	6.0346E+00	1.2276E-01	8.3001E+00	2.7862E+00	4.6691E-02	4.8197E-02
	99.90%						4.6590E-02	4.8510E-02
	99.98%						8.6040E-02	8.7429E-02
	95.00%						6.1093E-02	6.3298E-02
	99.00%						1.1981E-01	1.2355E-01
Π_1^*	99.50%	1.9534E+01	6.7058E-01	6.9470E-02	8.3285E+00	1.0317E+00	1.2119E-01	1.2319E-01
	99.90%						1.2081E-01	1.2320E-01
	99.98%						1.2041E-01	1.2385E-01
	95.00%						4.6147E-02	4.8056E-02
	99.00%						4.6382E-02	4.9104E-02
Π_2^*	99.50%	2.2968E+01	6.2474E+00	1.5334E-01	8.6662E+00	2.7181E+00	4.6001E-02	4.7835E-02
	99.90%						8.2625E-02	8.3669E-02
	99.98%						8.2738E-02	8.3888E-02

Table 5.20: Speedup to compute the cumulative loss probabilities for the synthetic portfolios

Portfolio	γ	SCONV	TR SCONV	FFFT	SFFT	TR SFFT	EW TR SFFT
	95.00%					1.5839E+02	1.5190E+02
	99.00%					1.5724E+02	1.5111E+02
Π_1	99.50%	1.0858E+03	4.6841E+03	1.3740E+00	9.2042E+00	8.0318E+01	7.8011E+01
	99.90%					7.9143E+01	7.8622E+01
	99.98%					8.0110E+01	7.8131E+01
	95.00%					5.0115E+02	4.8614E+02
	99.00%					5.0243E+02	4.8560E+02
Π_2	99.50%	3.8772E+00	1.9060E+02	2.8189E+00	8.3974E+00	5.0111E+02	4.8545E+02
	99.90%					5.0220E+02	4.8232E+02
	99.98%					2.7194E+02	2.6761E+02
	95.00%					3.1974E+02	3.0860E+02
	99.00%					1.6303E+02	1.5811E+02
Π_1^*	99.50%	2.9129E+01	2.8118E+02	2.3454E+00	1.8934E+01	1.6118E+02	1.5856E+02
	99.90%					1.6169E+02	1.5855E+02
	99.98%					1.6223E+02	1.5772E+02
	95.00%					4.9772E+02	4.7795E+02
	99.00%					4.9519E+02	4.6774E+02
Π_2^*	99.50%	3.6765E+00	1.4979E+02	2.6503E+00	8.4501E+00	4.9930E+02	4.8016E+02
	99.90%					2.7798E+02	2.7451E+02
	99.98%					2.7760E+02	2.7380E+02

Figure 5.37: Comparison of the CPU time for the synthetic portfolios
upper: linear scale; lower: log scale

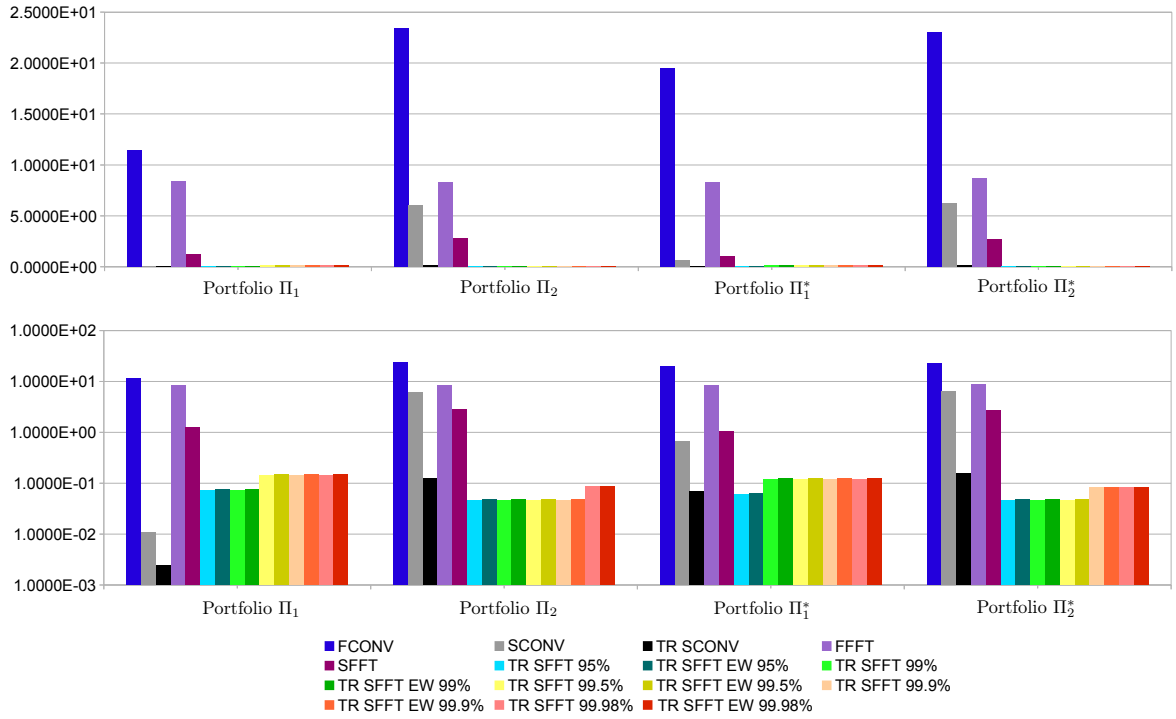
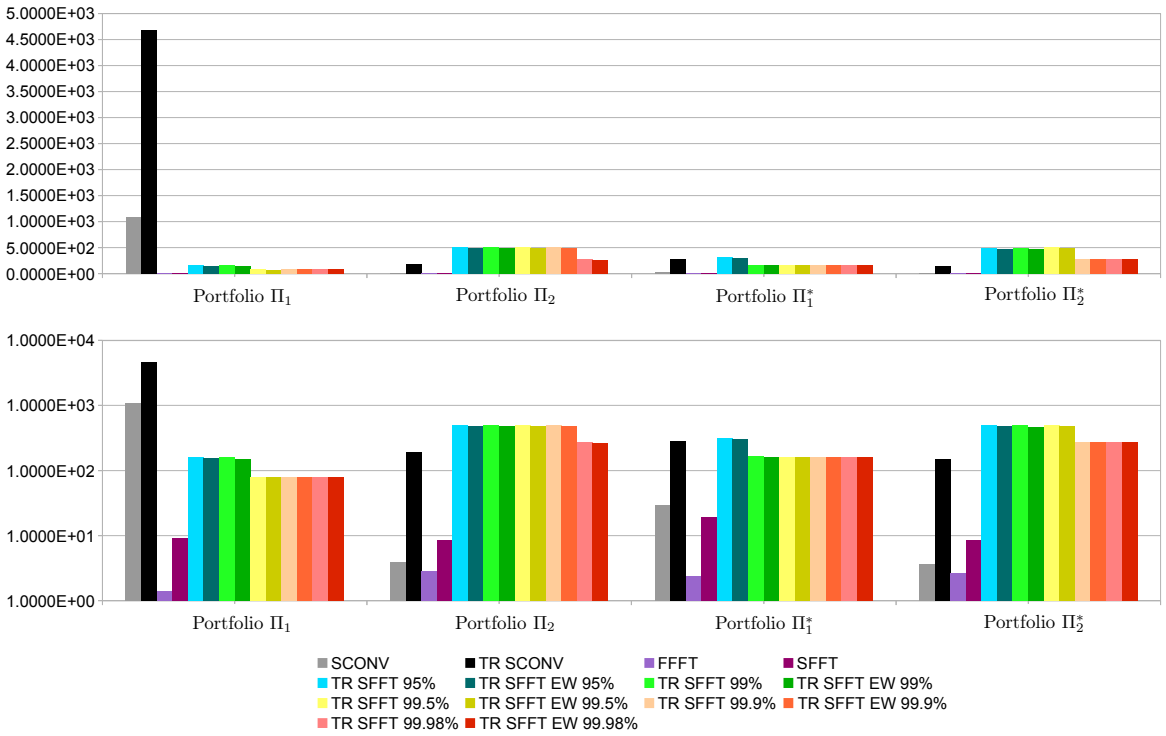


Figure 5.38: Comparison of the speedups for the synthetic portfolios
upper: linear scale; lower: log scale



We make the following observations based on the numerical results.

Speedup of the sparse methods Figure 5.38 and Table 5.20 show us that

- In all testing cases tested, the sparse methods (SCONV and SFFT method) are more efficient than the corresponding full exact methods (FCONV and FFFT). In addition, the truncated sparse methods (TR SCONV and (EW) TR SFFT) are faster than the corresponding sparse methods.

- The speedups for the TR SFFT and EW TR SFFT methods are very close in the cases tested, which indicates that the additional computational cost incurred by EW TR SFFT is negligible.

- Compared with the SFFT method, the SCONV method is more efficient when the vector of conditional probabilities is very sparse ($C = 2$), but slightly less efficient when the vector is less sparse ($C = 18$). Similar observation holds for the comparison of the TR SCONV method and the (EW) TR SFFT method for tested portfolios.

- For a fixed portfolio, as the confidence level γ increases, the speedup of the (EW) TR SFFT method behaves like a decreasing step function. For example, for portfolio Π_1, the speedup of (EW) TR SFFT method for $\gamma = 99\%$ is almost the same as that for $\gamma = 95\%$. However, the speedup decreases as γ increases to 99.5%, but thereafter remains the same level for $\gamma = 99.9\%$ and 99.98%. We believe this is due to the length, $\bar{K}$, of the truncated vectors of conditional probabilities. A larger confidence level α leads to a larger $\text{VaR}_\gamma = k_\gamma \delta$, and consequently a larger k_γ. Since $\bar{K}$ is determined by (5.3.17), it is an increasing step function of γ. Indeed, as shown in Table 5.21, for portfolio Π_1, $\bar{K} = 2^{12} = 4096$ for $\gamma = 95\%$ and 99%, and $\bar{K} = 2^{13} = 8192$ for $\gamma = 99.5\%$, 99.9% and 99.98%. The same observation can be made for other portfolios. These observations show that the efficiency of the (EW) TR SFFT methods depends mainly on $\bar{K}$.[17]

[17] This conclusion also applies to the SFFT method, although we did not examine the impact of K on the SFFT method.

Table 5.21: k_γ and $\bar{K}$ for (EW) TR SFFT method

γ	Π_1		Π_2		Π_1^*		Π_2^*	
	k_γ	$\bar{K}$	k_γ	$\bar{K}$	k_γ	$\bar{K}$	k_γ	$\bar{K}$
95.00%	3536	2^{12}	631	2^{10}	3775	2^{12}	781	2^{10}
99.00%	4046	2^{12}	782	2^{10}	4319	2^{13}	941	2^{10}
99.50%	4191	2^{13}	842	2^{10}	4525	2^{13}	1003	2^{10}
99.90%	4584	2^{13}	968	2^{10}	4959	2^{13}	1137	2^{11}
99.98%	4846	2^{13}	1084	2^{11}	5351	2^{13}	1261	2^{11}

When VaR_γ is very large, $\bar{K}$, could be very close to K. For example, for some portfolios, percentage VaR_γ could be larger than 0.5, then $\bar{K} = K$, and no truncation could be made to improve the SFFT method. Therefore, this could be a major factor which discourages people to rank (EW) TR SFFT method over the TR SCONV method.

Impact of sparsity. Based on Table 5.19, for each method, we compute the ratio of the CPU time required for $C = 18$ and for the similar portfolio with $C = 2$. The results are shown in Table 5.22.

Table 5.22: Ratio of CPU time for similar portfolios with $C = 18$ and $C = 2$

Portfolio	γ	FCONV	SCONV	TR SCONV	FFFT	SFFT	TR SFFT	EW TR SFFT
Homo-geneous	95.00%	2.04	570.78	50.09	0.99	2.23	0.64	0.64
	99.00%						0.64	0.63
	99.50%						**0.33**	**0.33**
	99.90%						**0.32**	**0.33**
	99.98%						0.60	0.60
Hetero-geneous	95.00%	1.18	9.32	2.21	1.04	2.63	0.76	0.76
	99.00%						**0.39**	**0.40**
	99.50%						**0.38**	**0.39**
	99.90%						0.68	0.68
	99.98%						0.69	0.68

Based on the results in Table 5.22, we make the following observations.

- For the convolution methods, if the portfolios are homogeneous, it takes longer to compute the cumulative loss probabilities for $C = 18$ than for $C = 2$. Moreover, the impact of sparsity on the

SCONV method and the TR SCONV method is much greater than on the FCONV method. This is as expected since larger C makes the vectors of conditional probabilities less sparse, which makes sparse convolution less effective. For heterogeneous portfolios, a similar observation holds, except the impact of sparsity is less significant in this case, because when a portfolio is inhomogeneous, nonzero elements are not aligned, whence the number of nonzero elements in $\tilde{\boldsymbol{p}}_n^{\boldsymbol{z}}$ increases more quickly than in the homogeneous case. The difference in the speed of increase of the number of nonzero elements in $\tilde{\boldsymbol{p}}_n^{\boldsymbol{z}}$ is more significant for $C = 2$ (we will see this point shortly in connection with Table 5.23), since, when $C = 18$, the vectors of conditional probabilities are less sparse, so the number of nonzero elements in $\tilde{\boldsymbol{p}}_n^{\boldsymbol{z}}$ increases rapidly in any case.

- Sparsity has almost no impact on the FFFT method. The SFFT method runs about twice as fast for $C = 2$ compared to $C = 18$ for both the homogeneous and inhomogeneous portfolios. For the TR SFFT and EW TR SFFT methods, if we exclude the cases where $\bar{K}$ is different for different values of C (the case where the ratio of CPU times are in bold in Table 5.22), then the ratios of CPU time are around 0.60-0.64 for homogeneous portfolios and around 0.68-0.76 for the inhomogeneous portfolios, indicating that the (EW) TR FFT method runs faster for $C = 18$ than for $C = 2$. The reason for this is that $\bar{K}$ is smaller for $C = 18$ than for $C = 2$, as can be seen from Table 5.21.

- For the FFT methods, the impact of sparsity is less significant than for the convolution methods, which follows from the observation that the ratios for the FFT methods are closer to 1 than are the ratios for the convolution methods.

Impact of heterogeneity. Based on Table 5.19, for each method and each of $C = 2$ and $C = 18$, we compute the ratio of the CPU time required for inhomogeneous portfolios over the homogenous portfolios for the same value of C. The results are shown in Table 5.23.

Table 5.23: Ratio of CPU time: heterogeneous over homogenous

C	γ	FCONV	SCONV	TR SCONV	FFFT	SFFT	TR SFFT	EW TR SFFT
2	95.00%	1.70	63.43	28.35	1.00	0.83	0.84	0.84
	99.00%						**1.64**	**1.63**
	99.50%						0.85	0.84
	99.90%						0.83	0.84
	99.98%						0.84	0.84
18	95.00%	0.98	1.04	1.25	1.04	0.98	0.99	1.00
	99.00%						1.00	1.02
	99.50%						0.99	0.99
	99.90%						**1.77**	**1.72**
	99.98%						0.96	0.96

Based on the results in Table 5.23, we make the following observations:

- Except for the SCONV and TR SCONV methods in cases where the vectors of conditional probabilities are very sparse ($C = 2$), the impact of heterogeneity is not very significant, since the ratios of CPU time are within the range $[0.83, 1.77]$.

- For convolution methods, when $C = 2$, it takes longer to compute the cumulative loss probabilities for inhomogeneous portfolios than for homogeneous portfolios, and the impact of heterogeneity on the SCONV and TR SCONV methods is much more significant than on the FCONV method. When $C = 18$, the impact of heterogeneity is not significant. For the SCONV and TR SCONV methods, notice that (5.1.27) shows that the CPU time depends on both the number of nonzero elements, C, and the positions of the nonzero elements. When $C = 2$, the vectors of conditional probabilities are very sparse. For homogeneous portfolios, nonzero elements are aligned for all obligors. This is the best case, as explained in Subsubsection 5.1.2.3, that generates the minimum number of new nonzero elements after each convolution operation. However, for inhomogeneous portfolios, the nonzero elements are not aligned, hence the number of nonzero elements increases more quickly. Therefore, it takes much more CPU time for the SCONV and TR SCONV methods to compute the cumulative loss probabilities for inhomogeneous portfolios. However, when $C = 18$, the vectors of conditional probabilities are no longer very sparse, and even for the homogeneous portfolios, vectors of conditional probabilities are not perfectly equally spaced, which leads to a rapid increase in the number of nonzero elements. Therefore, the impact of the difference in the

positions of the nonzero elements between homogeneous portfolios and inhomogeneous portfolios is much less significant, hence the impact of heterogeneity on the CPU time becomes very small.

- For FFT methods, the impact of heterogeneity is not very significant. Heterogeneity has almost no impact on the FFFT method. For sparse FFT methods (SFFT, TR SFFT and EW TR SFFT), if we exclude cases where $\bar{K}$ is different for the corresponding homogeneous portfolio and inhomogeneous portfolio (the case where the ratio of CPU times are in bold in Table 5.23), then the ratios of CPU times are quite stable around 0.84 for $C = 2$, indicating that the sparse FFT methods run a little faster for inhomogeneous portfolios. The reason for this is that the bit reversals of the conditional probabilities vectors are closer to the best case we discussed in Subsection 5.2.3.4. For $C = 18$, the ratios of CPU times are quite stable around 1.00, implying that heterogeneity has almost no impact on sparse FFT methods as well. The reason for this is similar to that for convolution methods. That is, when the conditional probabilities vectors are not very sparse, the impact of the difference in positions of nonzero elements between homogeneous portfolios and inhomogeneous portfolios is much less significant.

- Compared with sparse convolution methods, the efficiency of the sparse FFT methods is much less sensitive to heterogeneity of portfolios when the conditional probabilities vectors are very sparse ($C = 2$). When $C = 18$, the impact of heterogeneity is very limited for both sparse convolution methods and sparse FFT methods.

5.3.4 Comparison: MC and Exact Methods

In this subsection, we compare some sparse convolution methods to the MC methods for some synthetic portfolios.

We divide our testing cases into four groups depending on the homogeneity and the size of rating classes. Within each group, we vary the size of portfolios to check the impact of portfolio size on methods tested. For all portfolios, all obligors are in the highest rating class. In order to avoid discretization errors, the loss-given-credit-event, LGC_n^c, and the EAD, EAD_n, are specified such that each of L_n^c is on the discretization grid of loss with length of $K = 2^{16} = 65536$ constructed by (5.1.13) and (5.1.14). The following table summarizes the portfolios used in our tests.

Table 5.24: Testing portfolios for exact methods against MC method

Group	Homogeneity	C	$c(n)$	N	EAD_n	LGC_n^c	β_n
1	Homogeneous	2	2^0	$2^t, \quad t = 2, \ldots, 12$	1	$\frac{c}{2^0}$	0.5
2		17	2^4	$2^t, \quad t = 2, \ldots, 10$		$\frac{c}{2^4}$	
3	Heterogeneous	2	2^0	$2^t, \quad t = 2, \ldots, 12$	$\widetilde{\text{EAD}^3}_n$	$\frac{c}{2^0}$	$\text{Unif}(-1, 1)$
4		17	2^4	$2^t, \quad t = 2, \ldots, 10$	$\widetilde{\text{EAD}^4}_n$	$\frac{c}{2^0}$	

The credit migration matrices used in tests in this subsection are the migration matrix in Table 5.16 for $C = 2$, and the submatrix of the migration matrix in Table 5.17 for $C = 17$.

For Groups 3 and 4, we would like to have the histogram of the EAD similar to the PDF of the exponential distribution, whence the portfolio has more obligors with smaller EAD and less obligors with larger EAD. Meanwhile, to make each L_n^c fall on the discretization grid, we set $\widetilde{\text{EAD}^{3,4}}_n \in \mathbb{N}^+$, $\sum_{n=0}^{N-1} \widetilde{\text{EAD}^3}_n = 2^{16}$, and $\sum_{n=0}^{N-1} \widetilde{\text{EAD}^4}_n = 2^{12}$. Specifically, EAD's in Group 3 and 4 are set as follows:

Step 1 Generate $X_n \sim \exp(10)$, $n = 0, \ldots, N-1$;

Step 2 For $n = 0, \ldots, N-1$, calculate

$$Y_n = \begin{cases} \frac{X_n}{\sum_{n=0}^{N-1} X_n} \times 2^{16}, & \text{Group 3} \\ \frac{X_n}{\sum_{n=0}^{N-1} X_n} \times 2^{12}, & \text{Group 4} \end{cases};$$

Step 3 For $n = 1, \ldots, N-1$, calculate

$$\widetilde{\text{EAD}^{3,4}}_n = \begin{cases} \lceil Y_n \rceil; & \text{if } Y_n < \text{medium}(\boldsymbol{Y}) \\ \lfloor Y_n \rfloor; & \text{if } Y_n \geq \text{medium}(\boldsymbol{Y}) \end{cases},$$

where $\text{medium}(\boldsymbol{Y})$ is the medium of a sample of $\boldsymbol{Y}$;

Step 4 Calculate

$$\widetilde{\text{EAD}^3}_0 = 2^{16} - \sum_{n=1}^{N-1} \widetilde{\text{EAD}^3}_n,$$
$$\widetilde{\text{EAD}^4}_0 = 2^{12} - \sum_{n=1}^{N-1} \widetilde{\text{EAD}^4}_n;$$

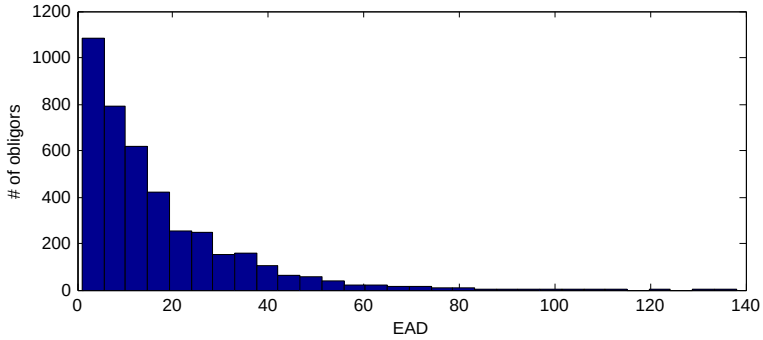
Step 5 If $\widetilde{\text{EAD}^{3,4}}_0 > 0$, stop, else repeat Step 1 to 5.

As an illustration, Figure 5.39 shows the histogram of EADs for the portfolio with size 2^{12} in Group

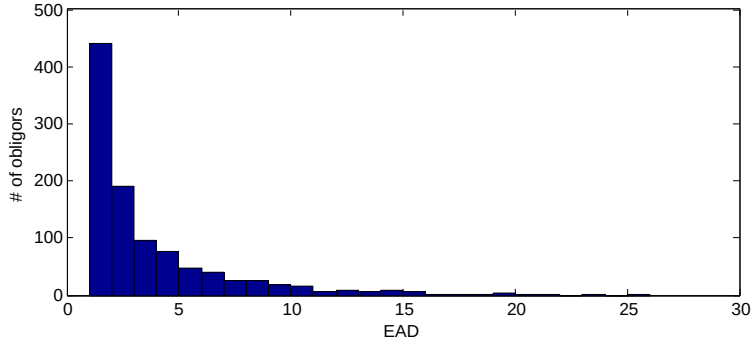
3 and for the portfolio with size 2^{10} in Group 4.

Figure 5.39: Histogram of EADs

(a) Histogram of EADs for the portfolio with size 2^{12} in Group 3



(b) Histogram of EADs for the portfolio with size 2^{10} in Group 4



For each constructed portfolio, we compute the cumulative loss probabilities

$$\hat{p}^{method}[k] \doteq \mathbb{P}\{\mathcal{L} \leq l_k\}$$

for $k = 0, \ldots, K-1$ using the SCONV, TR SCONV and MC method, respectively, then we can compute VaR at the confidence levels $\gamma = 95\%,\ 99\%,\ 99.9\%$ and 99.98% by $\text{VaR}_\gamma^{method} = k_\gamma^{method}\delta$, where $k_\gamma^{method} = \min\{k :\ \hat{p}^{method}[k] \geq \gamma\}$. Also, using the SCONV method as the benchmark, we compute the loss probability for the TR SCONV and MC methods, $\hat{p}^{method}\left[k_\gamma^{SCONV}\right]$, at the quantile $\text{VaR}_\gamma^{SCONV}$. Notice that, for the MC method, we vary the sample size from $2^2 = 4$ to $2^{14} = 16384$, and compare the results for each sample size. For the TR SCONV method, we let the threshold for errors in the cumulative loss probabilities be $Tol = 10^{-4}$ and set ϵ by (5.1.51).

5.3.4.1 Accuracy

To assess the accuracy, using the SCONV method as the benchmark, we first compare the loss probability at VaR computed by the TR SCONV and MC methods by computing the absolute difference

$$\delta_{\gamma}^{Prob,method} = \left| \hat{p}^{method}\left[k_{\gamma}^{SCONV}\right] - \hat{p}^{SCONV}\left[k_{\gamma}^{SCONV}\right] \right|.$$

The numerical results are presented in Figures 5.40 - 5.47. We then compare VaR computed by the TR SCONV and MC methods by computing the absolute difference

$$\delta_{\gamma}^{VaR,method} = \left| \mathrm{VaR}_{\gamma}^{method} - \mathrm{VaR}_{\gamma}^{SCONV} \right|.$$

The numerical results are presented in Figures 5.48 - 5.55.

From the numerical results, we make following observations:

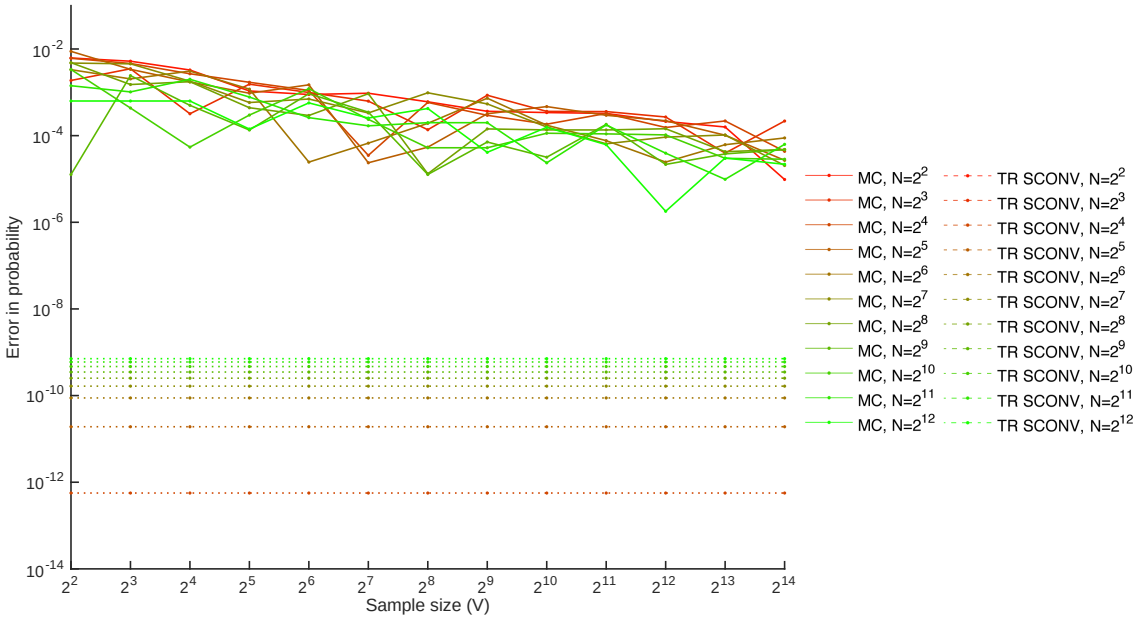
- Figures 5.40 - 5.47 show that, for all tested portfolios, $\delta_{\gamma}^{Prob,TR\ SCONV}$ is much smaller than $\delta_{\gamma}^{Prob,MC}$ and the predetermined the error bound $Tol = 10^{-4}$, which shows that the TR SCONV method generates much more accurate loss probabilities than the MC method. Also note that, due to the random nature of the MC method, $\delta_{\gamma}^{Prob,MC}$ fluctuates randomly (but with a downward trend with the sample size), while the $\delta_{\gamma}^{Prob,TR\ SCONV}$ is deterministic and constant, which indicates that the TR SCONV method enjoys a more stable error control in loss probabilities than the MC method.

- Figures 5.40 - 5.47 show that, for all tested portfolios, $\delta_{\gamma}^{Prob,MC}$ tends to decrease as the sample size increases. That is, the accuracy in the loss probabilities computed by the MC method tends to improve as the sample size increases. The reason for this is that the variance of the MC estimates deceases as the sample size increases.

- Figures 5.48 - 5.55 show that $\delta_{\gamma}^{VaR,TR\ SCONV}$ is zero for most of the testing portfolios, while $\delta_{\gamma}^{VaR,MC}$ is generally nonzero, especially for small sample sizes. Even in the case where $\delta_{\gamma}^{VaR,TR\ SCONV}$ is nonzero, it is usually smaller than $\delta_{\gamma}^{VaR,MC}$. This indicates that the TR SCONV method is more accurate at computing VaR than the MC method. The portfolio loss distribution in Merton's model is discontinuous and piecewise constant, hence a small fluctuation in the confidence level might have no impact on VaR. Therefore, even though the errors in the loss probability in both the MC and TR SCONV methods are not zero, small errors in the loss probability might lead to zero error in VaR.

Figure 5.40: Comparison of the errors in the loss probability at VaR computed by the TR SCONV and MC methods
Portfolios in Group 1

(a) $\gamma = 95\%$



(b) $\gamma = 99\%$

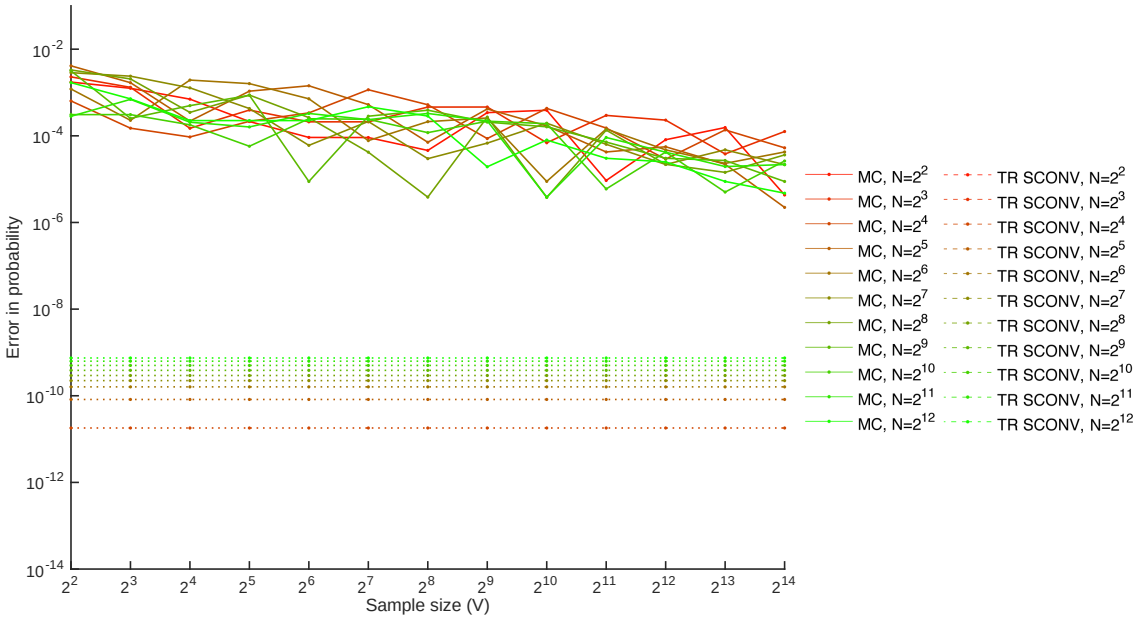
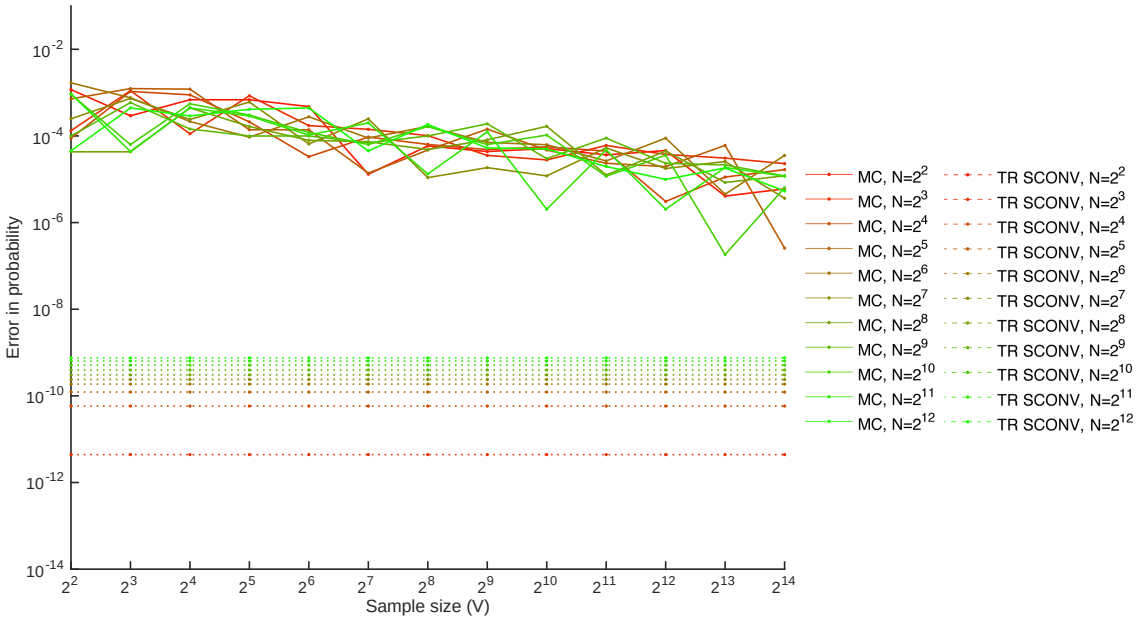


Figure 5.41: Comparison of the errors in the loss probability at VaR computed by the TR SCONV and MC methods
Portfolios in Group 1 (Cont'd)

(a) $\gamma = 99.9\%$



(b) $\gamma = 99.98\%$

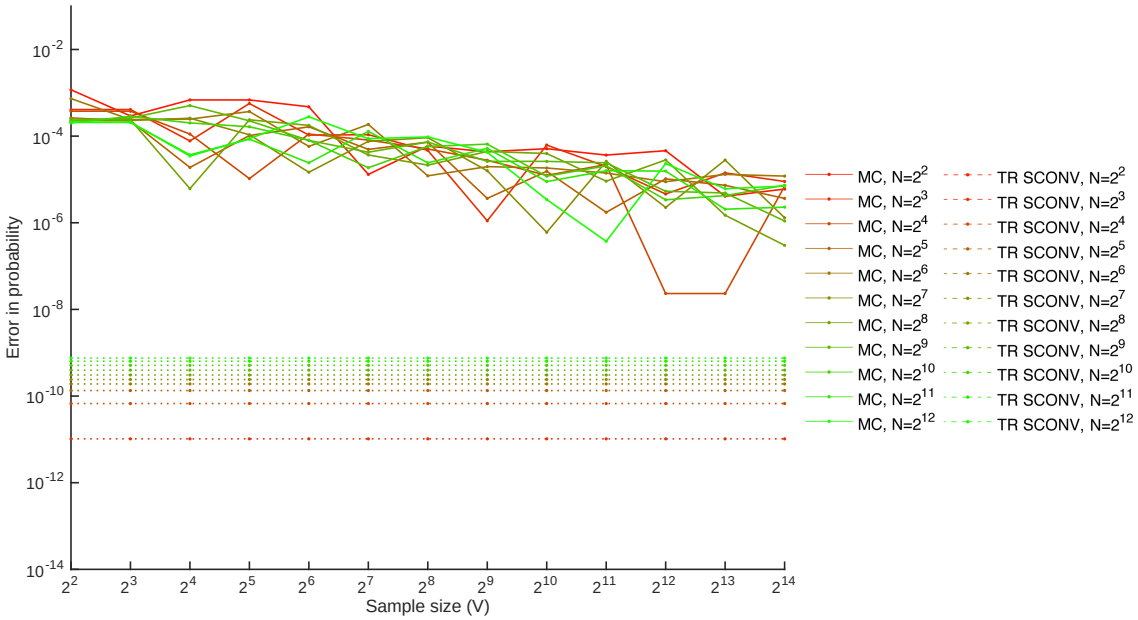
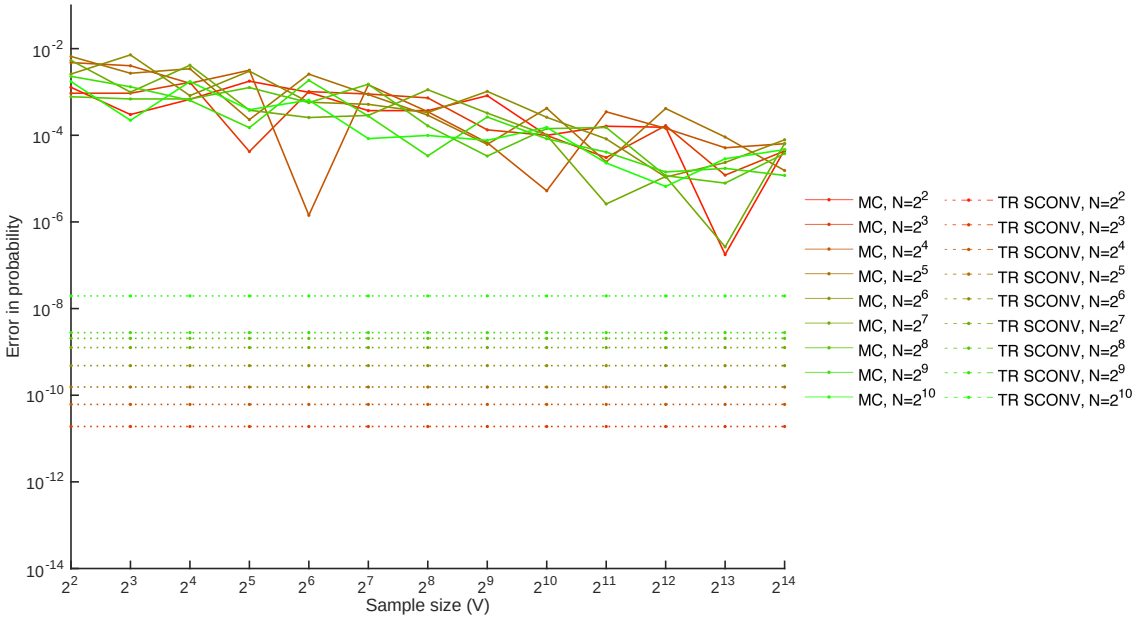


Figure 5.42: Comparison of the errors in the loss probability at VaR computed by the TR SCONV and MC methods
Portfolios in Group 2

(a) $\gamma = 95\%$



(b) $\gamma = 99\%$

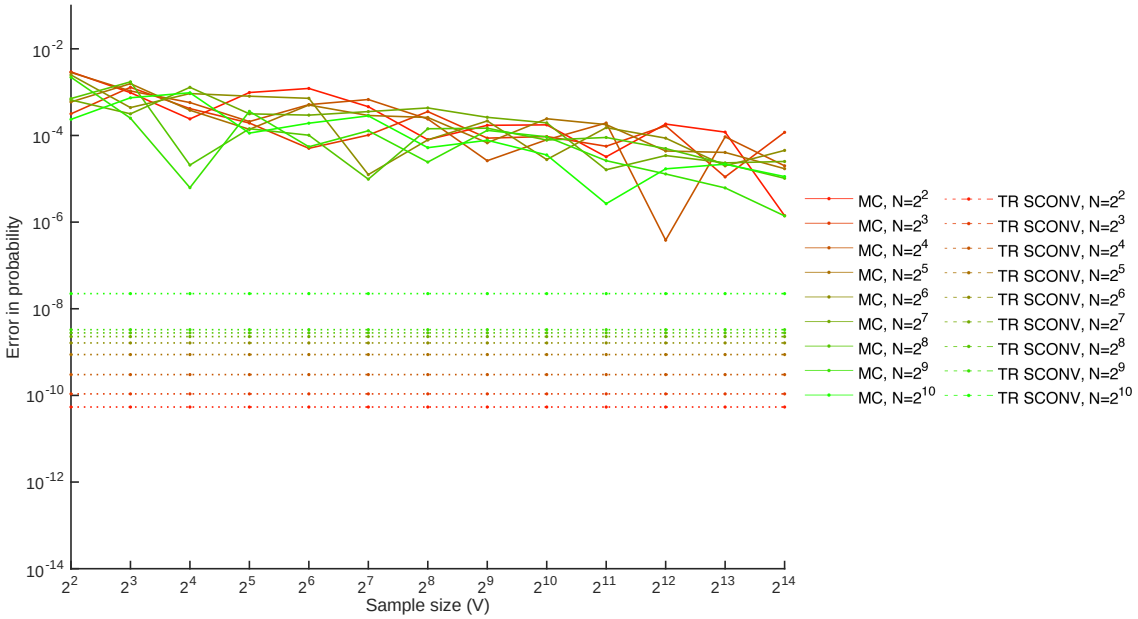
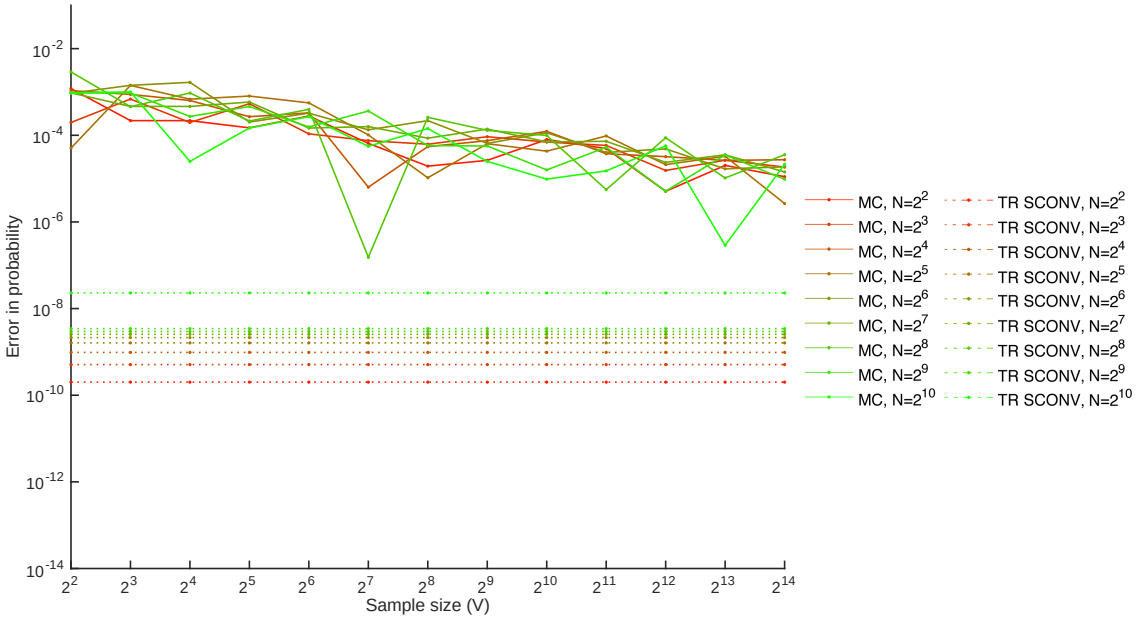


Figure 5.43: Comparison of the errors in the loss probability at VaR computed by the TR SCONV and MC methods
Portfolios in Group 2 (Cont'd)

(a) $\gamma = 99.9\%$



(b) $\gamma = 99.98\%$

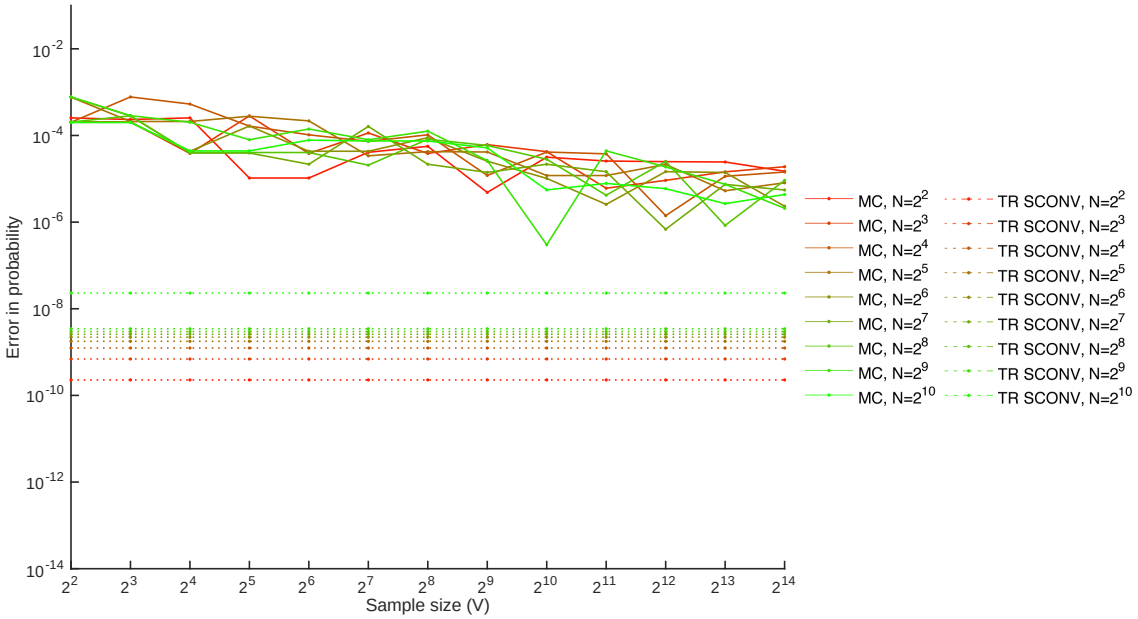
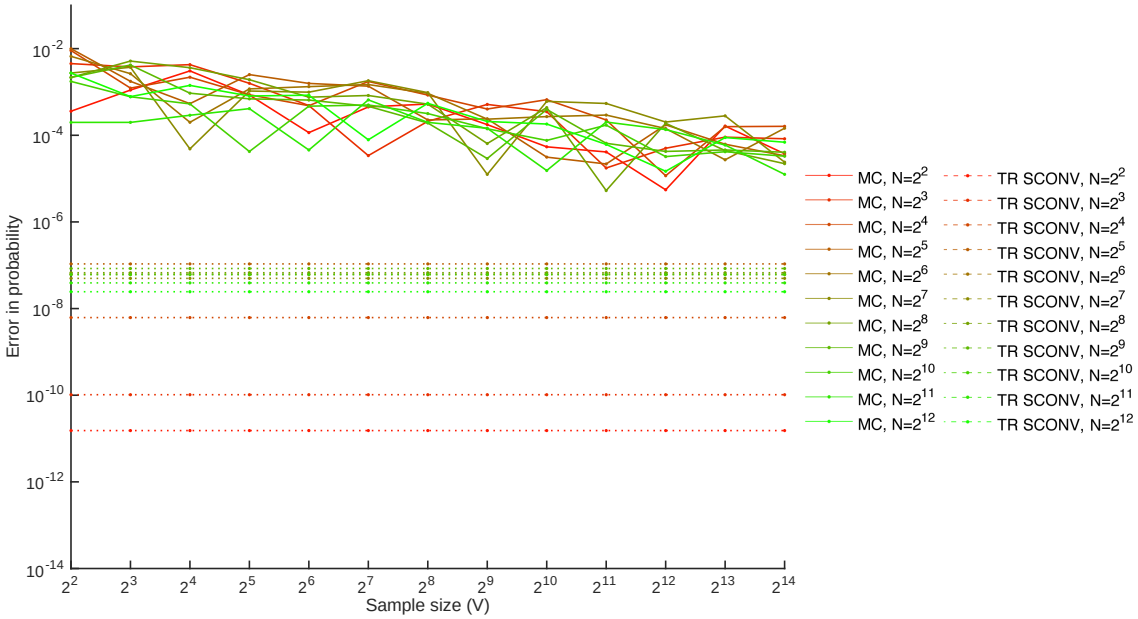


Figure 5.44: Comparison of the errors in the loss probability at VaR computed by the TR SCONV and MC methods
Portfolios in Group 3

(a) $\gamma = 95\%$



(b) $\gamma = 99\%$

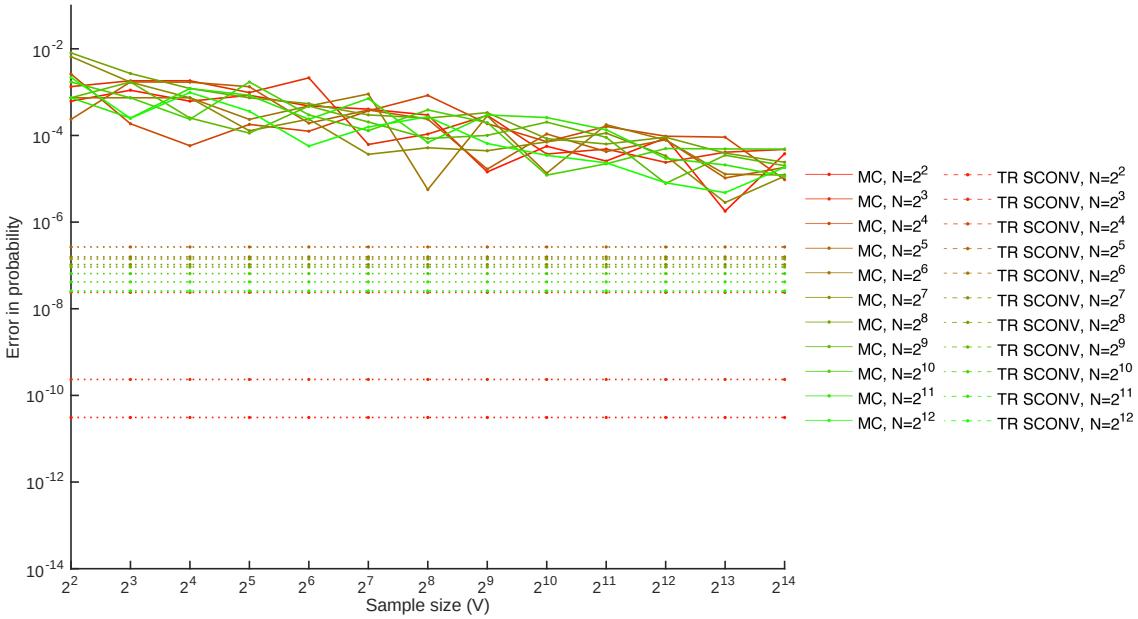
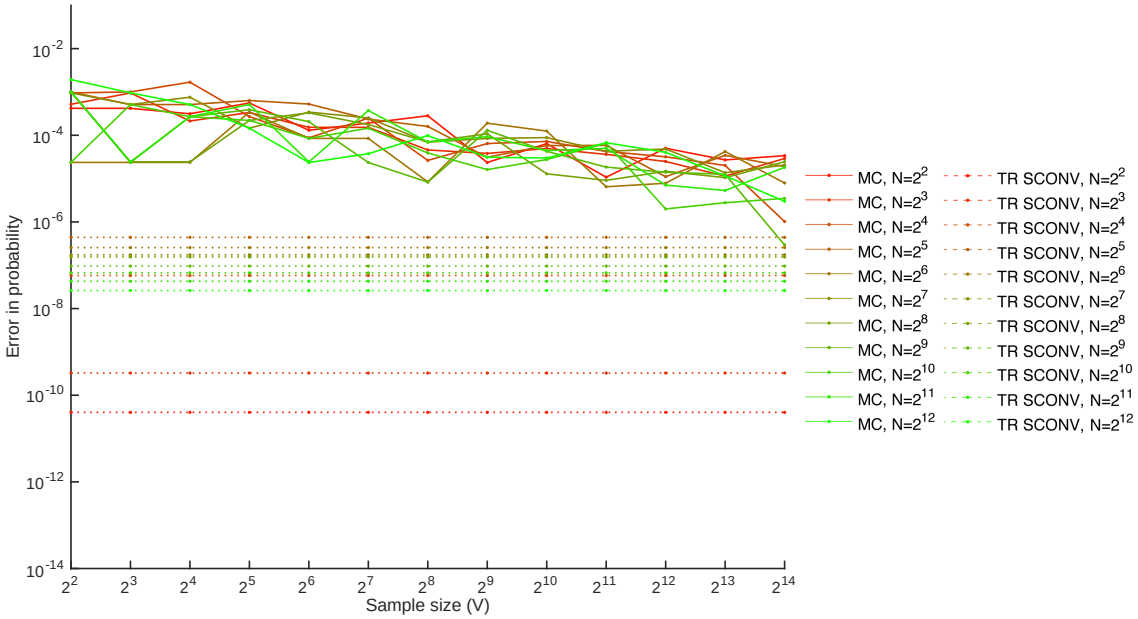


Figure 5.45: Comparison of the errors in the loss probability at VaR computed by the TR SCONV and MC methods
Portfolios in Group 3 (Cont'd)

(a) $\gamma = 99.9\%$



(b) $\gamma = 99.98\%$

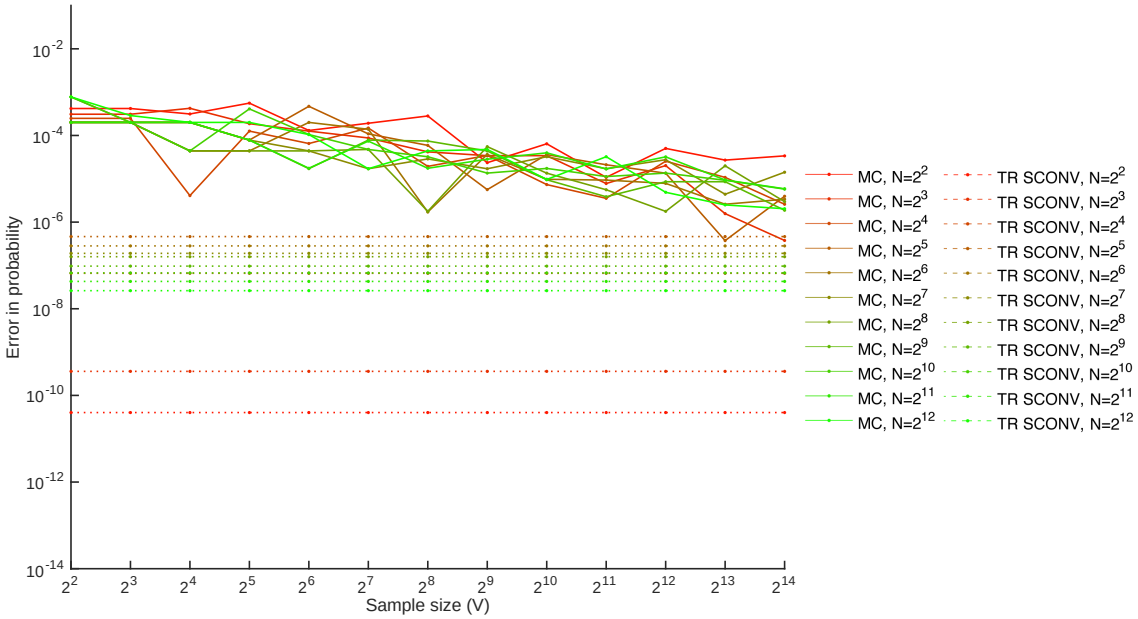
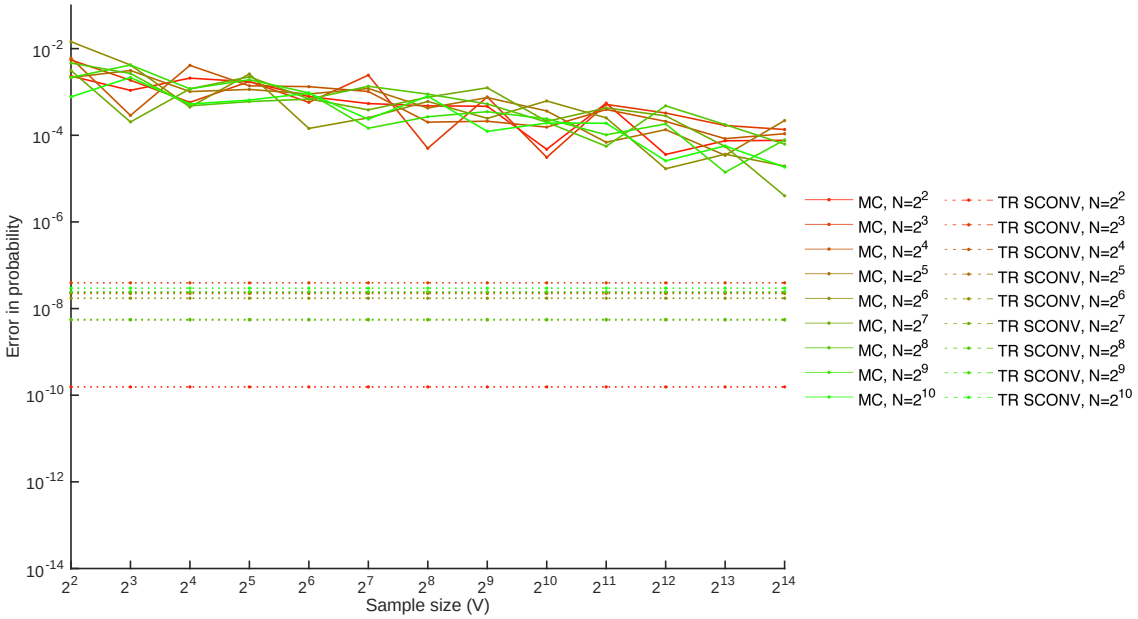


Figure 5.46: Comparison of the errors in the loss probability at VaR computed by the TR SCONV and MC methods
Portfolios in Group 4

(a) $\gamma = 95\%$



(b) $\gamma = 99\%$

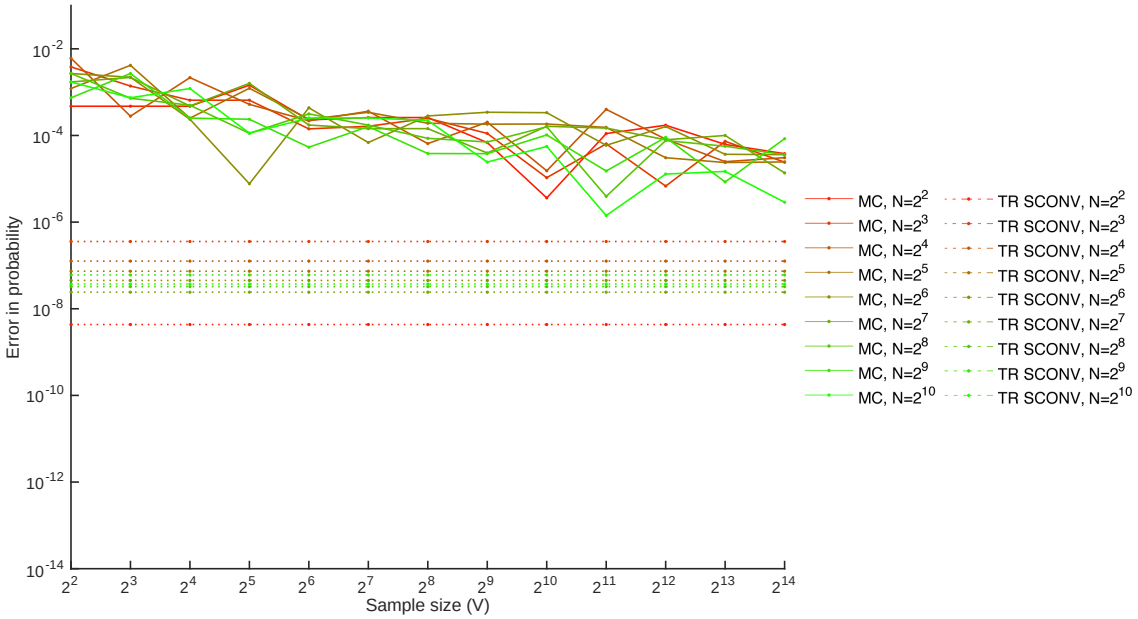
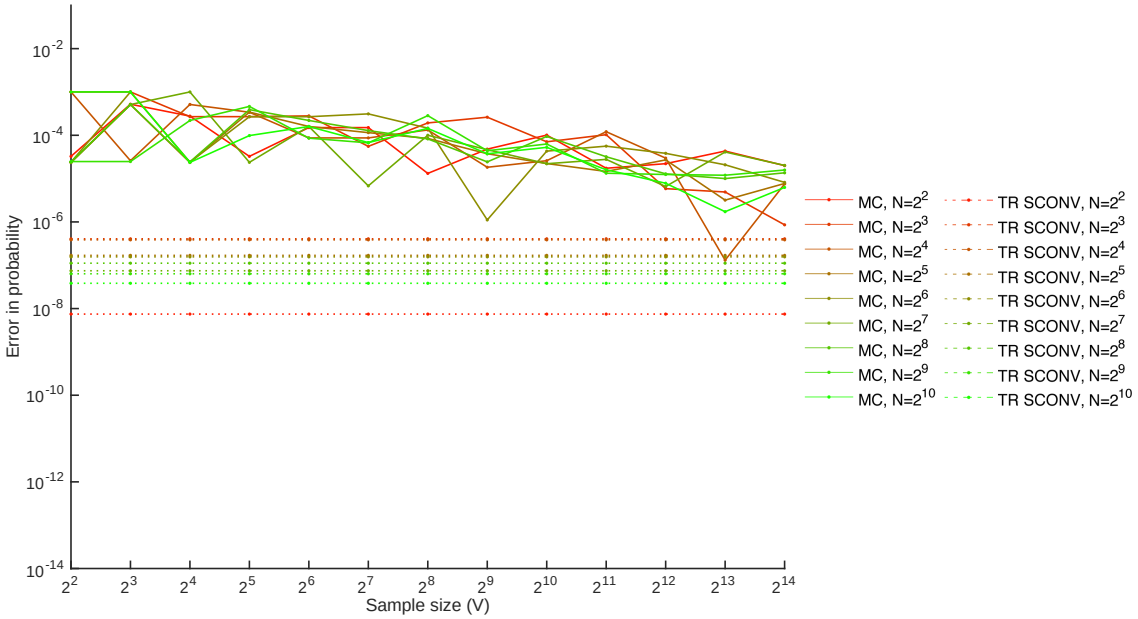


Figure 5.47: Comparison of the errors in the loss probability at VaR computed by the TR SCONV and MC methods
Portfolios in Group 4 (Cont'd)

(a) $\gamma = 99.9\%$



(b) $\gamma = 99.98\%$

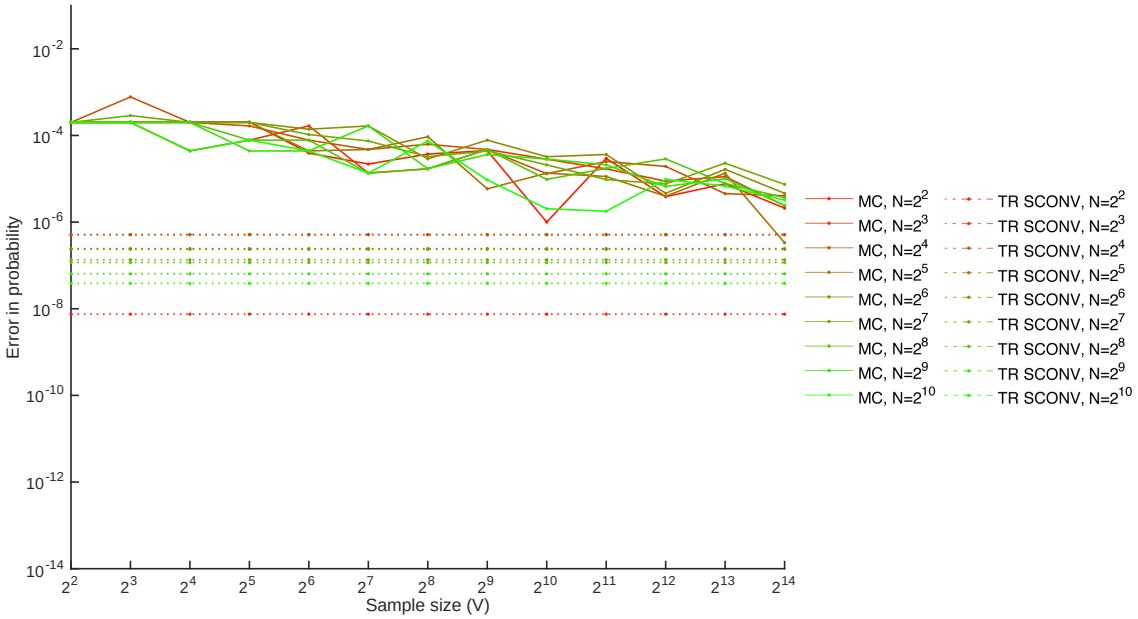
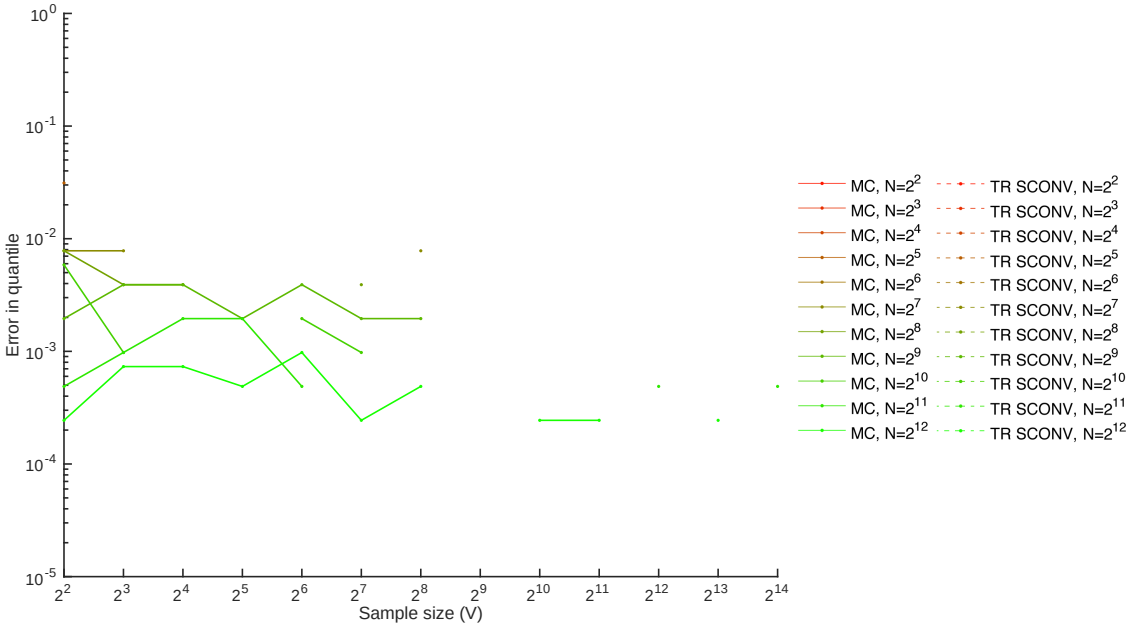


Figure 5.48: Comparison of the errors in VaR computed by the TR SCONV and MC methods
Portfolios in Group 1

(a) $\gamma = 95\%$



(b) $\gamma = 99\%$

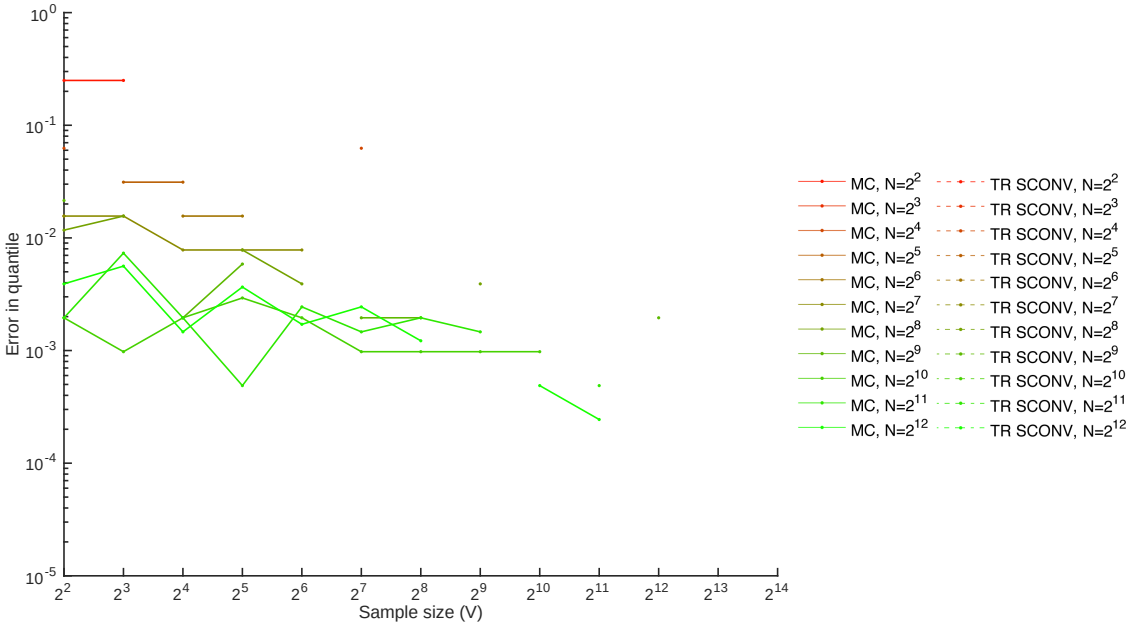
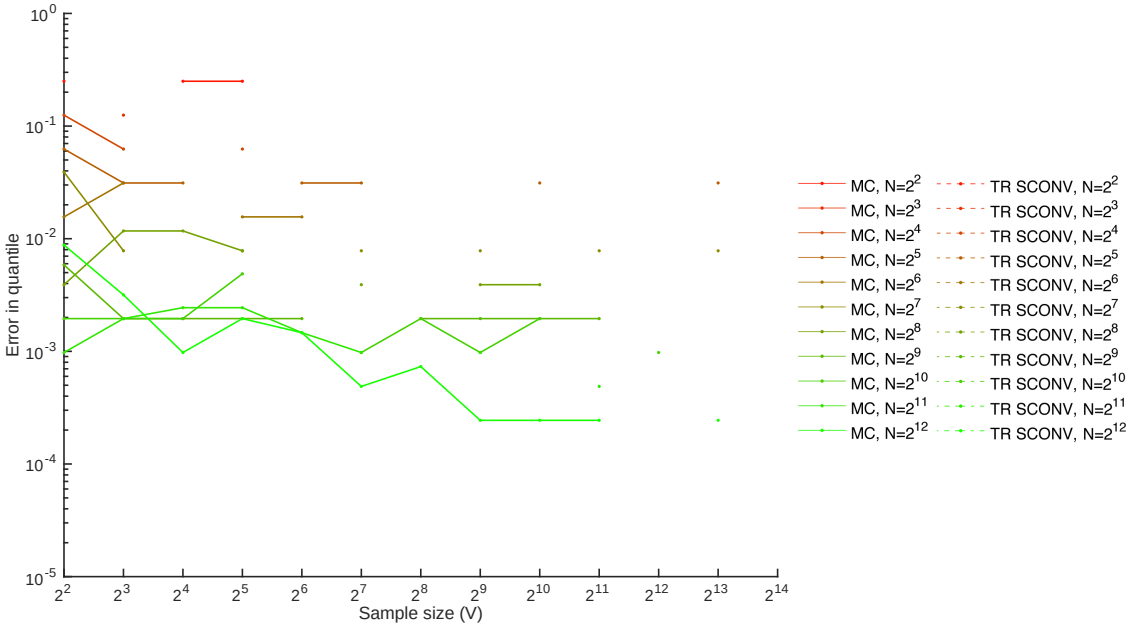


Figure 5.49: Comparison of the errors in VaR computed by the TR SCONV and MC methods *Portfolios in Group 1* (Cont'd)

(a) $\gamma = 99.9\%$



(b) $\gamma = 99.98\%$

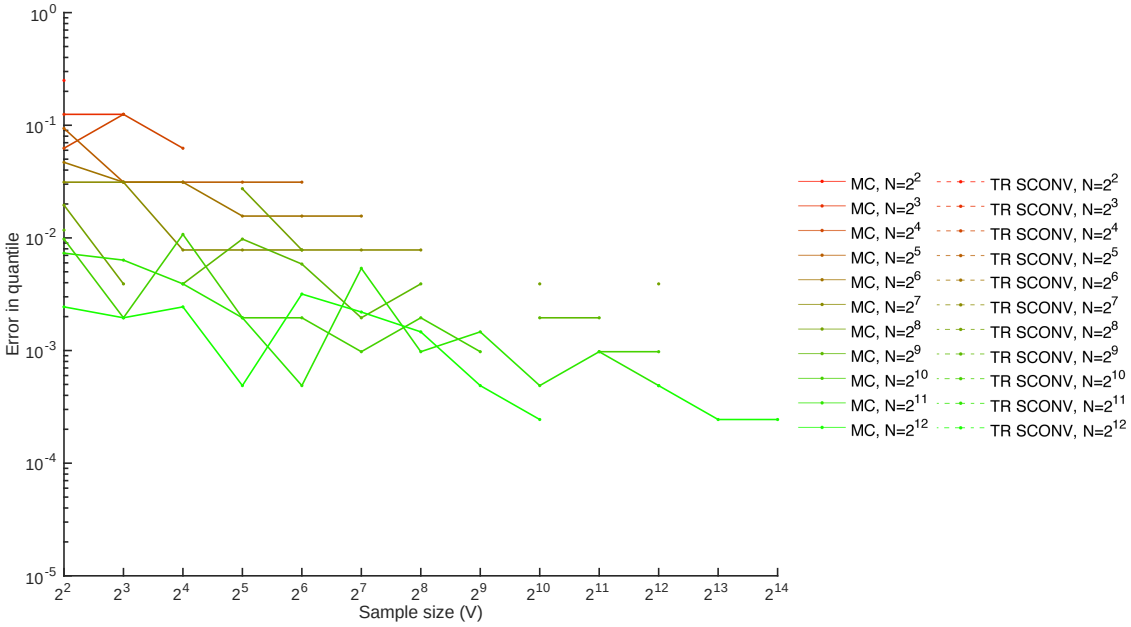
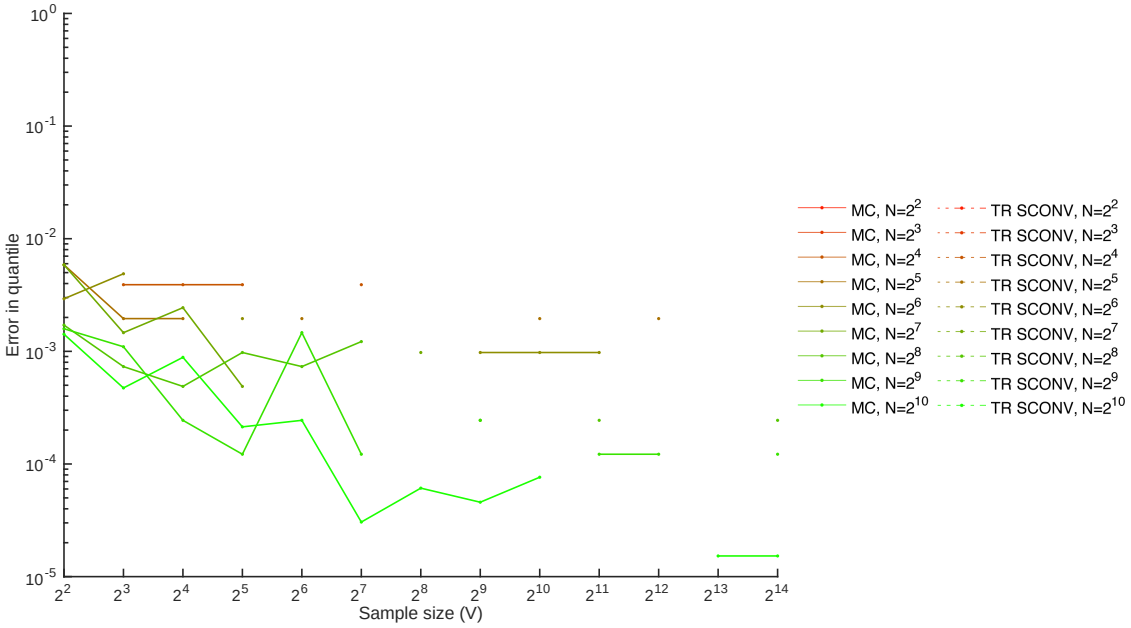


Figure 5.50: Comparison of the errors in VaR computed by the TR SCONV and MC methods
Portfolios in Group 2

(a) $\gamma = 95\%$



(b) $\gamma = 99\%$

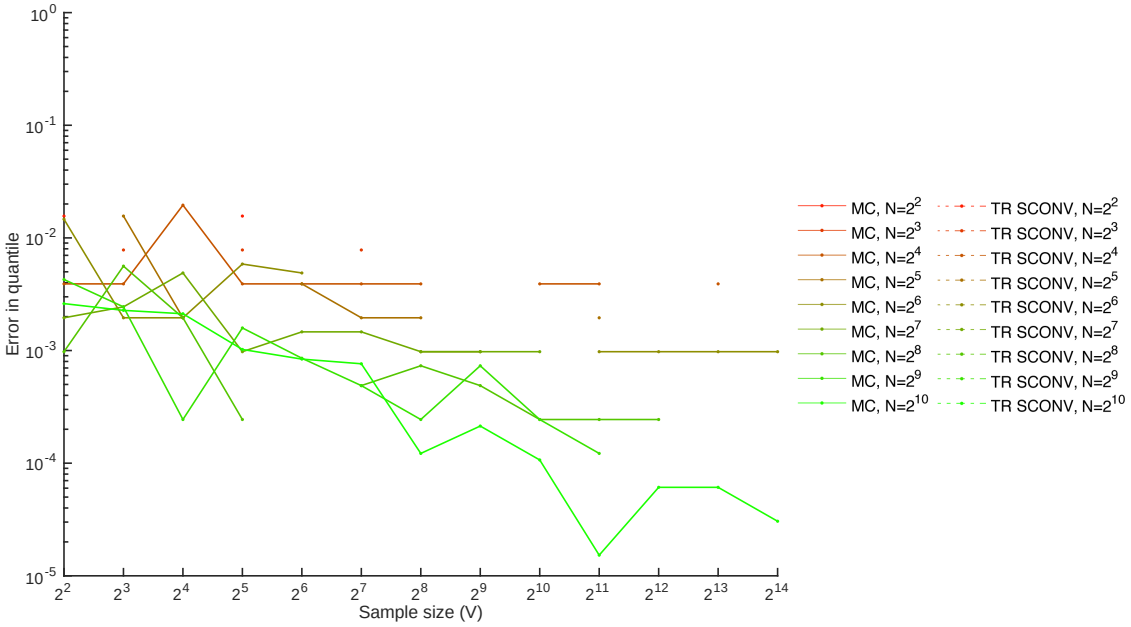
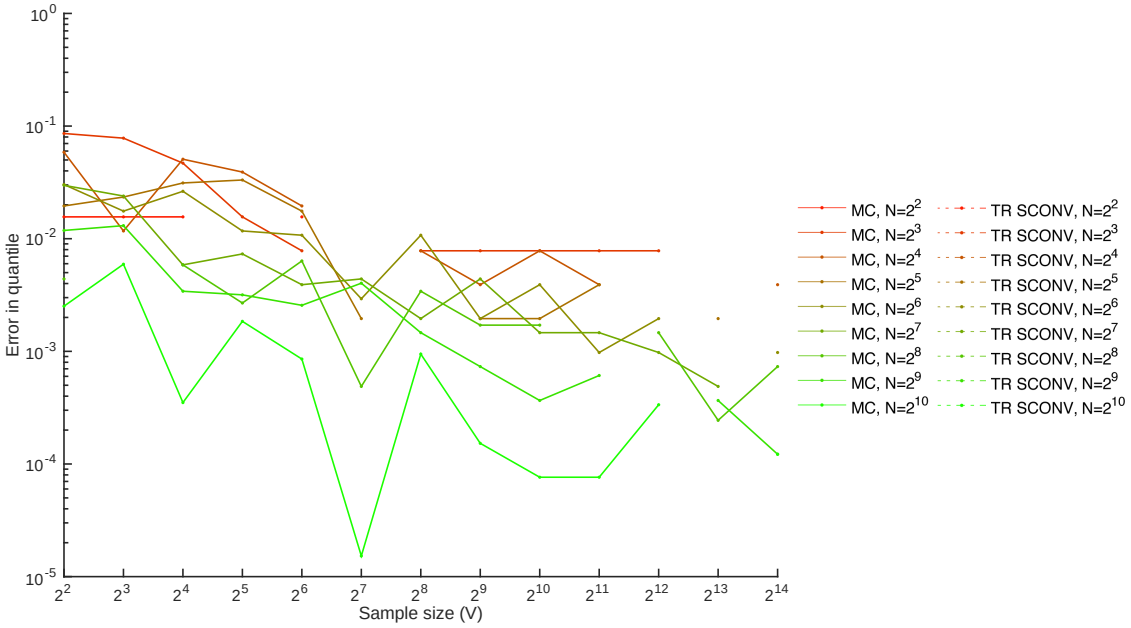


Figure 5.51: Comparison of the errors in VaR computed by the TR SCONV and MC methods *Portfolios in Group 2* (Cont'd)

(a) $\gamma = 99.9\%$



(b) $\gamma = 99.98\%$

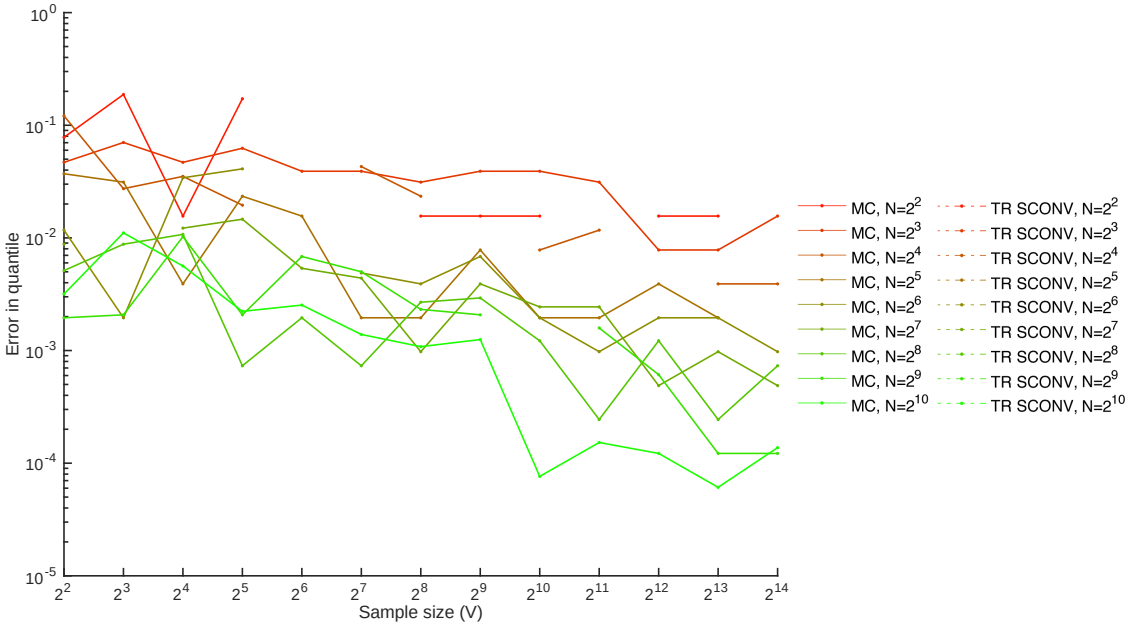
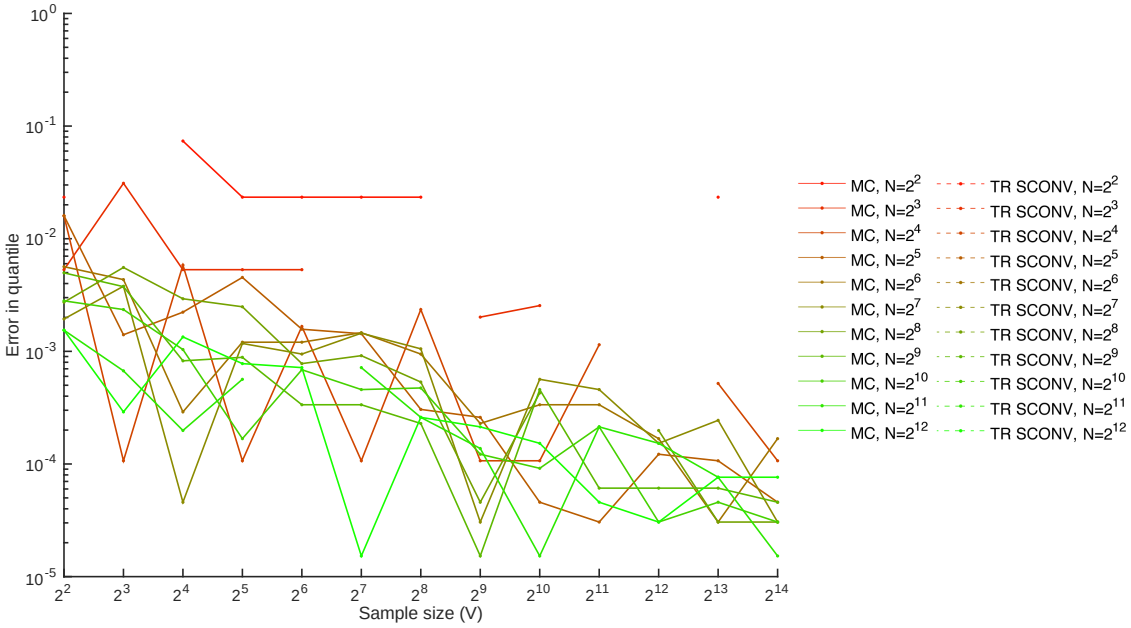


Figure 5.52: Comparison of the errors in VaR computed by the TR SCONV and MC methods
Portfolios in Group 3

(a) $\gamma = 95\%$



(b) $\gamma = 99\%$

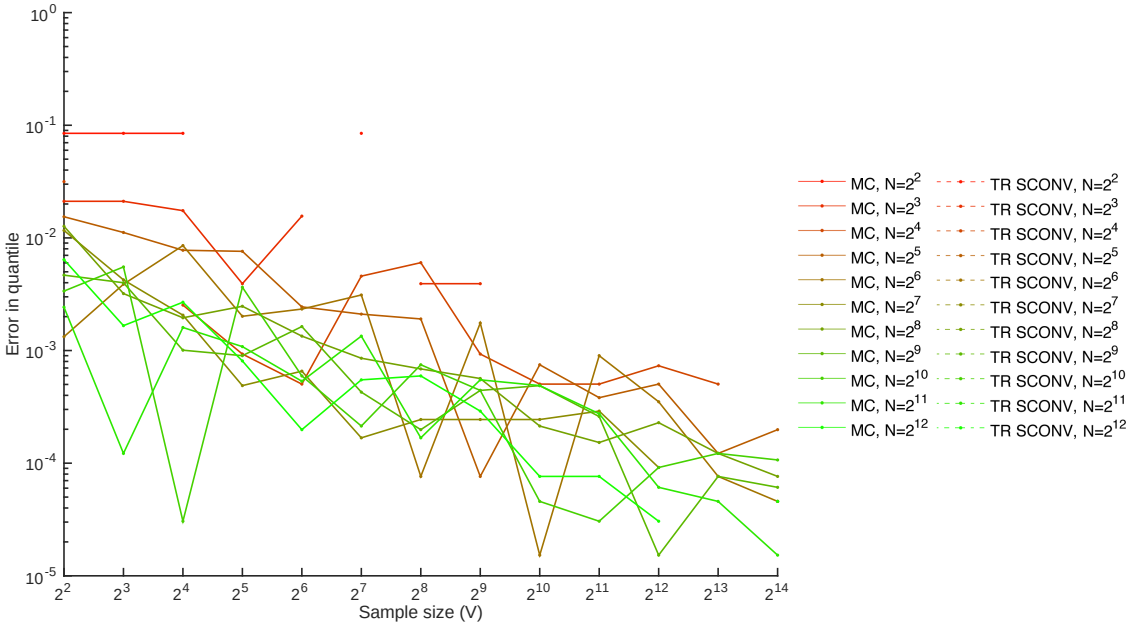
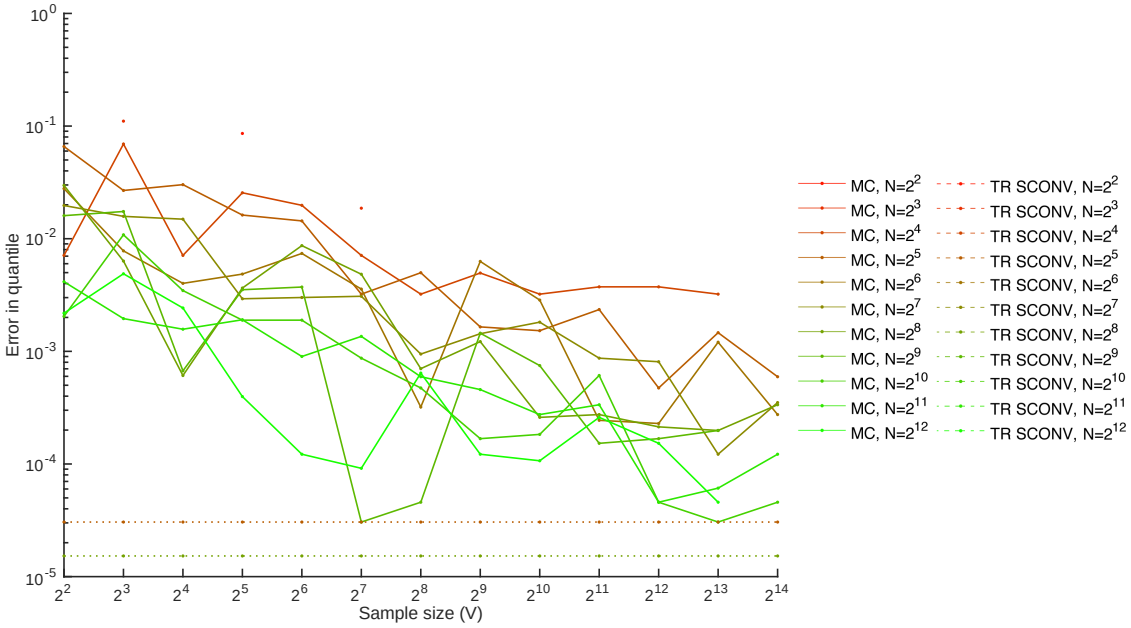


Figure 5.53: Comparison of the errors in VaR computed by the TR SCONV and MC methods *Portfolios in Group 3* (Cont'd)

(a) $\gamma = 99.9\%$



(b) $\gamma = 99.98\%$

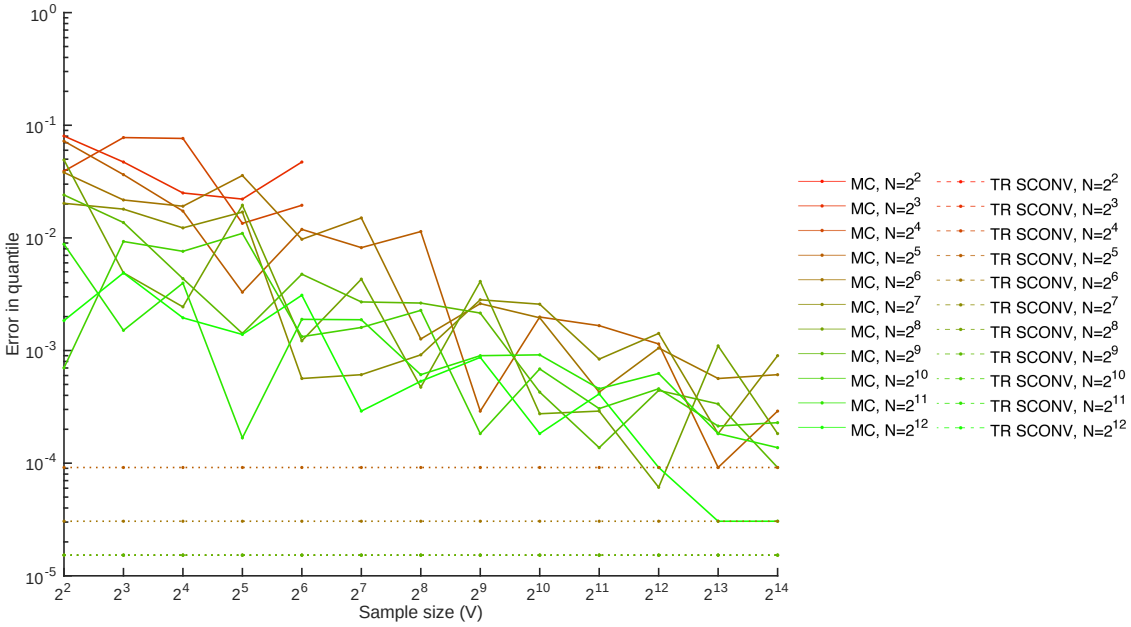
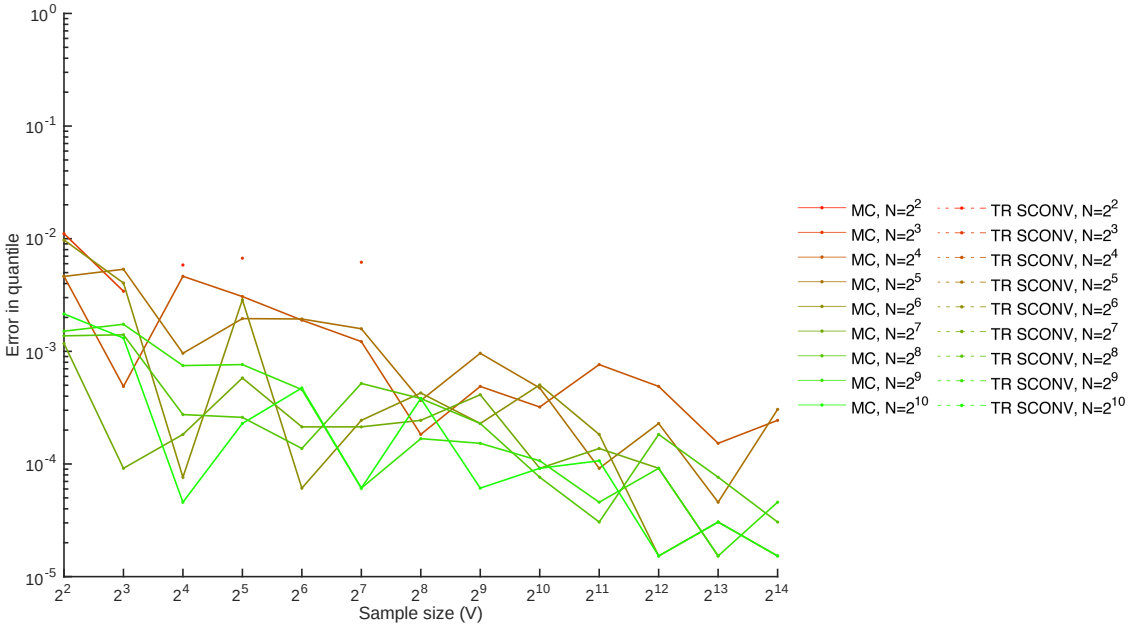


Figure 5.54: Comparison of the errors in VaR computed by the TR SCONV and MC methods
Portfolios in Group 4

(a) $\gamma = 95\%$



(b) $\gamma = 99\%$

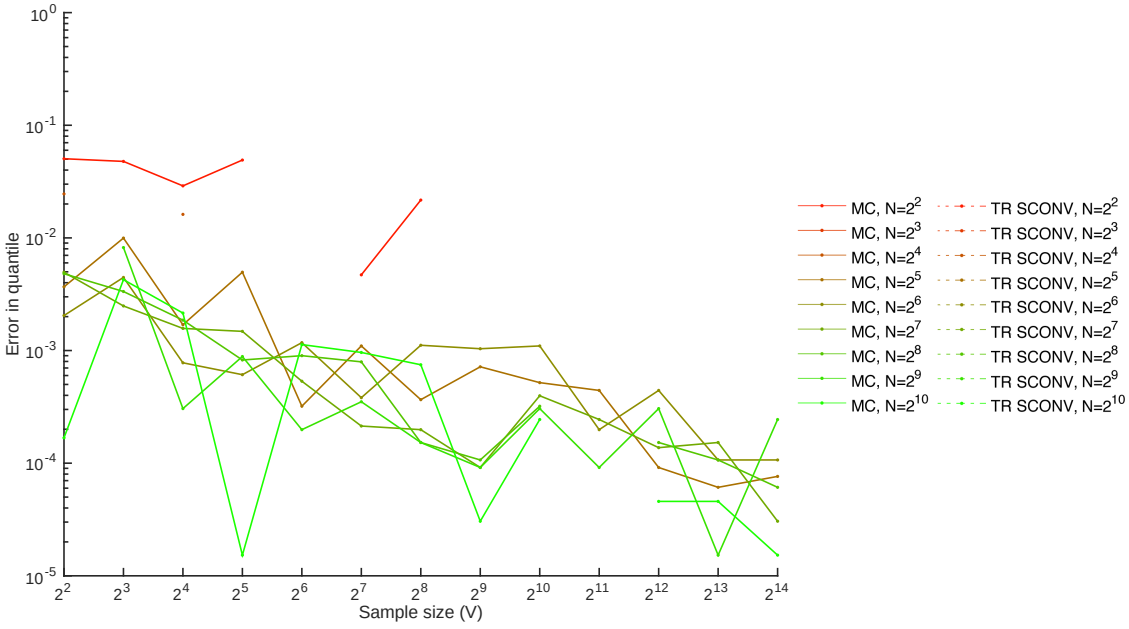
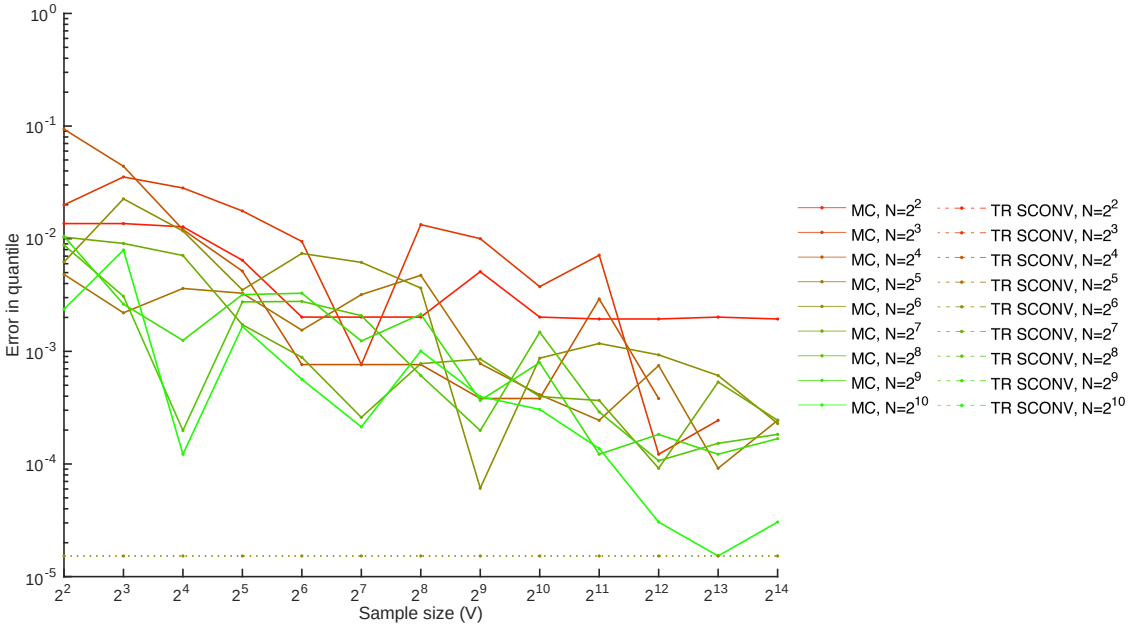
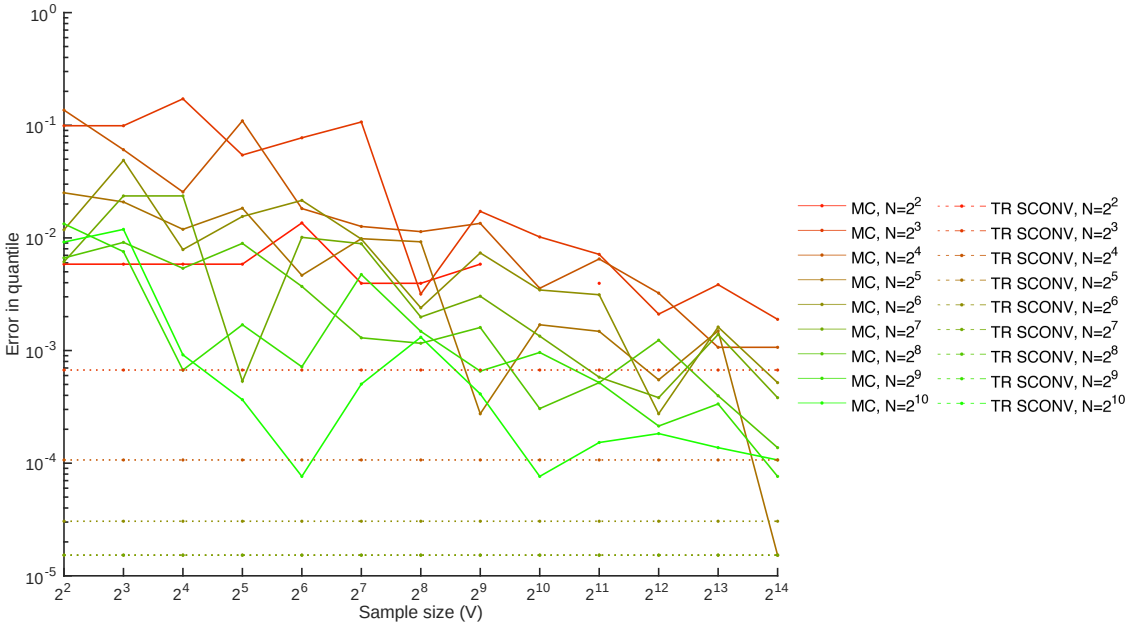


Figure 5.55: Comparison of the errors in VaR computed by the TR SCONV and MC methods *Portfolios in Group 4* (Cont'd)

(a) $\gamma = 99.9\%$



(b) $\gamma = 99.98\%$

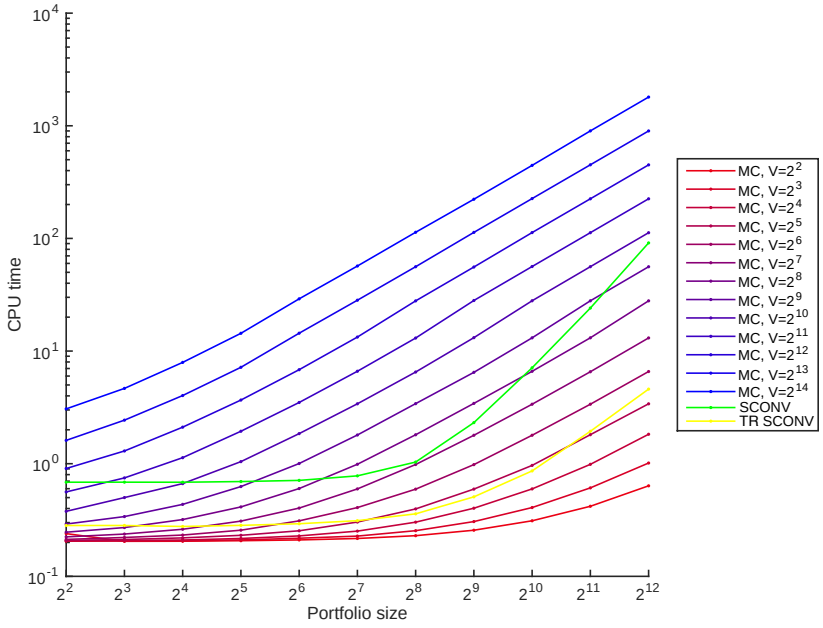


5.3.4.2 Efficiency

To compare the efficiency, we record the CPU time in seconds required by the MC, SCONV and TR SCONV methods to compute the loss distribution. The results are shown in Figures 5.56 - 5.57.

Figure 5.56: Comparison of the CPU time to compute the loss distribution by SCONV, TR SCONV and MC methods

(a) Portfolios in Group 1



(b) Portfolios in Group 2

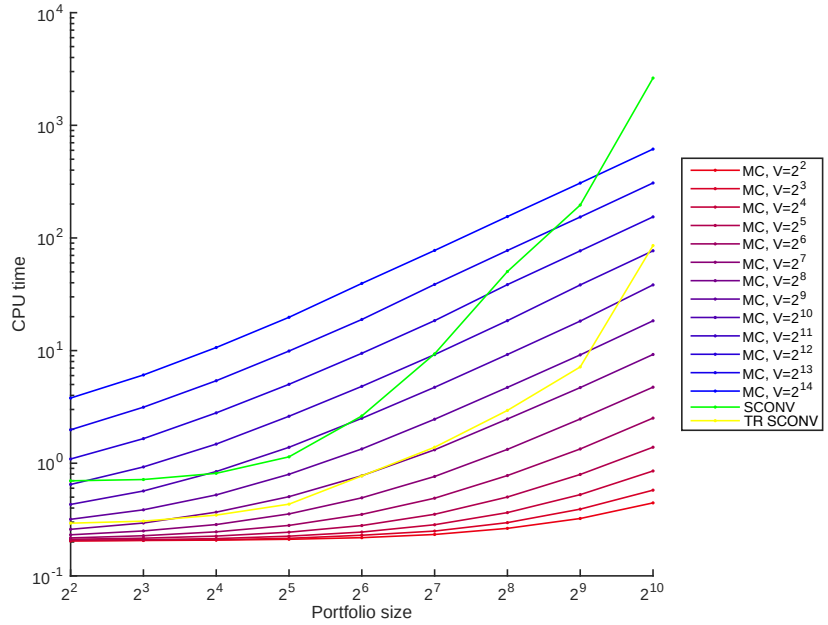
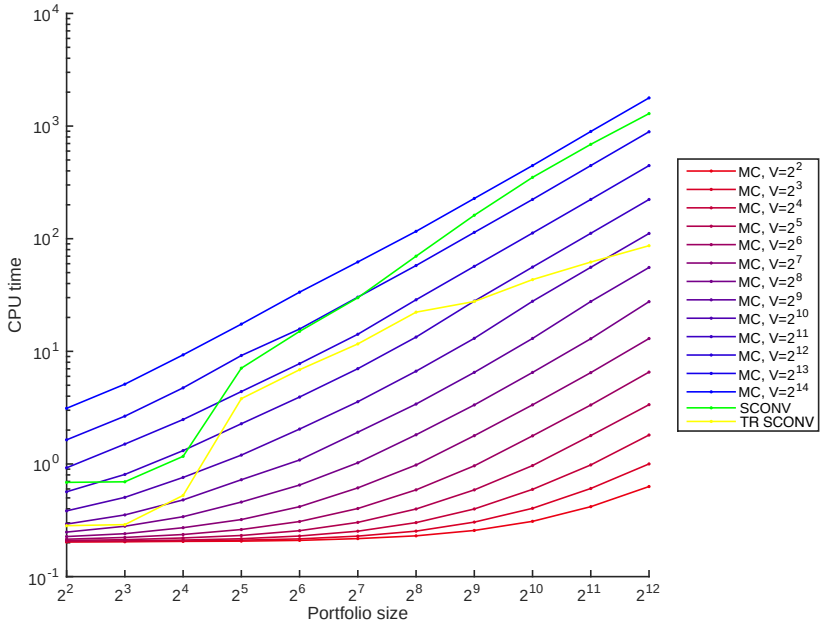
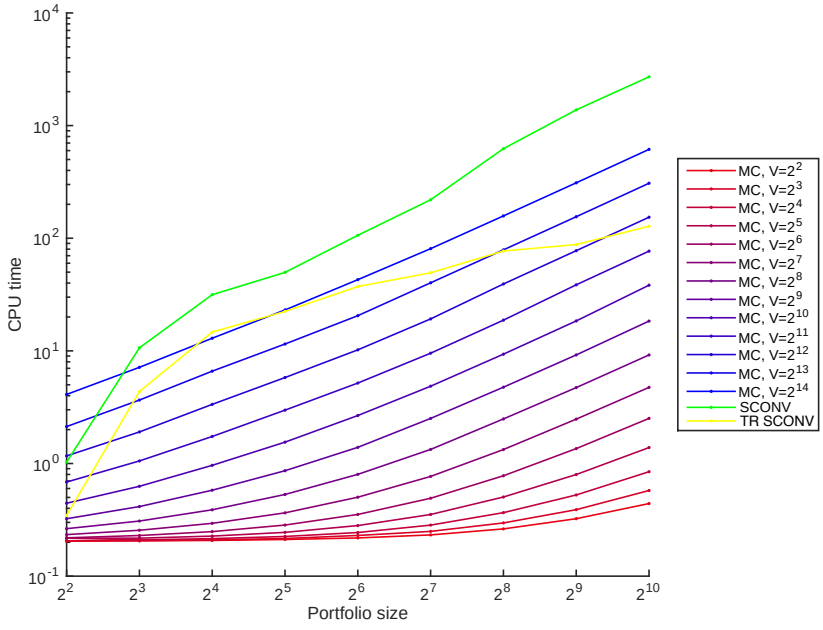


Figure 5.57: Comparison of the CPU time to compute the loss distribution by SCONV, TR SCONV and MC methods (Cont'd)

(a) Portfolios in Group 3



(b) Portfolios in Group 4



We make the following observations based on the testing results shown in Figures 5.56 - 5.57.

- For portfolios in Group 1, the CPU time of TR SCONV method is comparable to that of the MC method with the sample size of 2^5, and for portfolios in Group 2, the CPU time of the TR

SCONV method is comparable to that of the MC method with the sample size of 2^7, except for the portfolio with 2^{10} obligors where the CPU time of the TR SCONV method jumps to a level comparable to that of the MC method with the sample size of 2^{11}. In contrast to the CPU time of the TR SCONV method for homogeneous portfolios, the CPU time of the TR SCONV method for inhomogeneous portfolios increases with the the portfolio size at a faster speed when the portfolio size is small, but at a slower speed when portfolio size is large. Even though the TR SCONV method is slower for inhomogeneous portfolios than homogeneous ones, it is still faster than the MC method with the largest sample size of 2^{14} in almost all cases, with the only one exception being for the portfolio with 2^4 obligors in Group 4. For example, for the portfolio with 2^{12} obligors in Group 3, the CPU time of the TR SCONV method is between that of the MC method with the sample size of 2^9 and 2^{10}, and for the portfolio with 2^{10} obligors in Group 4, the CPU time of the TR SCONV method is between that of the MC method with the sample size of 2^{11} and 2^{12}. Therefore, together with our comparison on the accuracy in the last subsubsection, it is clear that TR SCONV method is more efficient and more accurate than the MC method with a reasonably large sample size.

- The CPU time of the MC method is similar for portfolios in different groups, which shows that the MC method is not sensitive to the sparsity and heterogeneity of the portfolio. However, testing results in Figures 5.56 - 5.57. re-confirm the conclusion reached in the last subsubsection on the impact of the sparsity and heterogeneity of the portfolio on the efficiency of the SCONV and TR SCONV methods: they are sensitive to the sparsity and heterogeneity of the portfolio, and as the portfolio becomes less sparse and less homogeneous, the SCONV and TR SCONV methods run slower.

- Regardless of size, sparsity and heterogeneity of the portfolio, the TR SCONV method is consistently faster than the SCONV method, which shows that dropping terms involving extremely small probabilities is very effective in reducing the computing time of the SCONV method.

Chapter 6

Hybrid Methods

One particular type of inhomogeneous, coarse-grained portfolio that often arises in practice is what we call a "lumpy" portfolio, which consists of a large, fairly homogeneous, fine-grained sub-portfolio and a small inhomogeneous sub-portfolio with large exposures to a few obligors. For lumpy portfolios, the methods discussed in the previous two chapters have different characteristics of efficiency and accuracy. Exact methods are very accurate, but relatively slow, even though the efficiency can be improved substantially by exploiting the sparsity. Asymptotic approximations are very fast, but their loss probabilities are close to the true loss probabilities only when the portfolio is fairly fine-grained and close to being homogeneous. Hence the accuracy of asymptotic approximations is problematic for lumpy portfolios. In this section, we develop a hybrid approximation for such lumpy portfolios. This hybrid approximation combines an asymptotic approximation (CLT or LLN) and an exact method or MC approximation. Our goal for the hybrid method is to be more accurate than either the LLN or CLT approximation for lumpy portfolios, but much less computationally expensive than an exact method or MC simulation.

For a given lumpy portfolio Π, we first divide it into two sub-portfolios Π_{homo} and Π_{lmp}. The sub-portfolio Π_{homo} is fine-grained and close to homogeneous, while the sub-portfolio Π_{lmp} consists of the remaining few obligors which are highly heterogeneous and coarse-grained. We relabel the obligors in Π so that obligors 0, 1, ..., $N_{\mathsf{homo}} - 1$ are in Π_{homo}, and obligors N_{homo}, $N_{\mathsf{homo}} + 1$, ..., $N - 1$ are in Π_{lmp}, and denote the number of obligors in Π_{lmp} by $N_{\mathsf{lmp}} = N - N_{\mathsf{homo}}$. Therefore, conditional on the systematic risk factors $\boldsymbol{z} = \boldsymbol{Z}^{(u)}$, the loss of the portfolio Π can be written as

$$L^{(N)}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}) = L_{\mathsf{homo}}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}_{\mathsf{homo}}) + L_{\mathsf{lmp}}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}_{\mathsf{lmp}}),$$

where $L_{\mathsf{homo}}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}_{\mathsf{homo}})$ and $L_{\mathsf{lmp}}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}_{\mathsf{lmp}})$ are the conditional losses of the sub-portfolios Π_{homo} and Π_{lmp},

respectively. Consequently, given loss level l_k on a discrete grid, the conditional loss probability can be computed as

$$\begin{aligned}\mathbb{P}\left\{L^{(N)}(\boldsymbol{z},\boldsymbol{\mathcal{E}})<l_k\right\} &= \mathbb{P}\left\{L_{\mathsf{homo}}(\boldsymbol{z},\boldsymbol{\mathcal{E}})+L_{\mathsf{lmp}}(\boldsymbol{z},\boldsymbol{\mathcal{E}})\le l_k\right\}\\ &= \mathbb{E}\left[\mathbb{P}\left\{L_{\mathsf{homo}}(\boldsymbol{z},\boldsymbol{\mathcal{E}})+L_{\mathsf{lmp}}(\boldsymbol{z},\boldsymbol{\mathcal{E}})\le l_k\right| L_{\mathsf{lmp}}(\boldsymbol{z},\boldsymbol{\mathcal{E}})\}\right].\end{aligned} \tag{6.0.1}$$

Since Π_{homo} is fine-grained and close to homogeneous, an asymptotic approximation can be applied to estimate the conditional loss probability very fast with good accuracy. Since Π_{lmp} is small, but highly coarse-grained and heterogeneous, we can apply MC simulation or one of our exact methods to calculate its conditional loss probability. Notice that conditional on $\boldsymbol{\mathcal{Z}}=\boldsymbol{z}$, $L_{\mathsf{homo}}(\boldsymbol{z},\boldsymbol{\mathcal{E}}_{\mathsf{homo}})$ and $L_{\mathsf{lmp}}(\boldsymbol{z},\boldsymbol{\mathcal{E}}_{\mathsf{lmp}})$ are independent, since elements in $\boldsymbol{\mathcal{E}}_{\mathsf{homo}}$ and elements in $\boldsymbol{\mathcal{E}}_{\mathsf{lmp}}$ are mutually independent. Therefore, we can easily obtain the conditional loss probability for the whole portfolio by computing the conditional loss probability for each sub-portfolio. Depending on the method applied to the lumpy sub-portfolio, two types of hybrid method can be developed.

6.1 Hybrid Method: MC+CLT/LLN

To compute (6.0.1), we can apply MC simulation to generate i.i.d. samples of $L_{\mathsf{lmp}}(\boldsymbol{z},\boldsymbol{\mathcal{E}})$, then calculate the loss probability conditional on each of these samples, and use the sample mean to approximate the expectation. To this end, we first sample from the multivariate standard normal distribution $\mathcal{N}\left(\mathbf{0},\mathbf{I}_{N_{\mathsf{lmp}}\times N_{\mathsf{lmp}}}\right)$ to obtain samples of the individual risk factors $\boldsymbol{\mathcal{E}}_{\mathsf{lmp}}^{(v)}$, $v=1,\ldots,V$, and then generate V loss scenarios $l^{(1)}(\boldsymbol{z})$, $l^{(2)}(\boldsymbol{z})$, …, $l^{(V)}(\boldsymbol{z})$ by

$$l_{\mathsf{lmp}}^{(v)}(\boldsymbol{z})=\sum_{n=N_{\mathsf{homo}}}^{N-1}\omega_n\left(\sum_{c=0}^{C-1}\mathrm{LGC}_n^c\mathbb{I}_{\left\{\frac{H_{c(n)}^{c-1}-\boldsymbol{\beta}_n^T\boldsymbol{z}}{\sqrt{1-\boldsymbol{\beta}_n^T\boldsymbol{\beta}_n}}\le\mathcal{E}_n^{(v)}<\frac{H_{c(n)}^{c}-\boldsymbol{\beta}_n^T\boldsymbol{z}}{\sqrt{1-\boldsymbol{\beta}_n^T\boldsymbol{\beta}_n}}\right\}}\right). \tag{6.1.1}$$

From (6.0.1) and (6.1.1), we now have

$$\begin{aligned}\mathbb{P}\left\{L^{(N)}(\boldsymbol{z},\boldsymbol{\mathcal{E}})\le l\right\} &\approx \frac{1}{V}\sum_{v=1}^{V}\mathbb{P}\left\{L_{\mathsf{homo}}(\boldsymbol{z},\boldsymbol{\mathcal{E}}_{\mathsf{homo}})+L_{\mathsf{lmp}}(\boldsymbol{z},\boldsymbol{\mathcal{E}}_{\mathsf{lmp}})\le l_k\right| L_{\mathsf{lmp}}(\boldsymbol{z},\boldsymbol{\mathcal{E}}_{\mathsf{lmp}})=l_{\mathsf{lmp}}^{(v)}(\boldsymbol{z})\Big\}\\ &= \frac{1}{V}\sum_{v=1}^{V}\mathbb{P}\left\{L_{\mathsf{homo}}(\boldsymbol{z},\boldsymbol{\mathcal{E}}_{\mathsf{homo}})\le l_k-l_{\mathsf{lmp}}^{(v)}(\boldsymbol{z})\right\}.\end{aligned} \tag{6.1.2}$$

Notice that, for the second equation, we used the fact that conditional on $\boldsymbol{\mathcal{Z}} = \boldsymbol{z}$, $L_{\mathsf{homo}}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}_{\mathsf{homo}})$ and $L_{\mathsf{lmp}}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}_{\mathsf{lmp}})$ are independent.

On the other hand, since Π_{homo} is fine-grained and close to homogeneous, an asymptotic approximation can be applied to estimate the portfolio loss. If we use the CLT approximation, then

$$L_{\mathsf{homo}}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}_{\mathsf{homo}}) \approx \sigma_{\mathsf{homo}}(\boldsymbol{z})\mathcal{X} + \mu_{\mathsf{homo}}(\boldsymbol{z}), \tag{6.1.3}$$

where $\mathcal{X} \sim N(0,1)$, $\sigma^2_{\mathsf{homo}}(\boldsymbol{z}) = \mathbb{V}\left[L_{\mathsf{homo}}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}_{\mathsf{homo}})\right]$ and $\mu_{\mathsf{homo}}(\boldsymbol{z}) = \mathbb{E}\left[L_{\mathsf{homo}}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}_{\mathsf{homo}})\right]$. Substituting (6.1.3) into (6.1.2), we have

$$\begin{aligned}
\mathbb{P}\left\{L^{(N)}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}) \le l\right\} &\approx \frac{1}{V}\sum_{v=1}^{V} \mathbb{P}\left\{\sigma_{\mathsf{homo}}(\boldsymbol{z})\mathcal{X} + \mu_{\mathsf{homo}}(\boldsymbol{z}) \le l_k - l^{(v)}_{\mathsf{lmp}}(\boldsymbol{z})\right\} \\
&= \frac{1}{V}\sum_{v=1}^{V} \Phi\left(\frac{l_k - l^{(v)}(\boldsymbol{z}) - \mu_{\mathsf{homo}}(\boldsymbol{z})}{\sigma_{\mathsf{homo}}(\boldsymbol{z})}\right).
\end{aligned} \tag{6.1.4}$$

If the LLN approximation is applied instead of the CLT approximation, then $L_{\mathsf{homo}}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}_{\mathsf{homo}}) \approx \mu_{\mathsf{homo}}(\boldsymbol{z})$, and

$$\mathbb{P}\left\{L^{(N)}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}) \le l_k\right\} \approx \frac{1}{V}\sum_{v=1}^{V} \mathbb{I}_{\left\{\mu_{\mathsf{homo}}(\boldsymbol{z}) \le l_k - l^{(v)}_{\mathsf{lmp}}(\boldsymbol{z})\right\}}. \tag{6.1.5}$$

To summarize, on the discretized gird, the portfolio loss probability is calculated by the MC+CLT/LLN method as follows

$$\mathbb{P}\left\{\mathcal{L} \le l_k\right\} = \begin{cases} \frac{1}{UV}\sum\limits_{u=1}^{U}\sum\limits_{v=1}^{V} \Phi\left(\frac{l_k - l_{m^{(u,v)}} - \mu^{(u)}_{\mathsf{homo}}}{\sigma^{(u)}_{\mathsf{homo}}}\right), & \text{MC+CLT}, \\ \frac{1}{UV}\sum\limits_{u=1}^{U}\sum\limits_{v=1}^{V} \mathbb{I}_{\left\{\mu^{(u)}_{\mathsf{homo}} \le l_k - l_{m^{(u,v)}}\right\}}, & \text{MC+LLN}, \end{cases} \tag{6.1.6}$$

where $m^{(u,v)} = l^{(u,v)}_{\mathsf{lmp}}/\delta$, $\mu^{(u)}_{\mathsf{homo}} \triangleq \mu_{\mathsf{homo}}\left(\boldsymbol{\mathcal{Z}}^{(\boldsymbol{u})}\right)$, $\sigma^{(u)}_{\mathsf{homo}} \triangleq \sigma_{\mathsf{homo}}\left(\boldsymbol{\mathcal{Z}}^{(\boldsymbol{u})}\right)$, and $\boldsymbol{\mathcal{Z}}^{(\boldsymbol{u})}$ is the uth realization of the systematic risk factor in the MC simulation.

6.2 Hybrid Method: EXACT+CLT/LLN

We can also apply one of the exact methods to calculate the conditional probability mass function of the loss of the lumpy sub-portfolio Π_{lmp}:

$$\mathbb{P}\left\{L_{\mathsf{lmp}}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}_{\mathsf{lmp}}) = l_m\right\} \triangleq p^m_{\mathsf{lmp}}(\boldsymbol{z}), \tag{6.2.1}$$

for $m \in \tilde{\boldsymbol{d}}_{\mathsf{lmp}}$, where $\tilde{\boldsymbol{d}}_{\mathsf{lmp}} \triangleq \left\{m : \ p^m_{\mathsf{lmp}}(\boldsymbol{z}) \neq 0\right\}$ is the set which includes indices of nonzero elements in the loss probability vector as shown in Algorithm 5.3. Therefore, from (6.0.1) and (6.2.1) we have

$$\begin{aligned}
\mathbb{P}\left\{L^{(N)}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}) \leq l_k\right\} &= \sum_{m \in \tilde{\boldsymbol{d}}_{\mathsf{lmp}}} \mathbb{P}\left\{ L_{\mathsf{homo}}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}_{\mathsf{homo}}) + L_{\mathsf{lmp}}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}_{\mathsf{lmp}}) \leq l_k | \, L_{\mathsf{lmp}}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}_{\mathsf{lmp}}) = l_m \right\} p^m_{\mathsf{lmp}}(\boldsymbol{z}) \\
&= \sum_{m \in \tilde{\boldsymbol{d}}_{\mathsf{lmp}}} \mathbb{P}\left\{L_{\mathsf{homo}}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}_{\mathsf{homo}}) \leq l_k - l_m\right\} p^m_{\mathsf{lmp}}(\boldsymbol{z}),
\end{aligned}$$

Notice that, as in MC+CLT/LLN method, we used the fact that conditional on $\boldsymbol{\mathcal{Z}} = \boldsymbol{z}$, $L_{\mathsf{homo}}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}_{\mathsf{homo}})$ and $L_{\mathsf{lmp}}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}_{\mathsf{lmp}})$ are independent to obtain the second equation. Applying CLT/LLN to the homogeneous sub-portfolio, we have

$$\mathbb{P}\left\{\mathcal{L} \leq l_k\right\} = \begin{cases} \frac{1}{U} \sum_{u=1}^{U} \sum_{m \in \tilde{\boldsymbol{d}}_{\mathsf{lmp}}} \Phi\left(\frac{l_k - l_m - \mu^{(u)}_{\mathsf{homo}}}{\sigma^{(u)}_{\mathsf{homo}}}\right) p^{(m,u)}_{\mathsf{lmp}}, & \text{EXACT+CLT}, \\ \frac{1}{U} \sum_{u=1}^{U} \sum_{m \in \tilde{\boldsymbol{d}}_{\mathsf{lmp}}} \mathbb{I}_{\left\{\mu^{(u)}_{\mathsf{homo}} \leq l_k - l_m\right\}} p^{(m,u)}_{\mathsf{lmp}}, & \text{EXACT+LLN}, \end{cases} \tag{6.2.2}$$

where $p^{(m,u)}_{\mathsf{lmp}} \triangleq p^m_{\mathsf{lmp}}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)$ and $\boldsymbol{\mathcal{Z}}^{(\boldsymbol{u})}$ is the uth realization of the systematic risk factor in the MC simulation.

Among the exact methods developed in the previous chapter, we believe the best choice of the exact method to apply to the lumpy sub-portfolio is the truncated sparse convolution method. Compared with the sparse convolution method, it provides significant improvement in efficiency at a cost of a user controlled loss of accuracy. Compared with the truncated sparse FFT method with exponential windowing, the truncated sparse convolution method only introduces truncation errors caused by dropping products of extremely small probabilities, while the truncated sparse FFT method with exponential windowing suffers from the two types of errors: (1) conditional probabilities, $p^m_{\mathsf{lmp}}(\boldsymbol{z})$, $m > \bar{K}$, are ignored, where $\bar{K}$ is the length of the truncated version of the conditional loss probability vector of the lumpy sub-portfolio; (2) aliasing errors are introduced in the computation of $p^m_{\mathsf{lmp}}(\boldsymbol{z})$, $m \leq \bar{K}$. An additional complication associated with using the truncated sparse FFT method is that, it is not easy to determine $\bar{K}$[1] and the

[1]This is not a problem if we apply the (truncated) sparse FFT method with exponential windowing to calculate VaR of

optimal τ chosen to compute the truncated conditional loss probability vector.[2]

6.3 Implementation of the Hybrid Methods

In this section, we discuss how to implement the hybrid methods to calculate the portfolio loss distribution and VaR based on (6.1.6) and (6.2.2).

6.3.1 Implementation of the Hybrid Methods to Compute the Portfolio Loss Distribution

6.3.1.1 Implementation Details

Several factors should be considered carefully when implementing the hybrid methods to compute the portfolio loss distribution efficiently.

First of all, it is not efficient to directly implement (6.1.6) for the MC+LLN method or (6.2.2) for the EXACT+LLN method. Given $l_k - l_{\mathsf{lmp}}^{(u,v)}$ and $\mu_{\mathsf{homo}}^{(u)}$, a direct implementation of (6.1.6) for the MC+LLN method costs UVK evaluations of the indicator function and the summation of UVK terms. Similarly, (6.2.2) for the EXACT+LLN method requires $UH_{N_{\mathsf{lmp}}-1}K$ evaluations of the indicator function and the summation of $UH_{N_{\mathsf{lmp}}-1}K$ terms, where $H_{N_{\mathsf{lmp}}-1} = \left|\tilde{\boldsymbol{d}}_{\mathsf{lmp}}\right|$ is the the number of nonzero values in the vector of loss probabilities of the lumpy sub-portfolio as defined in Theorem 5.1. Alternatively, two better options are offered for the MC+LLN method:

- Sort $\boldsymbol{a} = \left\{\mu_{\mathsf{homo}}^{(u)} + l_{m^{(u,v)}},\ u = 1,\dots,U,\ v = 1,\dots,V\right\}$ in ascending order, denoted by $\hat{\boldsymbol{a}} = \{\hat{a}_m :\ \hat{a}_1 \le \hat{a}_2 \le \cdots \le \hat{a}_{UV}\}$, then, for each $k = 0,\dots,K-1$, compute $\mathbb{P}\{\mathcal{L} \le l_k\}$ by

$$\mathbb{P}\{\mathcal{L} \le l_k\} = \frac{m_k}{UV},$$

  where $m_k = \max\{m :\ \hat{a}_m \le l_k,\ m = 1,\dots,UV\}$. A good sorting algorithm, such as quick sort and merge sort, can sort $a^{(u,v)}$ in $O(UV \log UV)$ comparisons, and a bisection search for m_k for

the whole portfolio, in which case $\bar{K}$ can be estimated by a pre-run of the CLT approximation or LLN approximation. By a pre-run of CLT method, one can obtain VaR, denoted by VaR_{clt}, obtained by the CLT method, then $\bar{K}$ can be estimated by

$$\bar{K} = \min\left\{2^t\ 2^t \ge \mathrm{VaR}_{clt}\right\}.$$

[2]The proposed method to determine the optimal τ discussed in the previous section does not apply in this case since it is designed to compute the optimal τ for the cumulative unconditional loss probability function valued at a particular point, $\mathbb{P}\{\mathcal{L} \le l_k\}$, while in the hybrid method, the exact method is used to calculate the conditional probability mass function of the lumpy sub-portfolio.

all $k = 0, \ldots, K-1$ requires $O(K \log_2 UV)$ comparisons. The total number of comparisons needed by this option is $O(UV \log UV)$ given $UV \gg K$.

- For each $u = 1, \ldots, U$, sort $\boldsymbol{a}^{(u)} = \left\{\mu_{\mathsf{homo}}^{(u)} + l_{m^{(u,v)}},\ v = 1, \ldots, V\right\}$ in ascending order, denoted by $\hat{\boldsymbol{a}}^{(u)} = \left\{\hat{a}_m^{(u)} :\ \hat{a}_1^{(u)} \le \hat{a}_2^{(u)} \le \cdots \le \hat{a}_V^{(u)}\right\}$, then, for each $k = 0, \ldots, K-1$, compute $\mathbb{P}\{\mathcal{L} \le l_k\}$ by

$$\mathbb{P}\{\mathcal{L} \le l_k\} = \frac{1}{UV} \sum_{u=1}^{U} m_k^{(u)},$$

  where $m_k^{(u)} = \max\left\{m :\ \hat{a}_m^{(u)} \le l_k,\ m = 1, \ldots, V\right\}$. Sorting $a^{(u,v)}$ requires $O(UV \log V)$ comparisons for all $u = 1, \ldots, U$, a bisection search for $m_k^{(u)}$ for all $u = 1, \ldots, U$ and $k = 0, \ldots, K-1$ requires $O(UK \log_2 V)$ comparisons and the summations require $O(UK)$ FLOPs . The total number of operations (i.e. comparisons or FLOPs) needed by this option is $O(UK \log_2 V)$ given $K \gg V$.

Both options are less computationally intensive than the direct implementation of (6.1.6) for the MC+LLN method, which requires $O(UVK)$ FLOPs. Comparing the two options, we believe the second option is often likely to be better even though its total number of FLOPs is larger than that required by the first option. The reason for this is, the first option needs to store and access a huge vector, $\boldsymbol{a}$, of UV elements while the second option only needs to store and access a vector, $\boldsymbol{a}^{(u)}$, of V elements. If the sample sizes U and V are very large, the system might not have large enough memory to store UV elements, and, even if the system is able to hold UV elements, memory access to a very long array could be very slow, which could offset the gain in CPU time from less operations. For the EXACT+LLN method, a similar strategy can be applied. For example, for the second option, instead of sorting $\boldsymbol{a}^{(u)}$, we need to sort the matrix

$$\begin{aligned}
\mathbf{A}^{(u)} &= \left[\boldsymbol{a}^{(u)}, \boldsymbol{p}_{\mathsf{lmp}}^{(u)}\right], \\
\boldsymbol{a}^{(u)} &= \left[\mu_{\mathsf{homo}}^{(u)} + l_{\tilde{d}_{\mathsf{lmp}}[1]}, \ldots, \mu_{\mathsf{homo}}^{(u)} + l_{\tilde{d}_{\mathsf{lmp}}\left[H_{N_{\mathsf{lmp}}-1}\right]}\right]^T, \\
\boldsymbol{p}_{\mathsf{lmp}}^{(u)} &= \left[p_{\mathsf{lmp}}^{\left(\tilde{d}_{\mathsf{lmp}}[1],u\right)}, \ldots, p_{\mathsf{lmp}}^{\left(\tilde{d}_{\mathsf{lmp}}[H_{N_{\mathsf{lmp}}-1}],u\right)}\right]^T,
\end{aligned}$$

by the first column in ascending order, denoted by

$$\hat{\mathbf{A}}^{(u)} = \left[\hat{\boldsymbol{a}}^{(u)}, \hat{\boldsymbol{p}}_{\mathsf{lmp}}^{(u)}\right],$$

then for each $k = 0, \ldots, K-1$, compute $\mathbb{P}\{\mathcal{L} \le l_k\}$ by

$$\mathbb{P}\{\mathcal{L} \le l_k\} = \frac{1}{U} \sum_{u=1}^{U} \sum_{m=1}^{m_k^{(u)}} \hat{p}_{\mathsf{lmp}}^{(m,u)}, \tag{6.3.1}$$

where $m_k^{(u)} = \max\left\{m : \hat{a}_m^{(u)} \le l_k,\ m = 1, \ldots, H_{N_{\mathsf{lmp}}-1}\right\}$. Sorting $\mathbf{A}^{(u)}$ by its first column requires $O(UH_{N_{\mathsf{lmp}}-1} \log H_{N_{\mathsf{lmp}}-1})$ comparisons for all $u = 1, \ldots, U$, a bisection search for $m_k^{(u)}$ for all $u = 1, \ldots, U$ and $k = 0, \ldots, K-1$ requires $O(UK \log_2 H_{N_{\mathsf{lmp}}-1})$ comparisons. Notice that, unlike the MC+LLN method, for the EXACT+LLN method, summations in (6.3.1) take $O(UKH_{N_{\mathsf{lmp}}-1}/2)$ FLOPs on average. Therefore, the total number of FLOPs needed by this option is still $O(UKH_{N_{\mathsf{lmp}}-1})$. Even though (6.3.1) requires less FLOPs to compute summations than (6.2.2), the total saving on the computational effort by (6.3.1) over (6.2.2) for the EXACT+LLN method is not as large as that for MC+LLN method.

Secondly, for the EXACT+CLT method, for each $u = 1, \ldots, U$, a naive implementation of (6.2.2) could compute $\Phi\left(\frac{l_k - l_m - \mu_{\mathsf{homo}}^{(u)}}{\sigma_{\mathsf{homo}}^{(u)}}\right)$ for every $k = 1, \ldots, K$ and $m \in \tilde{\boldsymbol{d}}_{\mathsf{lmp}}$. This wastes computational time on computing duplicate values of $\Phi\left(\frac{l_k - l_m - \mu_{\mathsf{homo}}^{(u)}}{\sigma_{\mathsf{homo}}^{(u)}}\right)$. For example, the value of $\Phi\left(\frac{l_k - l_m - \mu_{\mathsf{homo}}^{(u)}}{\sigma_{\mathsf{homo}}^{(u)}}\right)$ based on the combination of $k = 3$ and $m = 1$ is exactly same as for $k = 4$ and $m = 2$, or $k = 5$ and $m = 3$, etc., since $l_k - l_m = (k-m)\delta$. This could result in a severe loss in efficiency since the computation of the normal CDF function Φ is costly. Instead, for each $u = 1, \ldots, U$, one can first compute and save

$$\Phi_i^{(u)} = \Phi\left(\frac{l_i - \mu_{\mathsf{homo}}^{(u)}}{\sigma_{\mathsf{homo}}^{(u)}}\right),\ i = -\max\left(\tilde{\boldsymbol{d}}_{\mathsf{lmp}}\right), \ldots, K-1-\min\left(\tilde{\boldsymbol{d}}_{\mathsf{lmp}}\right), \tag{6.3.2}$$

and then compute

$$\mathbb{P}\{\mathcal{L} \le l_k\} = \frac{1}{U} \sum_{u=1}^{U} \sum_{m \in \tilde{\boldsymbol{d}}_{\mathsf{lmp}}} \Phi_{k-m}^{(u)} p_{\mathsf{lmp}}^{(m,u)}. \tag{6.3.3}$$

Compared with the naive implementation of the EXACT+CLT method based on (6.2.2), the implementation based on (6.3.2) and (6.3.3) can reduce the number of evaluations of the normal CDF function significantly from $KH_{N_{\mathsf{lmp}}-1}$ to $K - \min\left(\tilde{\boldsymbol{d}}_{\mathsf{lmp}}\right) + \max\left(\tilde{\boldsymbol{d}}_{\mathsf{lmp}}\right)$. This trick can be applied to the MC+CLT method as well.

Based on discussion above, we propose Algorithms 6.1 - 6.4 to compute the portfolio loss distribution by the MC+LLN, EXACT+LLN, MC+CLT and EXACT+CLT methods, respectively.

Algorithm 6.1 MC+LLN method to compute $\mathbb{P}\{\mathcal{L} \le l_k\}$, $k = 0, \dots, K-1$

Input: Subportfolio Π_{homo}, Π_{Imp} and loss levels $\boldsymbol{l}$;

Output: $\boldsymbol{p}$;

$\boldsymbol{p} \leftarrow \boldsymbol{0}$;
for $u = 1 : U$ **do**
  Generate $\boldsymbol{\mathcal{Z}}^{(u)}$, $\boldsymbol{z} \leftarrow \boldsymbol{\mathcal{Z}}^{(u)}$; # $O(US)$ FLOPs
  Compute $\mu_{\mathsf{homo}}^{(u)}$ based on $\boldsymbol{z}$, $\mu_{\mathsf{homo}} \leftarrow \mu_{\mathsf{homo}}^{(u)}$; # $O(UN_{\mathsf{homo}})$ FLOPs
  for $v = 1 : V$ **do**
    Generate $\boldsymbol{\mathcal{E}}_{\mathsf{Imp}}^{(u,v)}$, $\boldsymbol{\epsilon}_{\mathsf{Imp}} \leftarrow \boldsymbol{\mathcal{E}}_{\mathsf{Imp}}^{(u,v)}$; # $O(UVN_{\mathsf{Imp}})$ FLOPs
    Compute $l_{m^{(u,v)}}$ based on $\boldsymbol{z}$ and $\boldsymbol{\epsilon}_{\mathsf{Imp}}$, $l_{\mathsf{Imp}}[v] \leftarrow l_{m^{(u,v)}}$; # $O(UVN_{\mathsf{Imp}}C)$ FLOPs
    $a[v] \leftarrow \mu_{\mathsf{homo}} + l_{\mathsf{Imp}}[v]$; # $O(UV)$ FLOPs
  end for
  sort($\boldsymbol{a}$); # $O(UV \log(V)$ FLOPs
  for $k = 0 : K-1$ **do**
    Apply bisection search to find $m_k^{(u)}$ based on $\boldsymbol{a}$ and $l[k]$, $m \leftarrow m_k^{(u)}$; # $O(UK \log_2(V))$ FLOPs
    $p[k] \leftarrow p[k] + m$; # $O(UK)$ FLOPs
  end for
end for
$\boldsymbol{p} \leftarrow \boldsymbol{p}/(UV)$; # $O(K)$ FLOPs
return $\boldsymbol{p}$;

Algorithm 6.2 EXACT+LLN method to compute $\mathbb{P}\{\mathcal{L} \le l_k\}$, $k = 0, \dots, K-1$

Input: Subportfolio Π_{homo}, Π_{Imp} and loss levels $\boldsymbol{l}$;

Output: $\boldsymbol{p}$;

$\boldsymbol{p} \leftarrow \boldsymbol{0}$;
for $u = 1 : U$ **do**
  Generate $\boldsymbol{\mathcal{Z}}^{(u)}$, $\boldsymbol{z} \leftarrow \boldsymbol{\mathcal{Z}}^{(u)}$; # $O(US)$ FLOPs
  Compute $\mu_{\mathsf{homo}}^{(u)}$ based on $\boldsymbol{z}$, $\mu_{\mathsf{homo}} \leftarrow \mu_{\mathsf{homo}}^{(u)}$; # $O(UN_{\mathsf{homo}})$ FLOPs
  Compute $H_{N_{\mathsf{Imp}}-1}$, $\tilde{\boldsymbol{d}}_{\mathsf{Imp}}$, and $\boldsymbol{p}_{\mathsf{Imp}}^{(u)}$ based on $\boldsymbol{z}$ by an exact method, $\boldsymbol{p}_{\mathsf{Imp}} \leftarrow \boldsymbol{p}_{\mathsf{Imp}}^{(u)}$;
    # $O(UC^2N_{\mathsf{Imp}}^2)$ FLOPs (best case, SCONV)
    # $O(UC^{N_{\mathsf{Imp}}})$ FLOPs (worst case, SCONV)
  for $v = 0 : H_{N_{\mathsf{Imp}}-1} - 1$ **do**
    $a[v] \leftarrow \mu_{\mathsf{homo}} + l[\tilde{d}_{\mathsf{Imp}}[v]]$; # $O(UH_{N_{\mathsf{Imp}}-1})$ FLOPs
    for $k = 0 : K-1$ **do**
      $p[k] \leftarrow p[k] + \mathbb{I}_{\{a[v] \le l[k]\}} p_{\mathsf{Imp}}[\tilde{d}_{\mathsf{Imp}}[v]]$; # $O(UKH_{N_{\mathsf{Imp}}-1})$ FLOPs
    end for
  end for
end for
$\boldsymbol{p} \leftarrow \boldsymbol{p}/U$; # $O(K)$ FLOPs
return $\boldsymbol{p}$;

Algorithm 6.3 MC+CLT method to compute $\mathbb{P}\{\mathcal{L} \le l_k\}$, $k = 0, \ldots, K-1$

Input: Subportfolio Π_{homo}, Π_{lmp} and loss levels $\boldsymbol{l}$;

Output: $\boldsymbol{p}$;

$\boldsymbol{p} \leftarrow \mathbf{0}$;
Compute indices of minimum and maximum loss of the lumpy sub-portfolio i_{min} and i_{max}; # $O(N_{\mathsf{lmp}})$ FLOPs
for $u = 1 : U$ **do**
    Generate $\boldsymbol{\mathcal{Z}}^{(u)}$, $\boldsymbol{z} \leftarrow \boldsymbol{\mathcal{Z}}^{(u)}$; # $O(US)$ FLOPs
    Compute $\mu_{\mathsf{homo}}^{(u)}$ and $\sigma_{\mathsf{homo}}^{(u)}$ based on $\boldsymbol{z}$, $\mu_{\mathsf{homo}} \leftarrow \mu_{\mathsf{homo}}^{(u)}$, $\sigma_{\mathsf{homo}} \leftarrow \sigma_{\mathsf{homo}}^{(u)}$; # $O(UN_{\mathsf{homo}})$ FLOPs
    for $i = i_{min} : i_{max}$ **do**
        $\Phi[i] \leftarrow \Phi\left(\frac{l_i - \mu_{\mathsf{homo}}}{\sigma_{\mathsf{homo}}}\right)$; # $O(UK)$ FLOPs
    end for
    for $v = 1 : V$ **do**
        Generate $\boldsymbol{\mathcal{E}}_{\mathsf{lmp}}^{(u,v)}$, $\boldsymbol{\epsilon}_{\mathsf{lmp}} \leftarrow \boldsymbol{\mathcal{E}}_{\mathsf{lmp}}^{(u,v)}$; # $O(UVN_{\mathsf{lmp}})$ FLOPs
        Compute $l_{m^{(u,v)}}$ based on $\boldsymbol{z}$ and $\boldsymbol{\epsilon}_{\mathsf{lmp}}$, $m[v] \leftarrow m^{(u,v)}$; # $O(UVN_{\mathsf{lmp}}C)$ FLOPs
        for $k = 0 : K-1$ **do**
            $p[k] \leftarrow p[k] + \Phi[k - m[v]]$; # $O(UVK)$ FLOPs
        end for
    end for
end for
$\boldsymbol{p} \leftarrow \boldsymbol{p}/(UV)$; # $O(K)$ FLOPs
return $\boldsymbol{p}$;

Algorithm 6.4 EXACT+CLT method to compute $\mathbb{P}\{\mathcal{L} \le l_k\}$, $k = 0, \ldots, K-1$

Input: Subportfolio Π_{homo}, Π_{lmp} and loss levels $\boldsymbol{l}$;

Output: $\boldsymbol{p}$;

$\boldsymbol{p} \leftarrow \mathbf{0}$;
Compute indices of minimum and maximum loss of the lumpy sub-portfolio i_{min} and i_{max}; # $O(N_{\mathsf{lmp}})$ FLOPs
for $u = 1 : U$ **do**
    Generate $\boldsymbol{\mathcal{Z}}^{(u)}$, $\boldsymbol{z} \leftarrow \boldsymbol{\mathcal{Z}}^{(u)}$; # $O(US)$ FLOPs
    Compute $\mu_{\mathsf{homo}}^{(u)}$ and $\sigma_{\mathsf{homo}}^{(u)}$ based on $\boldsymbol{z}$, $\mu_{\mathsf{homo}} \leftarrow \mu_{\mathsf{homo}}^{(u)}$, $\sigma_{\mathsf{homo}} \leftarrow \sigma_{\mathsf{homo}}^{(u)}$; # $O(UN_{\mathsf{homo}})$ FLOPs
    Compute $H_{N_{\mathsf{lmp}}-1}$, $\tilde{\boldsymbol{d}}_{\mathsf{lmp}}$, and $\boldsymbol{p}_{\mathsf{lmp}}^{(u)}$ based on $\boldsymbol{z}$ by an exact method, $\boldsymbol{p}_{\mathsf{lmp}} \leftarrow \boldsymbol{p}_{\mathsf{lmp}}^{(u)}$;
        # $O(UC^2N_{\mathsf{lmp}}^2)$ FLOPs (best case, SCONV)
        # $O(UC^{N_{\mathsf{lmp}}})$ FLOPs (worst case, SCONV)
    for $i = i_{min} : i_{max}$ **do**
        $\Phi[i] \leftarrow \Phi\left(\frac{l_i - \mu_{\mathsf{homo}}}{\sigma_{\mathsf{homo}}}\right)$; # $O(UK)$ FLOPs
    end for
    for $v = 0 : H_{N_{\mathsf{lmp}}-1} - 1$ **do**
        for $k = 0 : K-1$ **do**
            $p[k] \leftarrow p[k] + \Phi[k - \tilde{d}_{\mathsf{lmp}}[v]]p_{\mathsf{lmp}}[\tilde{d}_{\mathsf{lmp}}[v]]$; # $O(UKH_{N_{\mathsf{lmp}}-1})$ FLOPs
        end for
    end for
end for
$\boldsymbol{p} \leftarrow \boldsymbol{p}/U$; # $O(K)$ FLOPs
return $\boldsymbol{p}$;

6.3.1.2 Complexity Analysis

In this subsection, we examine the complexity of the hybrid methods to compute the whole loss distribution, $\mathbb{P}\{\mathcal{L} \le l_k\}$, for $k = 0, \ldots, K-1$, Our analysis is based on the following observations: $K \gg V \gg N_{\mathsf{lmp}} \gtrsim C$, and $UV \gtrsim K$.

- For the MC+LLN method, Algorithm 6.1 shows that the total complexity to compute the whole loss distribution is $O(U \log_2 VK)$, where we use the observation that $K \log_2 V \gtrsim V N_{\mathsf{Imp}} C$;

- For the MC+CLT method, Algorithm 6.3 shows that the total complexity to compute the whole loss distribution is $O(UVK)$, where we use the observation that $K \gg N_{\mathsf{Imp}} C$;

- For EXACT+LLN/CLT methods, Algorithm 6.2 and 6.4 show that the total complexity to compute the whole loss distribution in the best case is $O\left(UKH_{N_{\mathsf{Imp}}-1}\right) = O\left(UKN_{\mathsf{Imp}}C\right)$, where we use the observation that $KH_{N_{\mathsf{Imp}}-1} \gg C^2 N_{\mathsf{Imp}}^2$ and the fact $H_{N_{\mathsf{Imp}}-1} = N_{\mathsf{Imp}}C$ by (5.1.30). In the worst case, the complexity is $O\left(UH_{N_{\mathsf{Imp}}-1}K\right) = O\left(UC^{N_{\mathsf{Imp}}}K\right)$, where we use the fact $H_{N_{\mathsf{Imp}}-1} = C^{N_{\mathsf{Imp}}}$ by (5.1.34).

As a comparison, to compute the whole portfolio loss distribution, the pure MC method requires $O(UVNC)$ FLOPs[3], SCONV method requires $O\left(UN^2C^2\right)$ FLOPs in the best case, and $O\left(UC^N\right)$ FLOPs in the worst case[4] and the TR SCONV method requires much less computation than the SCONV method. Hence, if the goal is to compute the whole loss distribution, it is clear that all the hybrid methods, except possibly the MC+LLN method, have a very slim advantage in efficiency over the MC method, or may even be inferior to it.

6.3.2 Implementation of the Hybrid Methods to Compute VaR

6.3.2.1 Implementation Details

VaR of a portfolio can be easily determined once the loss distribution of the portfolio is calculated. However, it is not necessary to compute the whole loss distribution to determine VaR. Notice that the portfolio loss probability, $\mathbb{P}\{\mathcal{L} \le l_k\}$, as shown in (6.1.6) and (6.2.2) for the hybrid methods, is an increasing function of the loss percentage level, l_k, hence one can apply a bisection search to calculate VaR. The number of evaluations of the portfolio loss probability function is at most $\log_2 K$ in this case. This is much less than K, which is required to compute the whole loss distribution. Since we only need to evaluate the portfolio loss probability function at $\log_2 K$ loss percentage levels, rather than all K loss percentage levels, improvements designed to calculate the whole loss distribution in the last subsection are not efficient when calculating VaR.

We propose below two new schemes to implement the bisection search to calculate VaR based on the hybrid methods.

[3]The complexity of the pure Monte Carlo method to compute the loss distribution is similar with that of the Monte Carlo method applied to the lumpy sub-portfolio in the MC+CLT/LLN method.

[4]The complexity of the SCONV method to compute the loss distribution is similar to that of the SCONV method applied to the lumpy sub-portfolio in the EXACT+CLT/LLN method.

- **Memory-intensive Scheme** Calculate and store $l_{m^{(u,v)}}$, $u = 1, \ldots, U$ and $v = 1, \ldots, V$ for the MC+CLT/LLN method, or l_m and $p_{\mathsf{lmp}}^{(m,u)}$ $u = 1, \ldots, U$ and $m \in \tilde{\boldsymbol{d}}_{\mathsf{lmp}}$ for the EXACT+CLT/LLN method, then perform the bisection search based on the stored $l_{m^{(u,v)}}$, or l_m and $p_{\mathsf{lmp}}^{(m,u)}$. Algorithms based on this scheme are presented in Algorithm 6.5 for the MC+CLT/LLN method, and in Algorithm 6.6 for the EXACT+CLT/LLN method.

Algorithm 6.5 MC+CLT/LLN method to compute $\mathrm{VaR}_\gamma(\mathcal{L})$ (Memory-intensive Scheme)

Input: Subportfolio Π_{homo}, Π_{lmp} and loss levels $\boldsymbol{l}$, γ;
Output: VaR;

for $u = 1 : U$ **do**
  Generate $\boldsymbol{\mathcal{Z}}^{(u)}$, $\boldsymbol{z} \leftarrow \boldsymbol{\mathcal{Z}}^{(u)}$; # $O(US)$ FLOPs
  Compute $\mu_{\mathsf{homo}}^{(u)}$ and $\sigma_{\mathsf{homo}}^{(u)}$ based on $\boldsymbol{z}$, $\mu_{\mathsf{homo}}[u] \leftarrow \mu_{\mathsf{homo}}^{(u)}$, $\sigma_{\mathsf{homo}}[u] \leftarrow \sigma_{\mathsf{homo}}^{(u)}$; # $O(UN_{\mathsf{homo}})$ FLOPs
  for $v = 1 : V$ **do**
    Generate $\boldsymbol{\mathcal{E}}_{\mathsf{lmp}}^{(u,v)}$, $\boldsymbol{\epsilon}_{\mathsf{lmp}} \leftarrow \boldsymbol{\mathcal{E}}_{\mathsf{lmp}}^{(u,v)}$; # $O(UVN_{\mathsf{lmp}})$ FLOPs
    Compute $l_{m^{(u,v)}}$ based on $\boldsymbol{z}$ and $\boldsymbol{\epsilon}_{\mathsf{lmp}}$, $l[u,v] \leftarrow l_{m^{(u,v)}}$; # $O(UVN_{\mathsf{lmp}}C)$ FLOPs
  end for
end for

$k_{\mathsf{L}} \leftarrow 0$, $k_{\mathsf{R}} \leftarrow K - 1$;
while $k_{\mathsf{L}} \neq k_{\mathsf{R}}$ **do**
  $k_{\mathsf{M}} \leftarrow \lceil (k_{\mathsf{L}} + k_{\mathsf{R}})/2 \rceil$;
  $p \leftarrow 0$; # $O(\log_2(K))$ FLOPs
  for $u = 1 : U$ **do**
    for $v = 1 : V$ **do**
      $p \leftarrow \begin{cases} p + \Phi\left(\frac{l[k_{\mathsf{M}}] - l_{\mathsf{lmp}}[u,v] - \mu_{\mathsf{homo}}[u]}{\sigma_{\mathsf{homo}}[u]}\right), & \text{MC+CLT}, \\ p + \mathbb{I}_{\{\mu_{\mathsf{homo}}[u] \le l[k_{\mathsf{M}}] - l_{\mathsf{lmp}}[u,v]\}}, & \text{MC+LLN}, \end{cases}$ # $O(UV\log_2(K))$ FLOPs
    end for
  end for
  $p \leftarrow p/(UV)$; # $O(\log_2(K))$ FLOPs
  if $p < \gamma$ **then**
    $k_{\mathsf{L}} \leftarrow k_{\mathsf{M}}$;
  else
    $k_{\mathsf{R}} \leftarrow k_{\mathsf{M}}$;
  end if
end while

VaR $\leftarrow l[k_{\mathsf{M}}]$;
return VaR;

Algorithm 6.6 EXACT+CLT/LLN method to compute $\mathrm{VaR}_\gamma(\mathcal{L})$ (Memory-intensive Scheme)

Input: Subportfolio Π_{homo}, Π_{Imp} and loss levels $\boldsymbol{l}$, γ;

Output: VaR;

for $u = 1 : U$ **do**
  Generate $\boldsymbol{\mathcal{Z}}^{(u)}$, $\boldsymbol{z} \leftarrow \boldsymbol{\mathcal{Z}}^{(u)}$; # $O(US)$ FLOPs
  Compute $\mu_{\mathsf{homo}}^{(u)}$ and $\sigma_{\mathsf{homo}}^{(u)}$ based on $\boldsymbol{z}$, $\mu_{\mathsf{homo}}[u] \leftarrow \mu_{\mathsf{homo}}^{(u)}$, $\sigma_{\mathsf{homo}}[u] \leftarrow \sigma_{\mathsf{homo}}^{(u)}$; # $O(UN_{\mathsf{homo}})$ FLOPs
  Compute $H_{N_{\mathsf{Imp}}-1}$, $\tilde{\boldsymbol{d}}_{\mathsf{Imp}}$, and $\boldsymbol{p}_{\mathsf{Imp}}^{(u)}$ based on $\boldsymbol{z}$ by an exact method, $\boldsymbol{p}_{\mathsf{Imp}}[u,:] \leftarrow \boldsymbol{p}_{\mathsf{Imp}}^{(u)}$;
  # $O(UC^2N_{\mathsf{Imp}}^2)$ FLOPs (best case, SCONV)
  # $O(UC^{N_{\mathsf{Imp}}})$ FLOPs (worst case, SCONV)
end for

$k_{\mathsf{L}} \leftarrow 0$, $k_{\mathsf{R}} \leftarrow K-1$;
while $k_{\mathsf{L}} \neq k_{\mathsf{R}}$ **do**
  $k_{\mathsf{M}} \leftarrow \lceil (k_{\mathsf{L}} + k_{\mathsf{R}})/2 \rceil$; # $O(\log_2(K))$ FLOPs
  $p \leftarrow 0$;
  for $u = 1 : U$ **do**
    for $v = 0 : H_{N_{\mathsf{Imp}}-1} - 1$ **do**
      $p \leftarrow \begin{cases} p + \Phi\left(\frac{l[k]-l_{\mathsf{Imp}}[u,\tilde{d}_{\mathsf{Imp}}[v]]-\mu_{\mathsf{homo}}[u]}{\sigma_{\mathsf{homo}}[u]}\right) p_{\mathsf{Imp}}[u,\tilde{d}_{\mathsf{Imp}}[v]], & \text{EXACT+CLT}, \\ p + \mathbb{I}_{\{\mu_{\mathsf{homo}}[u] \le l[k]-l_{\mathsf{Imp}}[u,\tilde{d}_{\mathsf{Imp}}[v]]\}} p_{\mathsf{Imp}}[u,\tilde{d}_{\mathsf{Imp}}[v]], & \text{EXACT+LLN}, \end{cases}$ # $O(UH_{N_{\mathsf{Imp}}-1}\log_2(K))$ FLOPs
    end for
  end for
  $p \leftarrow p/V$; # $O(\log_2(K))$ FLOPs
  if $p < \gamma$ **then**
    $k_{\mathsf{L}} \leftarrow k_{\mathsf{M}}$;
  else
    $k_{\mathsf{R}} \leftarrow k_{\mathsf{M}}$;
  end if
end while

VaR $\leftarrow l[k_{\mathsf{M}}]$;
return VaR;

- **Memory-saving Scheme** For each evaluation of the portfolio loss probability function, recalculate $l_{m^{(u,v)}}$, $u = 1, \ldots, U$ and $v = 1, \ldots, V$, using the same random seed for the MC+CLT/LLN method, or l_m and $p_{\mathsf{Imp}}^{(m,u)}$, $u = 1, \ldots, U$ and $m \in \tilde{\boldsymbol{d}}_{\mathsf{Imp}}$, for the EXACT+CLT/LLN method. Algorithms based on this scheme are presented in Algorithm 6.7 for the MC+CLT/LLN method, and in Algorithm 6.8 for the EXACT+CLT/LLN method.

Algorithm 6.7 MC+CLT/LLN method to compute $\mathrm{VaR}_\gamma(\mathcal{L})$ (Memory-saving Scheme)

Input: Subportfolio Π_{homo}, Π_{Imp} and loss levels $\boldsymbol{l}$, γ;

Output: VaR;

$k_{\mathsf{L}} \leftarrow 0$, $k_{\mathsf{R}} \leftarrow K-1$;
while $k_{\mathsf{L}} \neq k_{\mathsf{R}}$ **do**
  $k_{\mathsf{M}} \leftarrow \lceil (k_{\mathsf{L}} + k_{\mathsf{R}})/2 \rceil$; # $O(\log_2(K))$ FLOPs
  $p \leftarrow 0$;
  for $u = 1 : U$ **do**
    Generate $\boldsymbol{\mathcal{Z}}^{(u)}$, $\boldsymbol{z} \leftarrow \boldsymbol{\mathcal{Z}}^{(u)}$ with seed 0; # $O(US\log_2(K))$ FLOPs
    Compute $\mu_{\mathsf{homo}}^{(u)}$ and $\sigma_{\mathsf{homo}}^{(u)}$ based on $\boldsymbol{z}$, $\mu_{\mathsf{homo}} \leftarrow \mu_{\mathsf{homo}}^{(u)}$, $\sigma_{\mathsf{homo}} \leftarrow \sigma_{\mathsf{homo}}^{(u)}$; # $O(UN_{\mathsf{homo}}\log_2(K))$ FLOPs
    for $v = 1 : V$ **do**
      Generate $\boldsymbol{\mathcal{E}}_{\mathsf{Imp}}^{(u,v)}$, $\boldsymbol{\epsilon}_{\mathsf{Imp}} \leftarrow \boldsymbol{\mathcal{E}}_{\mathsf{Imp}}^{(u,v)}$; # $O(UVN_{\mathsf{Imp}}\log_2(K))$ FLOPs
      Compute $l_{m^{(u,v)}}$ based on $\boldsymbol{z}$ and $\boldsymbol{\epsilon}_{\mathsf{Imp}}$, $l[v] \leftarrow l_{m^{(u,v)}}$; # $O(UVN_{\mathsf{Imp}}C\log_2(K))$ FLOPs
      $p \leftarrow \begin{cases} p + \Phi\left(\frac{l[k_{\mathsf{M}}] - l_{\mathsf{Imp}}[v] - \mu_{\mathsf{homo}}}{\sigma_{\mathsf{homo}}}\right), & \text{MC+CLT}, \\ p + \mathbb{I}_{\{\mu_{\mathsf{homo}} \le l[k_{\mathsf{M}}] - l_{\mathsf{Imp}}[v]\}}, & \text{MC+LLN}, \end{cases}$ # $O(UV\log_2(K))$ FLOPs
    end for
  end for
  $p \leftarrow p/(UV)$; # $O(\log_2(K))$ FLOPs
  if $p < \gamma$ **then**
    $k_{\mathsf{L}} \leftarrow k_{\mathsf{M}}$;
  else
    $k_{\mathsf{R}} \leftarrow k_{\mathsf{M}}$;
  end if
end while

VaR $\leftarrow l[k_{\mathsf{M}}]$;
return VaR;

Algorithm 6.8 EXACT+CLT/LLN method to compute $\mathrm{VaR}_\gamma(\mathcal{L})$ (Memory-saving Scheme)

Input: Subportfolio Π_{homo}, Π_{Imp} and loss levels $\boldsymbol{l}$, γ;

Output: VaR;

$k_{\mathsf{L}} \leftarrow 0$, $k_{\mathsf{R}} \leftarrow K-1$;
while $k_{\mathsf{L}} \neq k_{\mathsf{R}}$ **do**
  $k_{\mathsf{M}} \leftarrow \lceil (k_{\mathsf{L}} + k_{\mathsf{R}})/2 \rceil$; # $O(\log_2(K))$ FLOPs
  $p \leftarrow 0$;
  for $u = 1 : U$ **do**
    Generate $\boldsymbol{\mathcal{Z}}^{(u)}$, $\boldsymbol{z} \leftarrow \boldsymbol{\mathcal{Z}}^{(u)}$; # $O(US\log_2(K))$ FLOPs
    Compute $\mu_{\mathsf{homo}}^{(u)}$ and $\sigma_{\mathsf{homo}}^{(u)}$ based on $\boldsymbol{z}$, $\mu_{\mathsf{homo}} \leftarrow \mu_{\mathsf{homo}}^{(u)}$, $\sigma_{\mathsf{homo}} \leftarrow \sigma_{\mathsf{homo}}^{(u)}$; # $O(UN_{\mathsf{homo}}\log_2(K))$ FLOPs
    Compute $H_{N_{\mathsf{Imp}}-1}$, $\tilde{\boldsymbol{d}}_{\mathsf{Imp}}$, and $\boldsymbol{p}_{\mathsf{Imp}}^{(u)}$ based on $\boldsymbol{z}$ by an exact method, $\boldsymbol{p}_{\mathsf{Imp}} \leftarrow \boldsymbol{p}_{\mathsf{Imp}}^{(u)}$;
      # $O(UC^2N_{\mathsf{Imp}}^2\log_2(K))$ FLOPs (best case, SCONV)
      # $O(UC^{N_{\mathsf{Imp}}}\log_2(K))$ FLOPs (worst case, SCONV)
    for $v = 0 : H_{N_{\mathsf{Imp}}-1} - 1$ **do**
      $p \leftarrow \begin{cases} p + \Phi\left(\frac{l[k]-l_{\mathsf{Imp}}[u,\tilde{d}_{\mathsf{Imp}}[v]]-\mu_{\mathsf{homo}}}{\sigma_{\mathsf{homo}}}\right) p_{\mathsf{Imp}}[\tilde{d}_{\mathsf{Imp}}[v]], & \text{EXACT+CLT}, \\ p + \mathbb{I}_{\{\mu_{\mathsf{homo}}[u] \le l[k]-l_{\mathsf{Imp}}\tilde{d}_{\mathsf{Imp}}[v]]\}} p_{\mathsf{Imp}}[\tilde{d}_{\mathsf{Imp}}[v]], & \text{EXACT+LLN}, \end{cases}$ # $O(UH_{N_{\mathsf{Imp}}-1}\log_2(K))$ FLOPs
    end for
  end for
  $p \leftarrow p/V$; # $O(\log_2(K))$ FLOPs
  if $p < \gamma$ **then**
    $k_{\mathsf{L}} \leftarrow k_{\mathsf{M}}$;
  else
    $k_{\mathsf{R}} \leftarrow k_{\mathsf{M}}$;
  end if
end while

$\mathrm{VaR} \leftarrow l[k_{\mathsf{M}}]$;
return VaR;

Compared to the memory-saving schemes, the memory-intensive schemes reduce the number of FLOPs by a factor of $\log_2 K$ but increases the memory usage by a factor of U. As we discussed in the last subsection, memory access to a very long array could be very slow, which could offset the advantage of fewer FLOPs. Therefore, we believe the memory-saving scheme is preferable.

6.3.2.2 Complexity Analysis

In this subsection, we examine the complexity of the hybrid methods to compute VaR, $\mathrm{VaR}_\gamma(\mathcal{L})$. Our analysis is based on the following observations: $K \gg V \gg N_{\mathsf{Imp}} \gtrsim C$, and $UV \gtrsim K$.

- For the MC+CLT/LLN method, Algorithm 6.7 shows that the total complexity to compute VaR is $O\left(UVN_{\mathsf{Imp}}C\log_2 K\right)$;
- For the EXACT+LLN/CLT method, Algorithm 6.8 shows that the total complexity to compute VaR in the best case is $O\left(UC^2N_{\mathsf{Imp}}^2\log_2 K\right)$. In the worst case, the complexity is $O\left(UH_{N_{\mathsf{Imp}}-1}\log_2 K\right) = O\left(UC^{N_{\mathsf{Imp}}}\log_2 K\right)$, where we use the fact $H_{N_{\mathsf{Imp}}-1} = C^{N_{\mathsf{Imp}}}$ by (5.1.34).

The complexity of the benchmark pure MC method and the (TR) SCONV method to calculate VaR is similar to that required to calculate the whole distribution. Therefore, if the goal is to compute VaR, it is clear that all hybrid methods are more efficient than the benchmark pure MC method and (TR) SCONV method.

6.4 Numerical Results and Comparisons

We end this chapter with a comparison of the accuracy and efficiency of the hybrid methods developed in this chapter and the MC, asymptotic and exact methods developed in earlier chapters. The comparison includes the computation of both the loss distribution and VaR with different confidence levels.

As mention in Subsection 6.1, a lumpy portfolio is a portfolio consisting of two sub-portfolios: a large, fairly homogeneous sub-portfolio and a small heterogeneous sub-portfolio, where the obligors in the fairly homogeneous sub-portfolio have about the same EAD, while the obligors in the heterogeneous sub-portfolio have substantially larger EAD. For our tests, we construct eight testing lumpy portfolios differing in the following factors:

- Number of obligors in the homogeneous sub-portfolio: 64 or 2048;
- Homogeneity of the homogeneous sub-portfolio: homogeneous or inhomogeneous, depending on whether β_n are identical or randomly generated from $\text{Unif}(-1, 1)$;
- Size of rating classes: 2 or 17.

For all the portfolios, all the obligors start in the highest rating class. In order to avoid discretization errors, the loss-given-credit-event, LGC_n^c, and the EAD, EAD_n, are specified such that each L_n^c is on the discretization grid for the loss with length of $K = 2^{16} = 65536$ constructed by (5.1.13) and (5.1.14). The following table summarizes the portfolios used in our tests.

Table 6.1: Testing portfolios for hybrids methods against exact/MC methods

Portfolio	C	$c(n)$	LGC_n^c	Homogeneous sub-portfolio				Inhomogeneous sub-portfolio			
				N_{homo}	EAD_n	β_n	$\sum \text{EAD}_n$	N_{Imp}	EAD_n	β_n	$\sum \text{EAD}_n$
64-1	2	2^0	$1-\frac{c}{2^0}$	64	1	0.5	64	16	$\widetilde{\text{EAD}^{64}}_n$	0.5	448
64-2	17	2^4	$1-\frac{c}{2^4}$								
64-3	2	2^0	$1-\frac{c}{2^0}$			Unif$(-1,1)$				Unif$(-1,1)$	
64-4	17	2^4	$1-\frac{c}{2^4}$								
2048-1	2	2^0	$1-\frac{c}{2^0}$	2048	1	0.5	2048	64	$\widetilde{\text{EAD}^{2048}}_n$	0.5	2048
2048-2	17	2^4	$1-\frac{c}{2^4}$								
2048-3	2	2^0	$1-\frac{c}{2^0}$			Unif$(-1,1)$				Unif$(-1,1)$	
2048-4	17	2^4	$1-\frac{c}{2^4}$								

The credit migration matrices used in the tests in this subsection are the migration matrix in Table 5.16 for $C = 2$, and the submatrix of the migration matrix in Table 5.17 for for $C = 17$.

In the portfolios with 64 obligors in the homogeneous part, there are 16 obligors in the inhomogeneous part. In the portfolios with 2048 obligors in the homogeneous part, there are 64 obligors in the inhomogeneous part. The EADs of obligors in the inhomogeneous sub-portfolio are generated in a way similar to that used in Subsection 5.3.4. That is, we specify the EADs as follows.

Step 1 For $n = 0, \ldots, N_{\mathsf{Imp}} - 1$, generate $X_n \sim \exp(10)$;

Step 2 For $n = 0, \ldots, N_{\mathsf{Imp}} - 1$, calculate

$$Y_n = \begin{cases} \frac{X_n}{\sum_{n=0}^{N-1} X_n} \times 448, & \text{for } \widetilde{\text{EAD}^{64}}_n \\ \frac{X_n}{\sum_{n=0}^{N-1} X_n} \times 2048, & \text{for } \widetilde{\text{EAD}^{2048}}_n \end{cases};$$

Step 3 For $n = 1, \ldots, N_{\mathsf{Imp}} - 1$, calculate

$$\widetilde{\text{EAD}^{64,2048}}_n = \begin{cases} \lceil Y_n \rceil; & \text{if } Y_n < \text{medium}(\boldsymbol{Y}) \\ \lfloor Y_n \rfloor; & \text{if } Y_n \geq \text{medium}(\boldsymbol{Y}) \end{cases};$$

Step 4 Calculate

$$\widetilde{\mathrm{EAD}^{64}}_0 = 448 - \sum_{n=1}^{N_{\mathrm{Imp}}-1} \widetilde{\mathrm{EAD}^{64}}_n,$$
$$\widetilde{\mathrm{EAD}^{2048}}_0 = 2048 - \sum_{n=1}^{N_{\mathrm{Imp}}-1} \widetilde{\mathrm{EAD}^{2048}}_n;$$

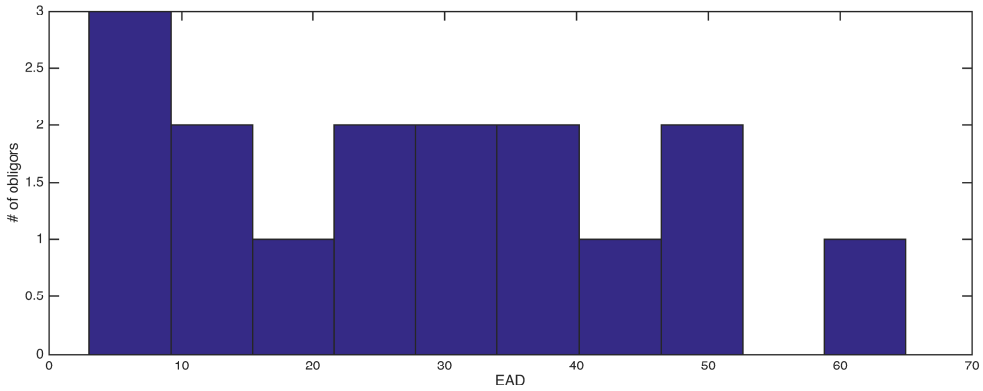
Step 5 If $\widetilde{\mathrm{EAD}^{64,2048}}_0 > 0$, stop, else repeat Step 1 to 5.

As an illustration, Figures 6.1a and 6.1b show the histograms for $\widetilde{\mathrm{EAD}^{64}}_n$ and $\widetilde{\mathrm{EAD}^{2048}}_n$, respectively

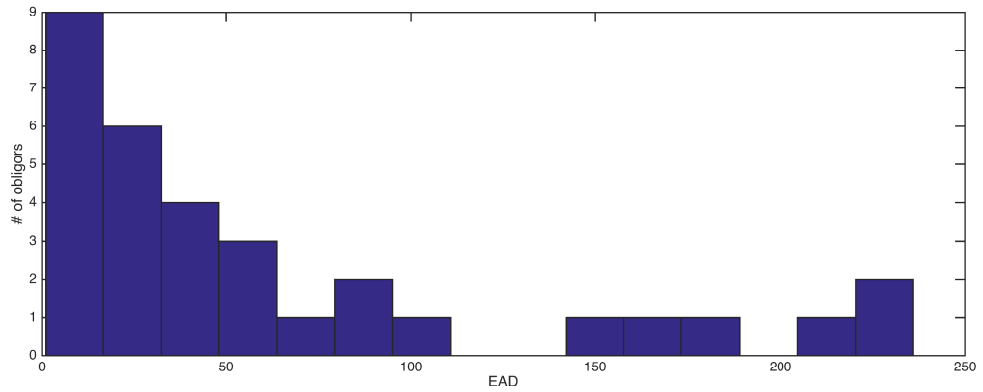
.

Figure 6.1: Histogram of EADs for inhomogeneous portfolios

(a) Histogram of $\widetilde{\mathrm{EAD}^{64}}_n$



(b) Histogram of $\widetilde{\mathrm{EAD}^{2048}}_n$



6.4.1 Comparison for Computing the Loss Distribution

For each constructed portfolio, we compute the cumulative loss probabilities

$$\hat{p}^{method}[k] \doteq \mathbb{P}\{\mathcal{L} \leq l_k\}$$

for $k = 0, \ldots, K-1$ using the LLN, CLT, Hybrid MC+LLN (Algorithm 6.1), Hybrid MC+CLT (Algorithm 6.3), MC, TR SCONV, and SCONV method. For the TR SCONV method, we let the threshold for the errors in the cumulative loss probabilities be $Tol = 10^{-4}$, and set ϵ by (5.1.51). For the MC method, we set the sample size to $U = 2^{14}$, $V = 2^{12}$.

6.4.1.1 Accuracy

Figures 6.2 - 6.9 compare the loss probabilities $\mathbb{P}\{\mathcal{L} \leq l_k\}$ and $\mathbb{P}\{\mathcal{L} > l_k\}$ for different testing portfolios computed by different methods. Since the tail is very close to zero and very flat, we plot $\mathbb{P}\{\mathcal{L} > l\}$ on a semi-log scale. To further assess the accuracy, we use SCONV method as the benchmark, and compute the absolute errors in the loss probabilities by:

$$\delta_k^{Prob,method} = \left|\hat{p}^{method}[k] - \hat{p}^{SCONV}[k]\right|.$$

Figures 6.10 - 6.13 present these absolute errors.

We make the following observations about the numerical results in Figures 6.2 - 6.13.

1. Both the CLT approximation and the LLN approximation fail to produce accurate loss distributions. The distributions generated by the MC+CLT approximation and by the MC+LLN approximation are much closer to the true distributions, and the accuracy of these two hybrid methods is comparable with that of the pure MC simulation. This shows that the hybrid methods' use of a MC simulation for the lumpy sub-portfolio is effective, and the hybrid approximations are indeed superior to the corresponding pure asymptotic approximations in terms of accuracy.

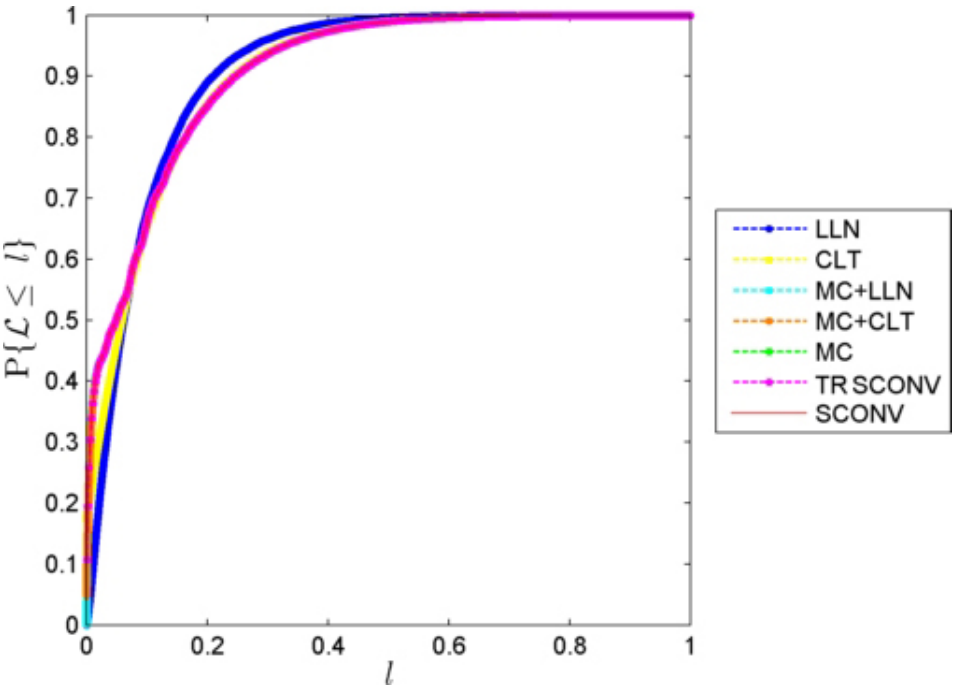
2. When the loss level l_k is not too big, the TR SCONV method produces the most accurate loss probabilities. As the quantile l_k increases, errors associated with the two hybrid approximations decrease, while errors associated with the TR SCONV method increase. For our testing portfolios, given the selected sample sizes and drop tolerance, the two hybrid approximations tend to outperform the TR SCONV method to produce extremely large loss probabilities in the right tail of the loss distribution ($\mathbb{P}\{\mathcal{L} \leq l_k\} \geq 1 - 10^{-8}$).

3. From Figures 6.10 - 6.13, we see that the errors associated with the MC+CLT approximation are usually smaller than those for all approximations other than TR SCONV. Errors associated with the MC+LLN approximation are smaller than those associated with the CLT approximation and the LLN approximation. Though errors associated with the MC+LLN approximation are generally larger than those associated with the MC+CLT, the difference in accuracy for two hybrid methods

is not material, especially at the high probability end.

Figure 6.2: Comparison of the loss distribution

(a) $\mathbb{P}\{\mathcal{L} \leq l\}$: Portfolio 64-1



(b) $\mathbb{P}\{\mathcal{L} > l\}$: Portfolio 64-1

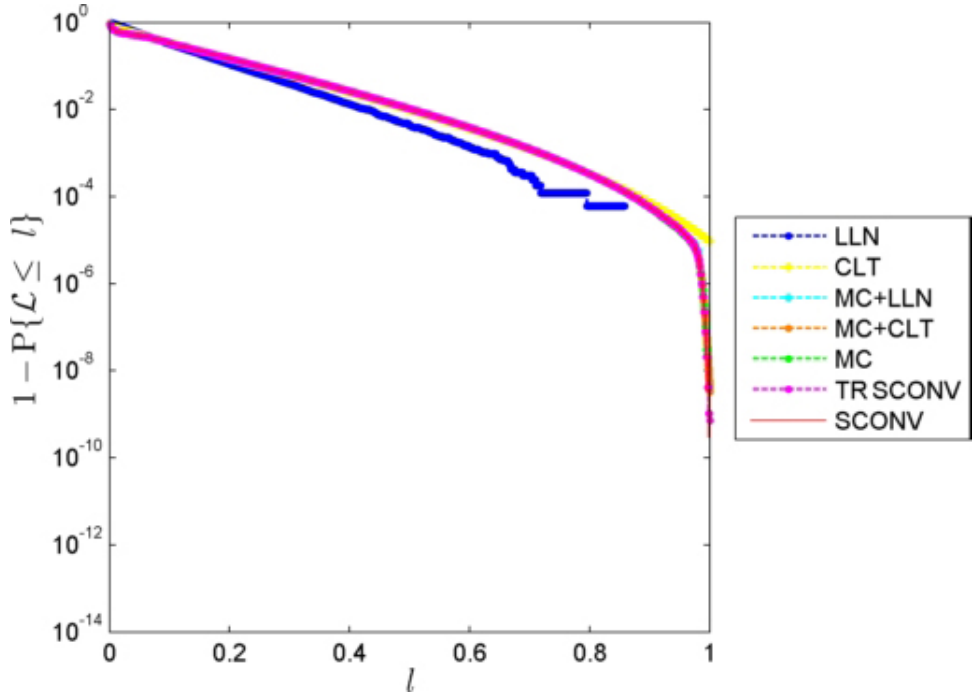
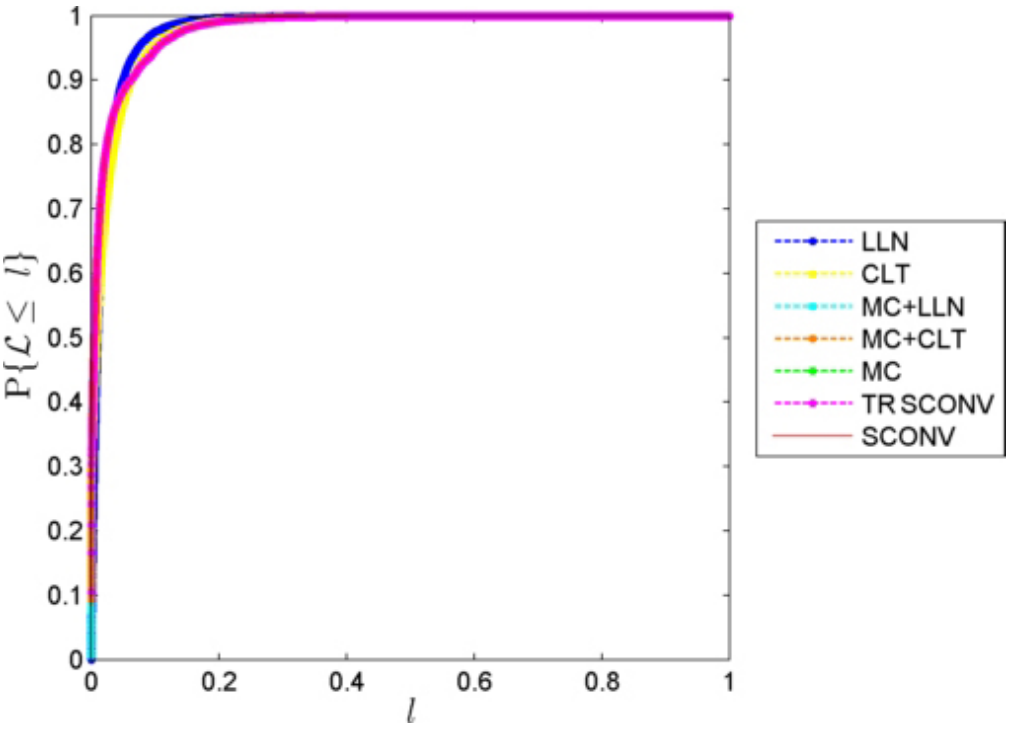


Figure 6.3: Comparison of the loss distribution (Cont'd)

(a) $\mathbb{P}\{\mathcal{L} \leq l\}$: Portfolio 64-2



(b) $\mathbb{P}\{\mathcal{L} > l\}$: Portfolio 64-2

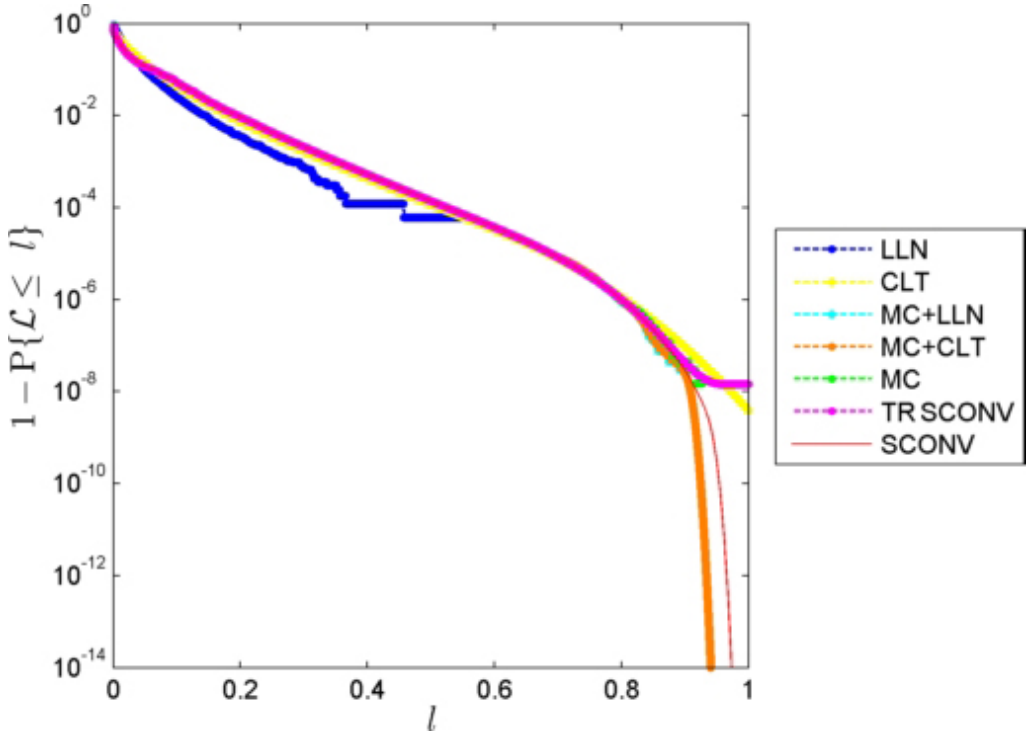
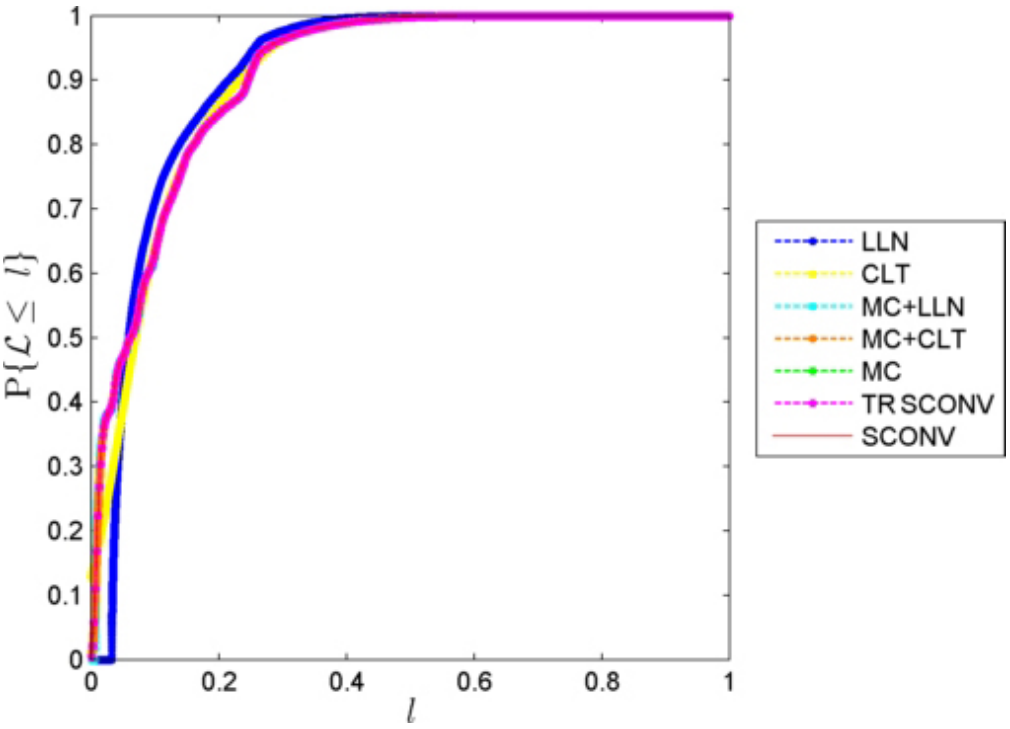


Figure 6.4: Comparison of the loss distribution (Cont'd)

(a) $\mathbb{P}\{\mathcal{L} \leq l\}$: Portfolio 64-3



(b) $\mathbb{P}\{\mathcal{L} > l\}$: Portfolio 64-3

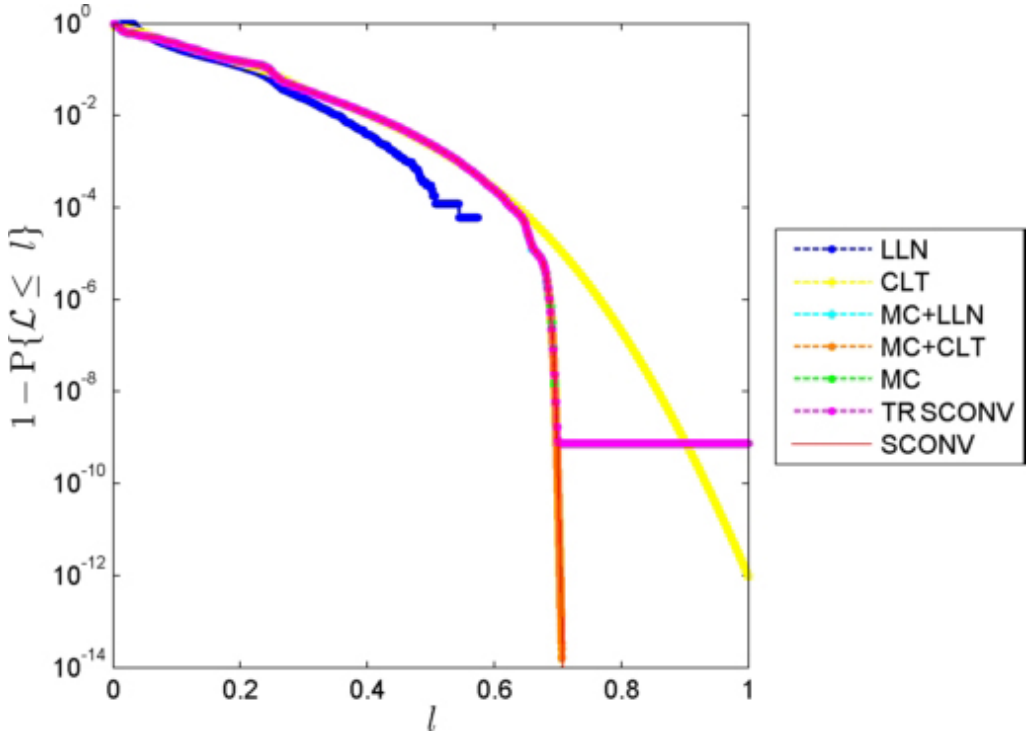
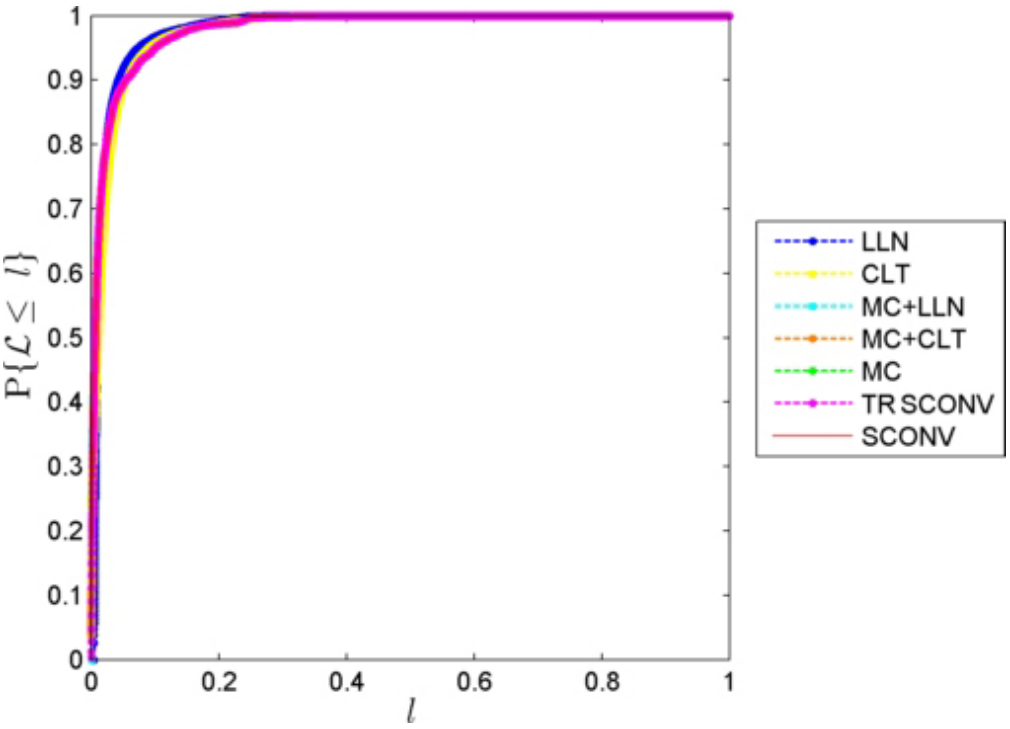


Figure 6.5: Comparison of the loss distribution (Cont'd)

(a) $\mathbb{P}\{\mathcal{L} \leq l\}$: Portfolio 64-4



(b) $\mathbb{P}\{\mathcal{L} > l\}$: Portfolio 64-4

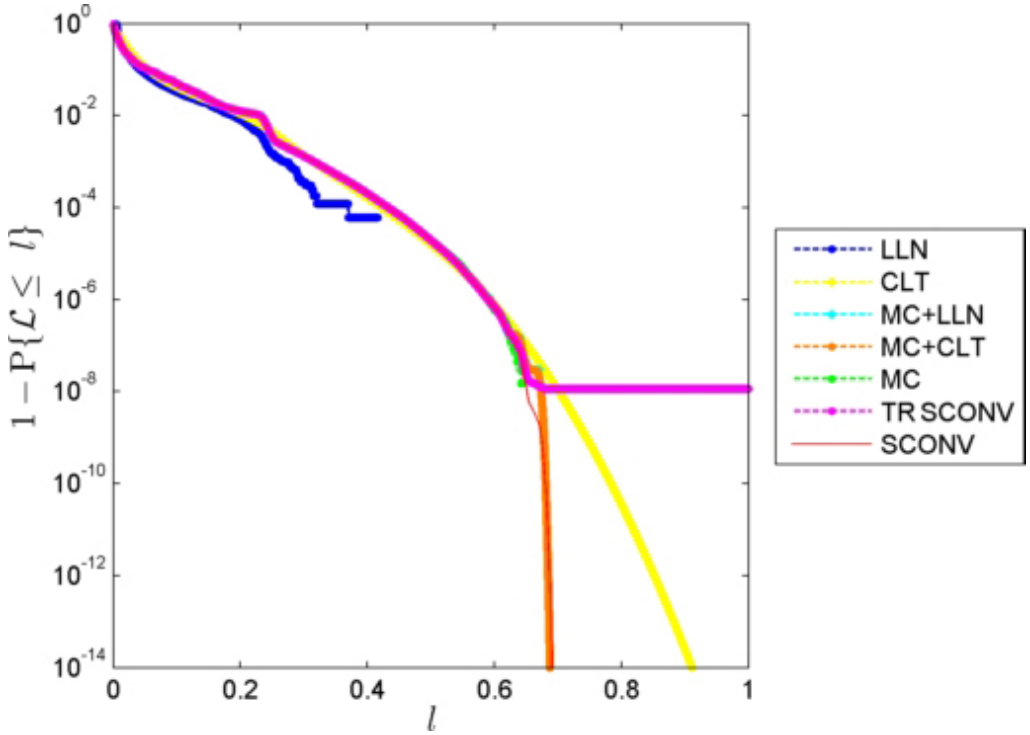
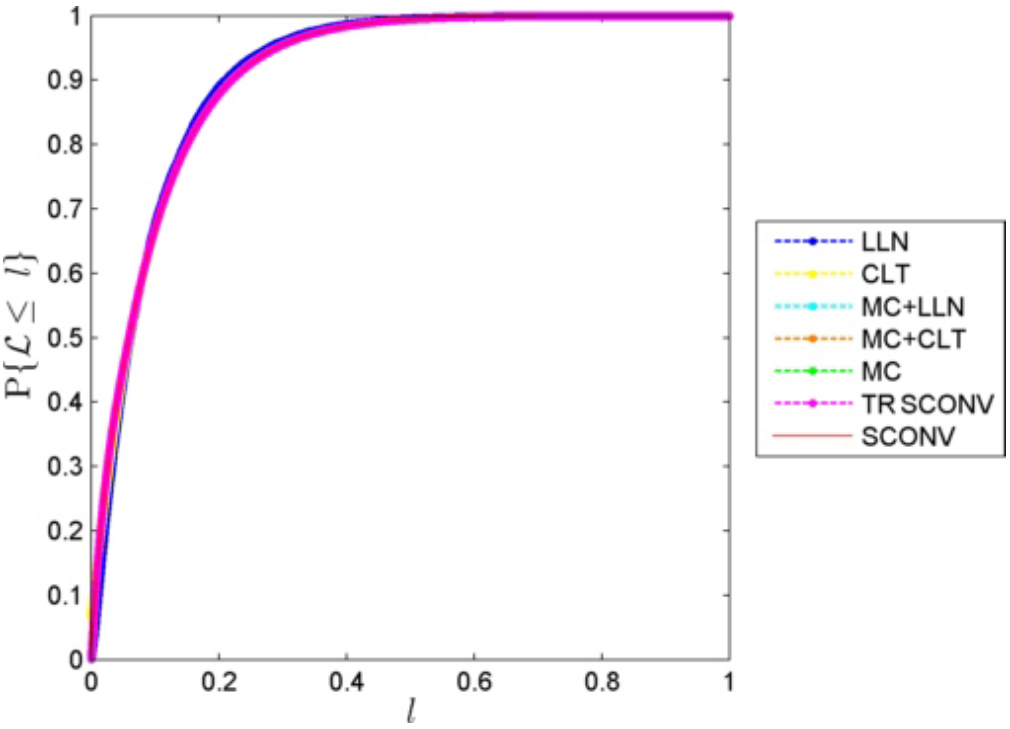


Figure 6.6: Comparison of the loss distribution (Cont'd)

(a) $\mathbb{P}\{\mathcal{L} \leq l\}$: Portfolio 2048-1



(b) $\mathbb{P}\{\mathcal{L} > l\}$: Portfolio 2048-1

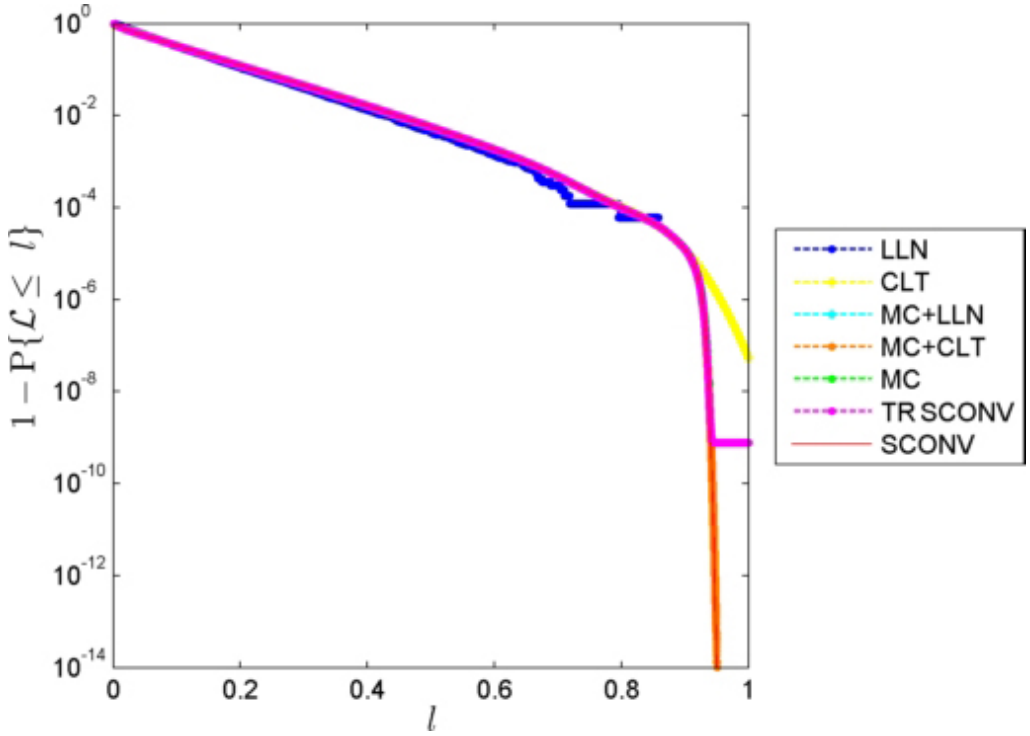
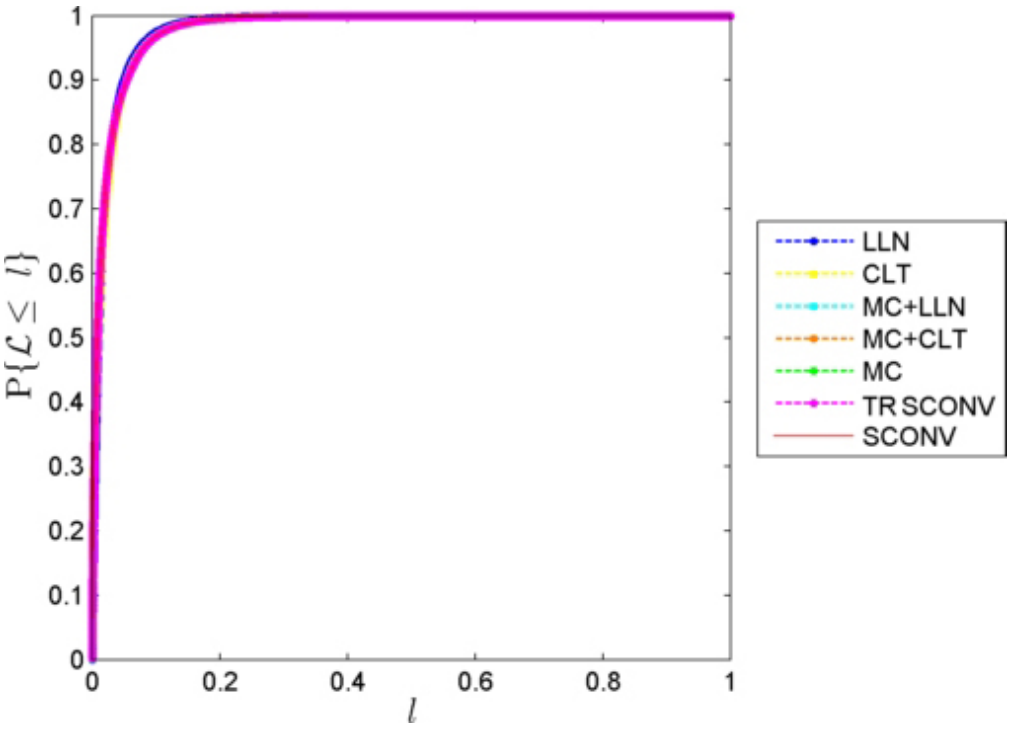


Figure 6.7: Comparison of the loss distribution (Cont'd)

(a) $\mathbb{P}\{\mathcal{L} \leq l\}$: Portfolio 2048-2



(b) $\mathbb{P}\{\mathcal{L} > l\}$: Portfolio 2048-2

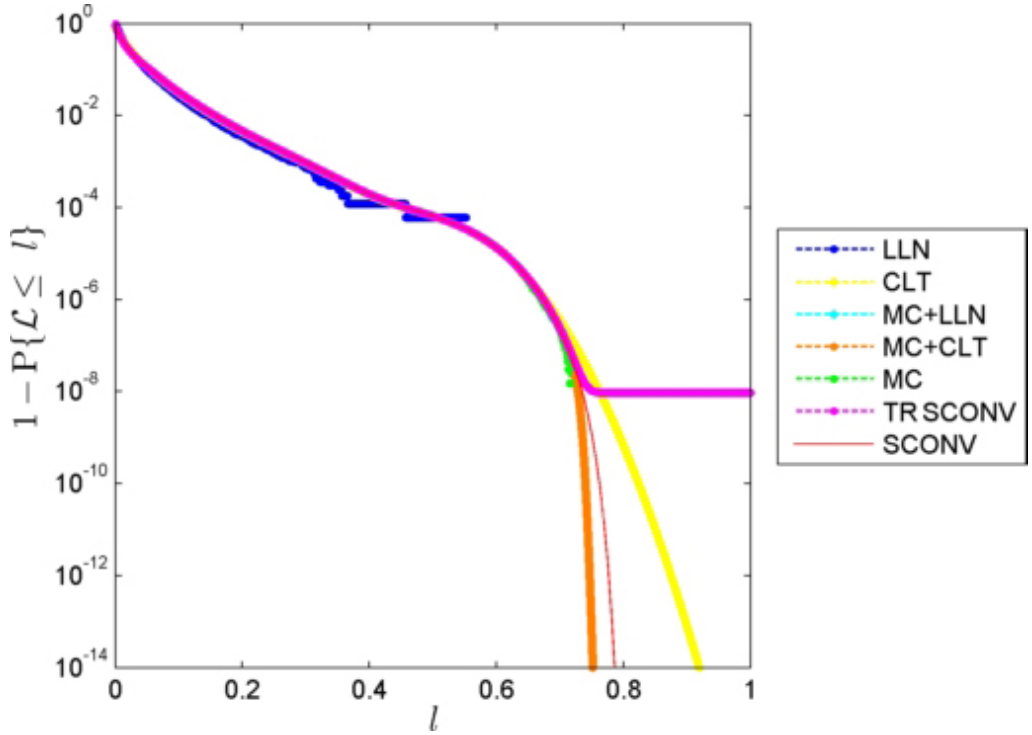
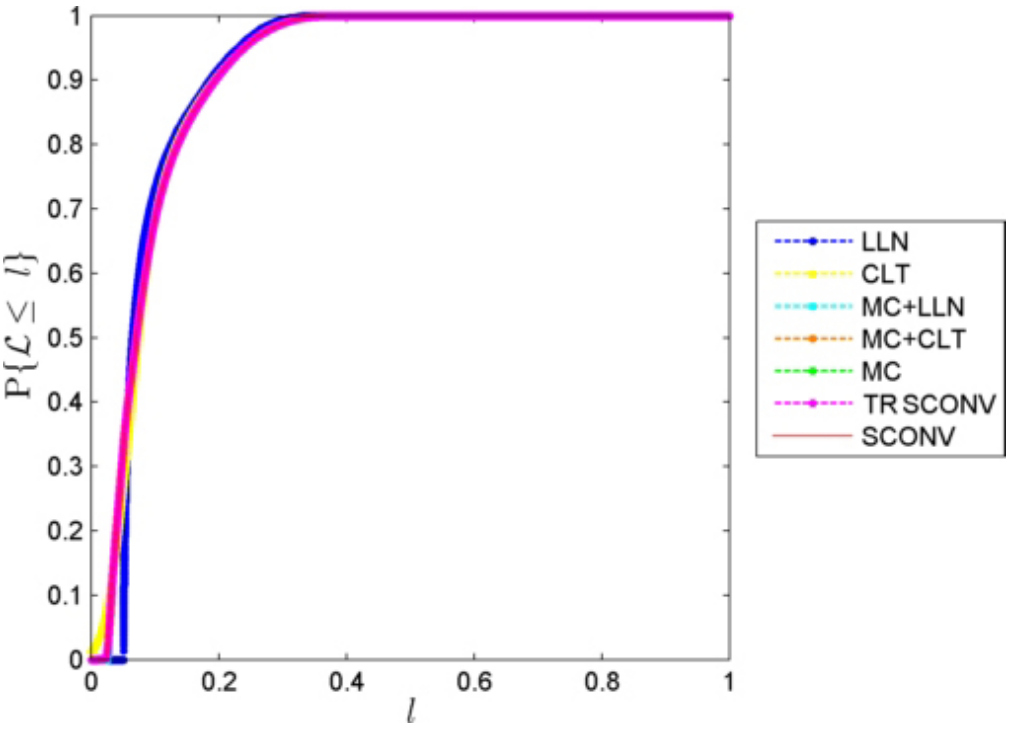


Figure 6.8: Comparison of the loss distribution (Cont'd)

(a) $\mathbb{P}\{\mathcal{L} \leq l\}$: Portfolio 2048-3



(b) $\mathbb{P}\{\mathcal{L} > l\}$: Portfolio 2048-3

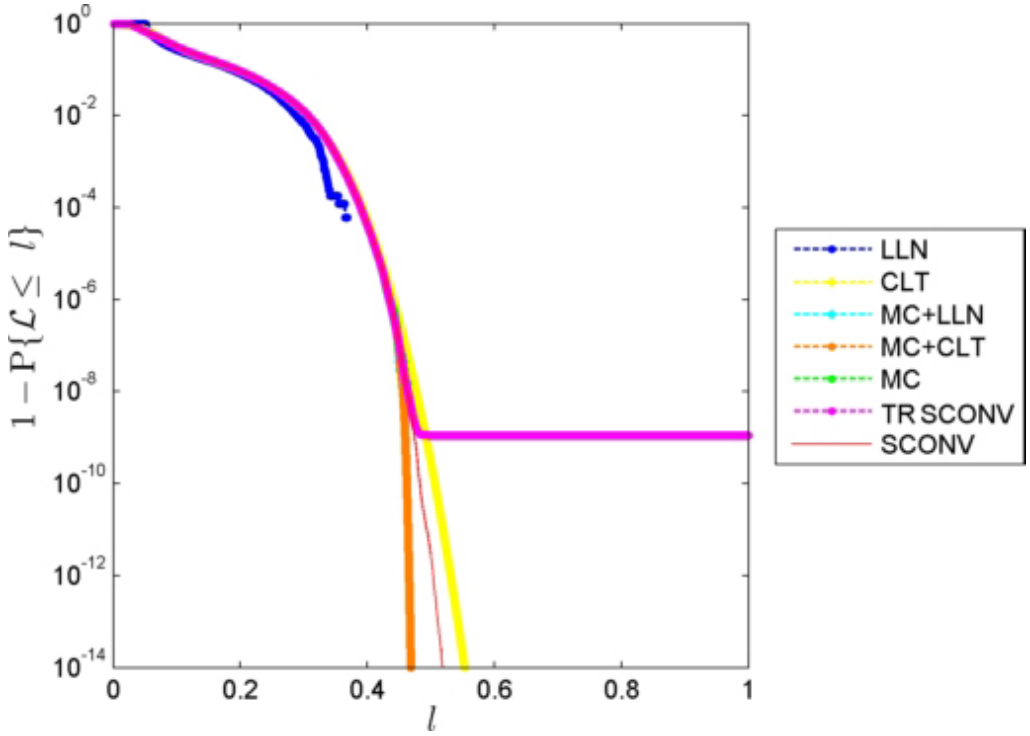
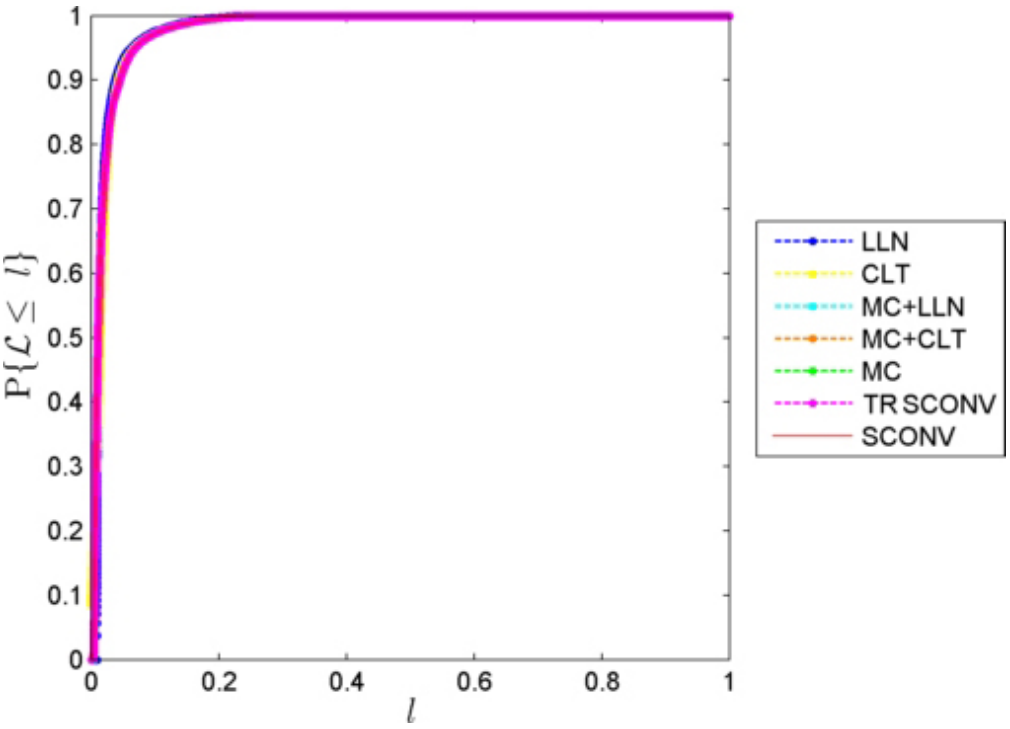


Figure 6.9: Comparison of the loss distribution (Cont'd)

(a) $\mathbb{P}\{\mathcal{L} \leq l\}$: Portfolio 2048-4



(b) $\mathbb{P}\{\mathcal{L} > l\}$: Portfolio 2048-4

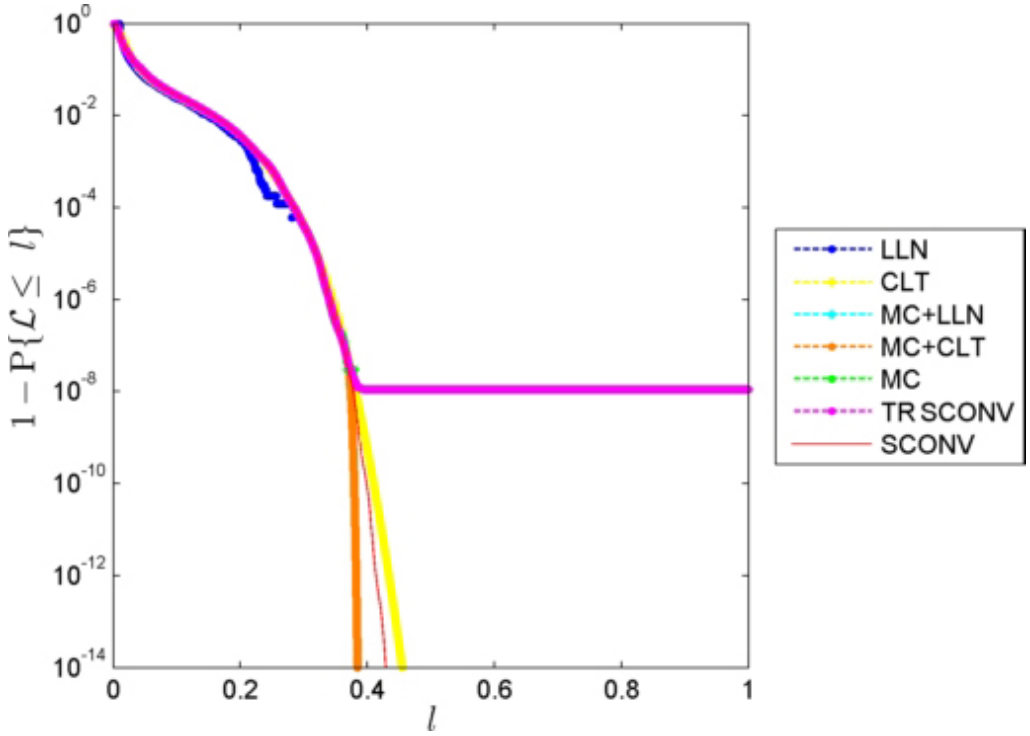
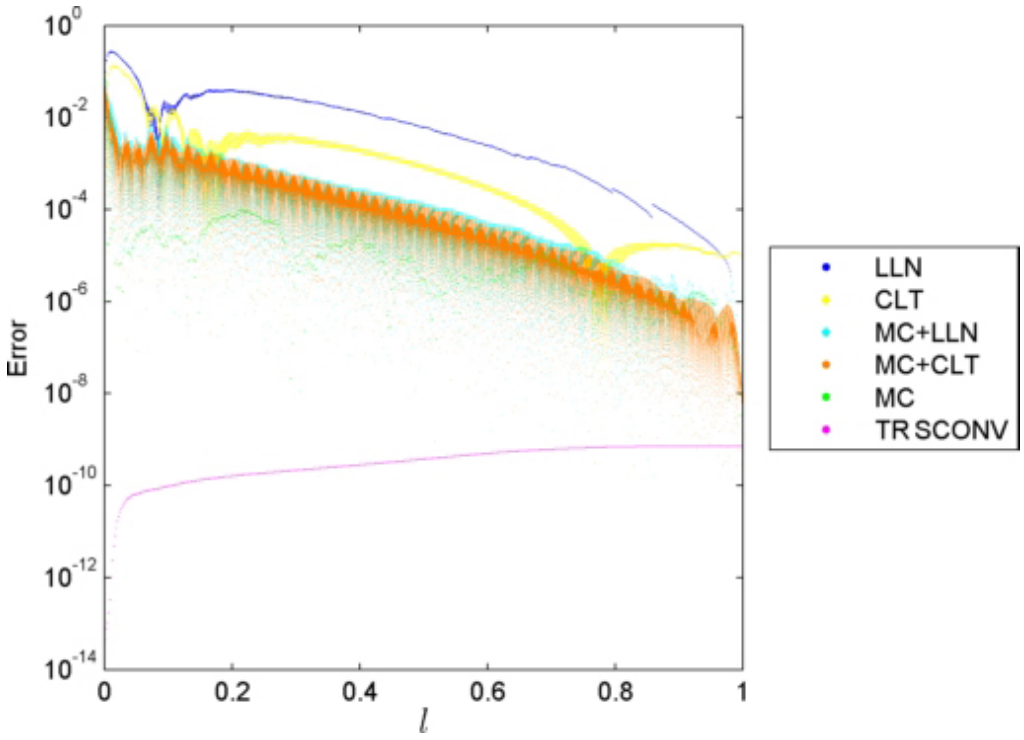


Figure 6.10: Comparison of the errors in the loss distribution

(a) $\delta_k^{Prob,method}$: Portfolio 64-1



(b) $\delta_k^{Prob,method}$: Portfolio 64-2

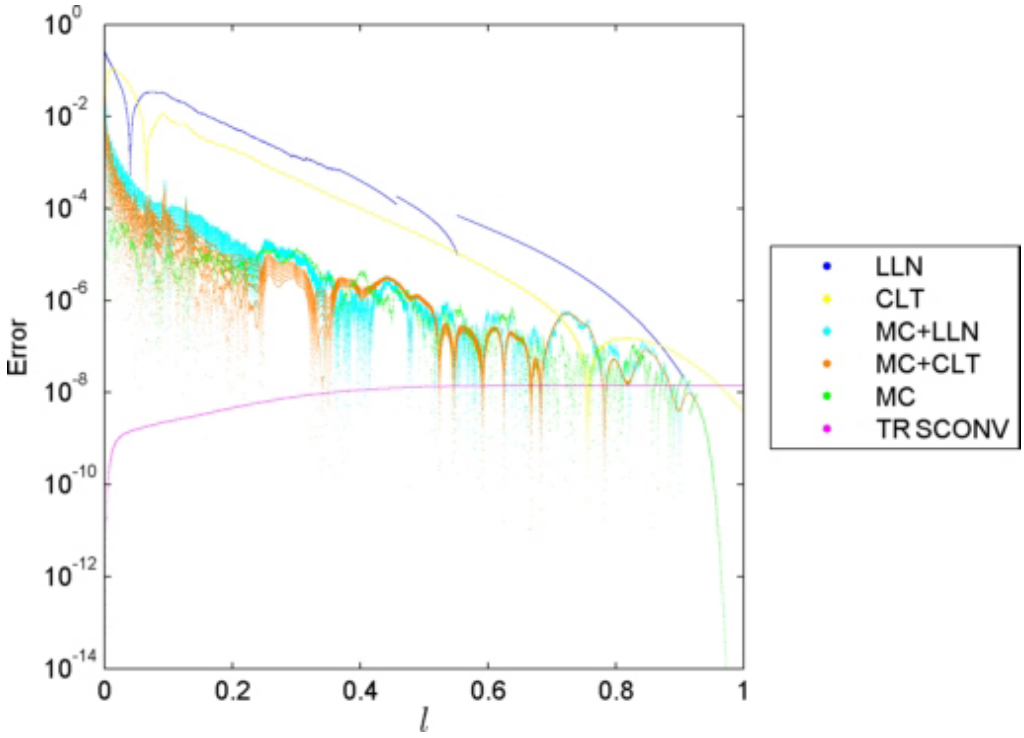
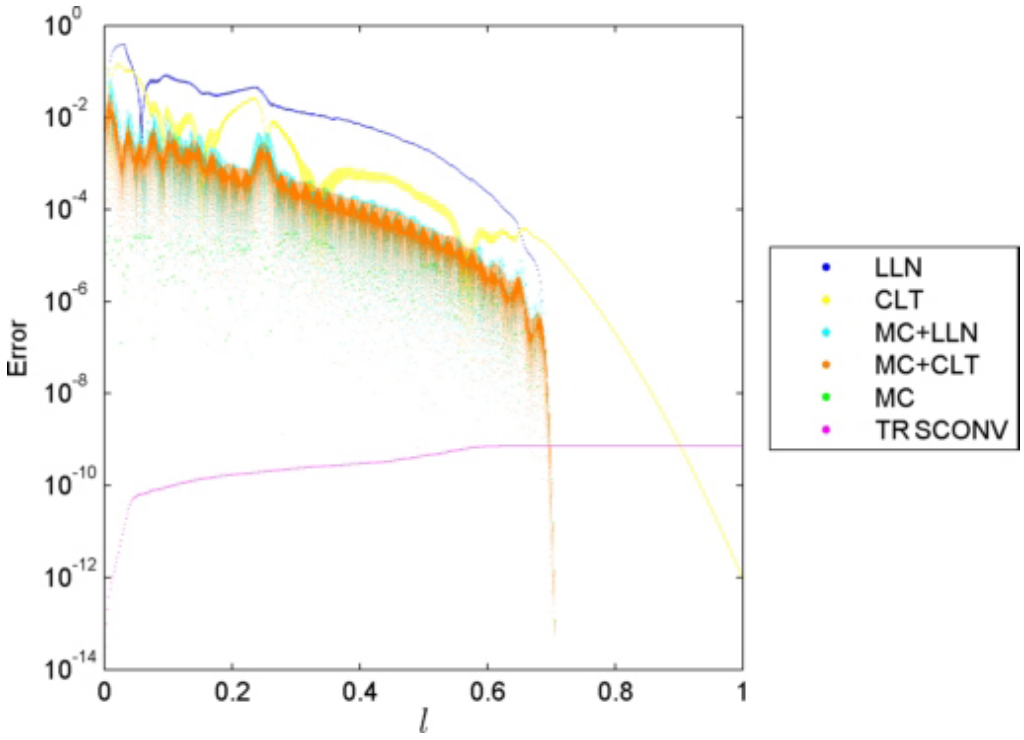


Figure 6.11: Comparison of the errors in the loss distribution (Cont'd)

(a) $\delta_k^{Prob,method}$: Portfolio 64-3



(b) $\delta_k^{Prob,method}$: Portfolio 64-4

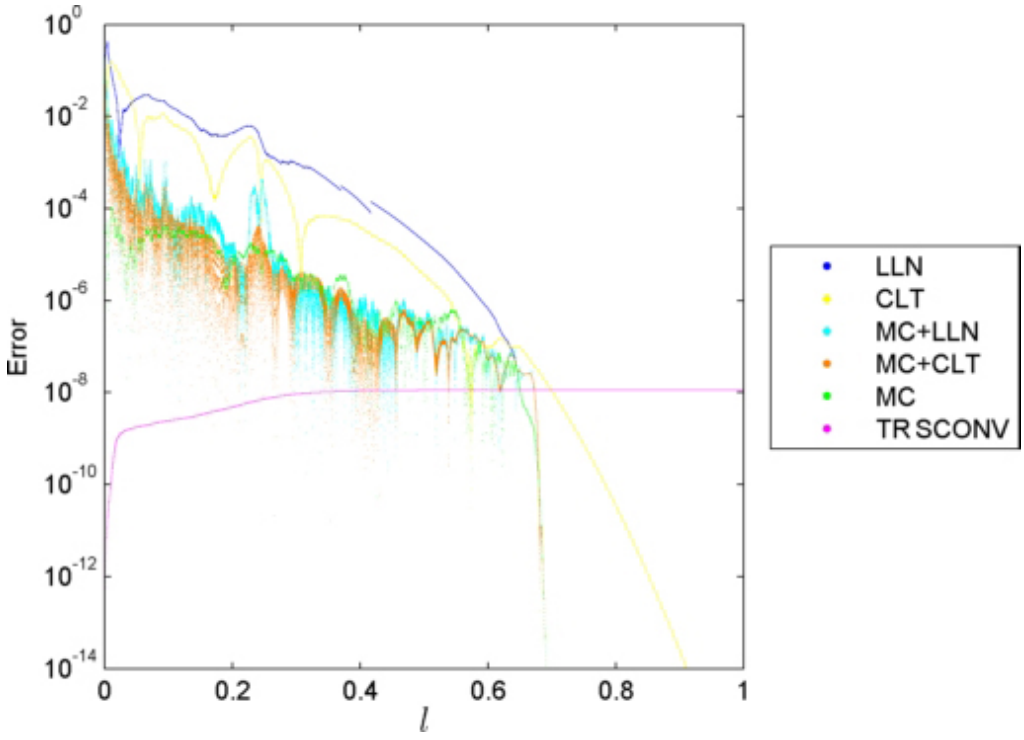
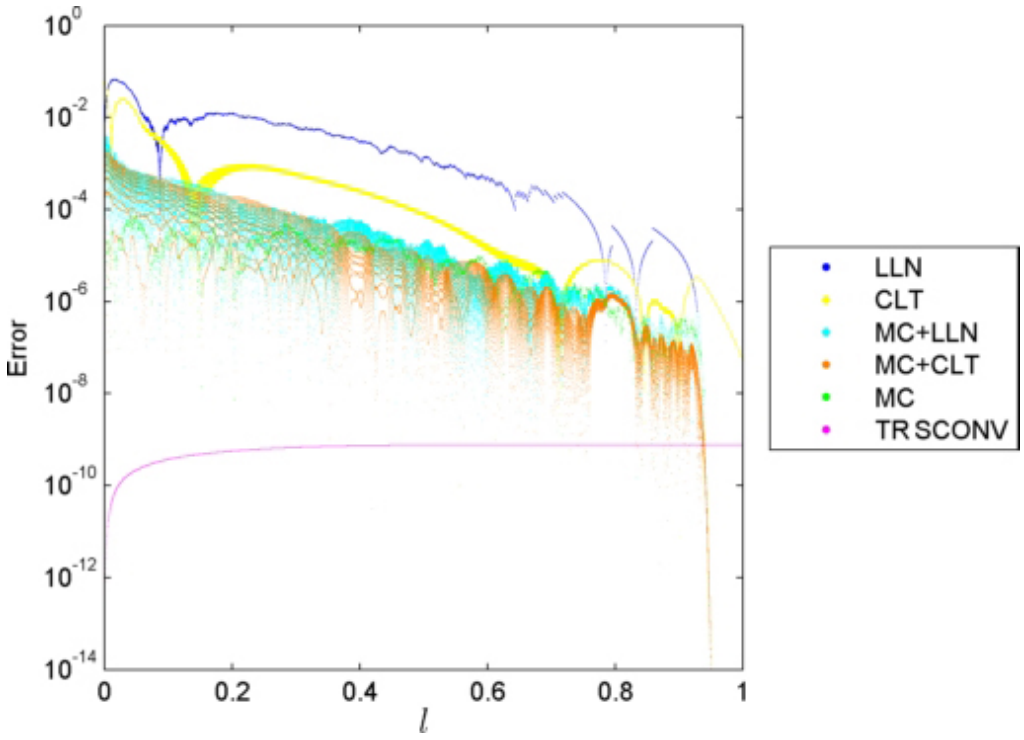


Figure 6.12: Comparison of the errors in the loss distribution

(a) $\delta_k^{Prob,method}$: Portfolio 2048-1



(b) $\delta_k^{Prob,method}$: Portfolio 2048-2

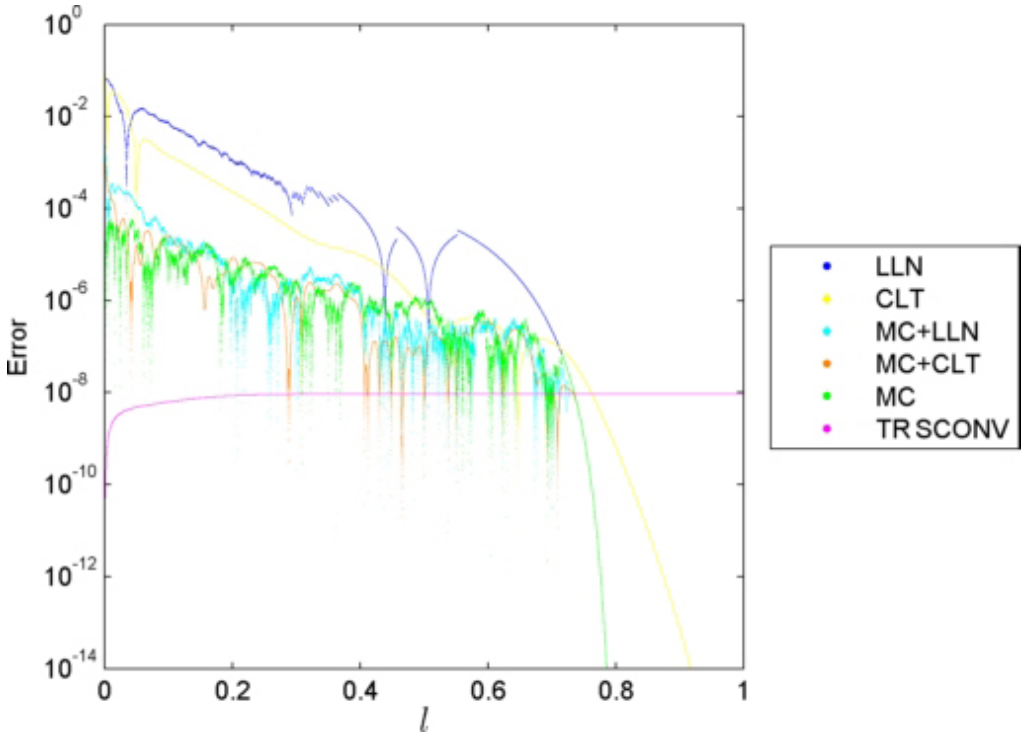
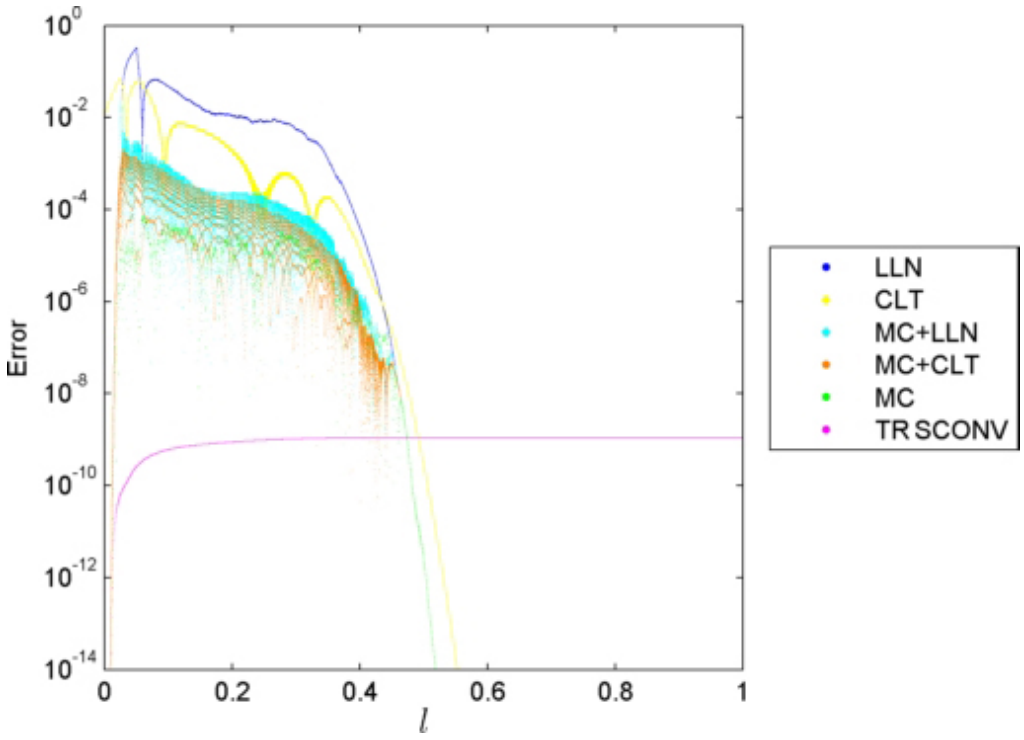
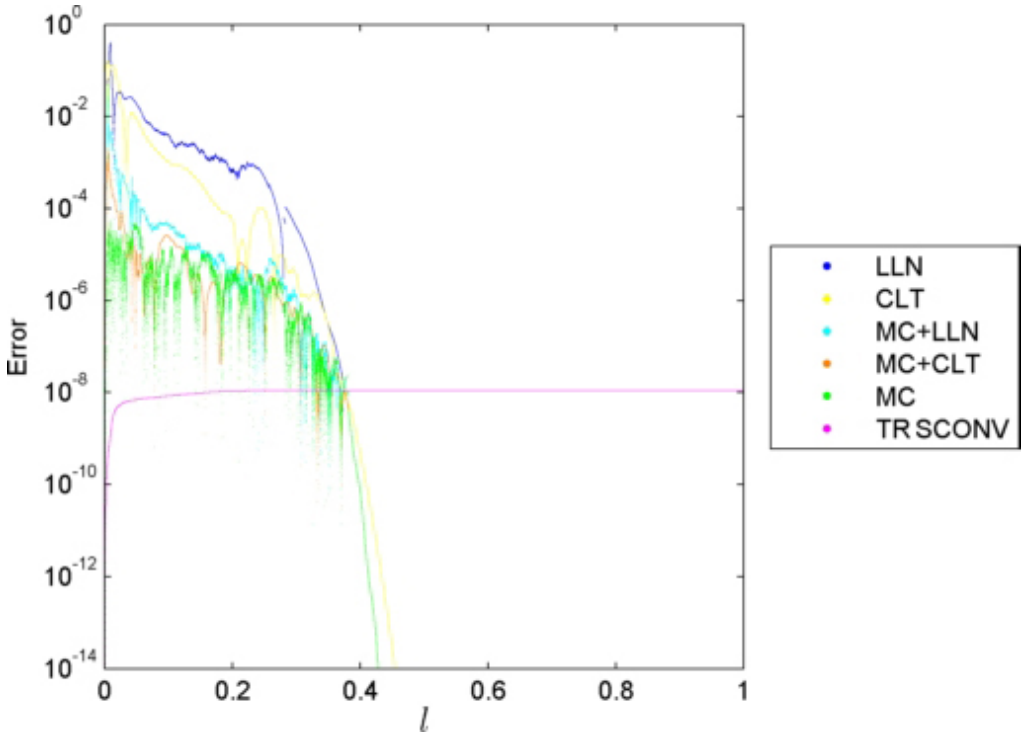


Figure 6.13: Comparison of the errors in the loss distribution (Cont'd)

(a) $\delta_k^{Prob,method}$: Portfolio 2048-3



(b) $\delta_k^{Prob,method}$: Portfolio 2048-4



6.4.1.2 Efficiency

Figure 6.14 compares the CPU time in seconds required by the tested methods, as well as the speedup of the the MC+LLN method relative to the pure MC, TR SCONV and SCONV methods, to compute the loss distribution. In the bottom part of Figures 6.14a and 6.13c, we plot the CPU time on a semi-log scale to better distinguish the efficiency of different methods. We make the following observations.

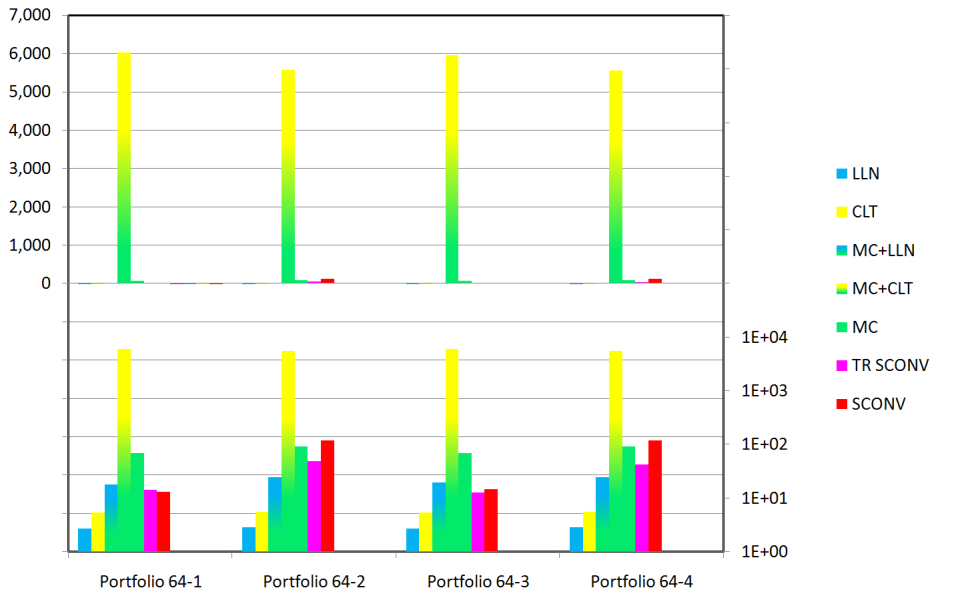
1. In all our tests, the MC+LLN method is faster than the MC method. Compared with the MC method, the MC+LLN method is more than 3 times faster for the coarser portfolios 64-1, 64-2, 64-3 and 64-4, and more than 50 times faster for the finer portfolios 2048-1, 2048-2, 2048-3, and 2048-4. There are two sources for this speed-up. First, fewer random numbers are generated by the hybrid approximations compared to the MC approximation. As mentioned earlier, the MC approximation generates $U \cdot (V \cdot N + S)$ random numbers, while the hybrid approximations require only $U \cdot (V \cdot N_{\mathsf{lmp}} + S)$ random numbers, where usually $N_{\mathsf{lmp}} \ll N$. (For our numerical results, $N = 64,\ 2048$, while $N_{\mathsf{lmp}} = 16,\ 64$.). Secondly, because $N_{\mathsf{lmp}} \ll N$, the computational work required for (6.1.1) is much less than that required for (3.3.7), which accelerates the computation significantly.

2. In all our tests, the MC+CLT method is significantly slower than the MC+LLN method, and it is even slower than the MC method, as in Subsection 6.3.1.2 predicted. Since the MC+CLT and MC+LLN methods are comparable in accuracy, we recommend using the MC+LLN method instead of the MC+CLT method to compute the loss distribution.

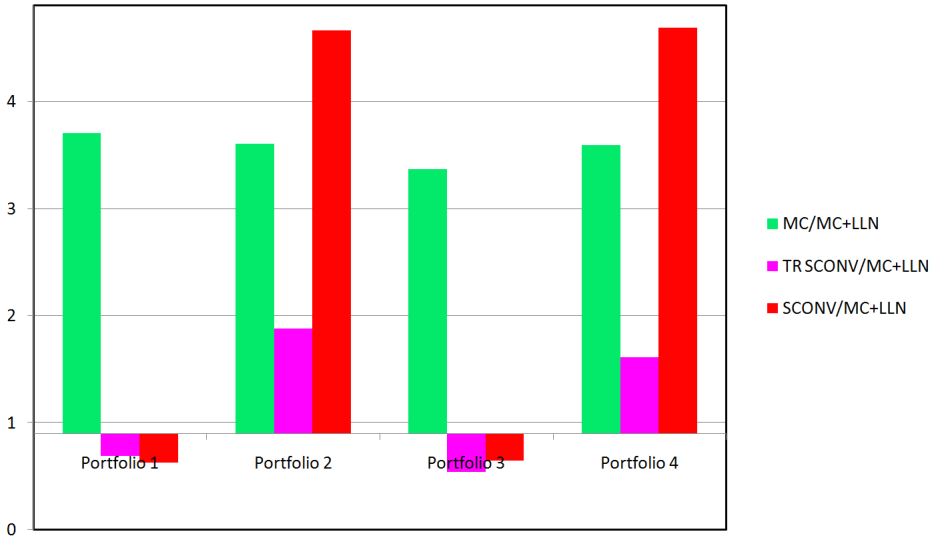
3. In most of our tests, the MC+LLN method is significantly faster than the SCONV method, especially for finer portfolios with a complex rating system. On the other hand, the speed of the MC+LLN method is comparable to that of the TR SCONV method for portfolios 64-1, 64-2, 64-3, 64-4, 2048-1 and 2048-3. Given that the TR SCONV method is more accurate than the MC+LLN method to generate most of the loss probabilities, we recommend using the TR SCONV method to compute the loss distribution for coarse portfolios with more complex rating systems. However, the MC+LLN method is more than 10 times faster than the TR SCONV method for Portfolio 2048-2 and 2048-4, which shows that, for a very fine portfolio with a complex rating system, the MC+LLN method provides a very good balance between accuracy and efficiency.

Figure 6.14: Comparison of efficiency in computing the loss distribution

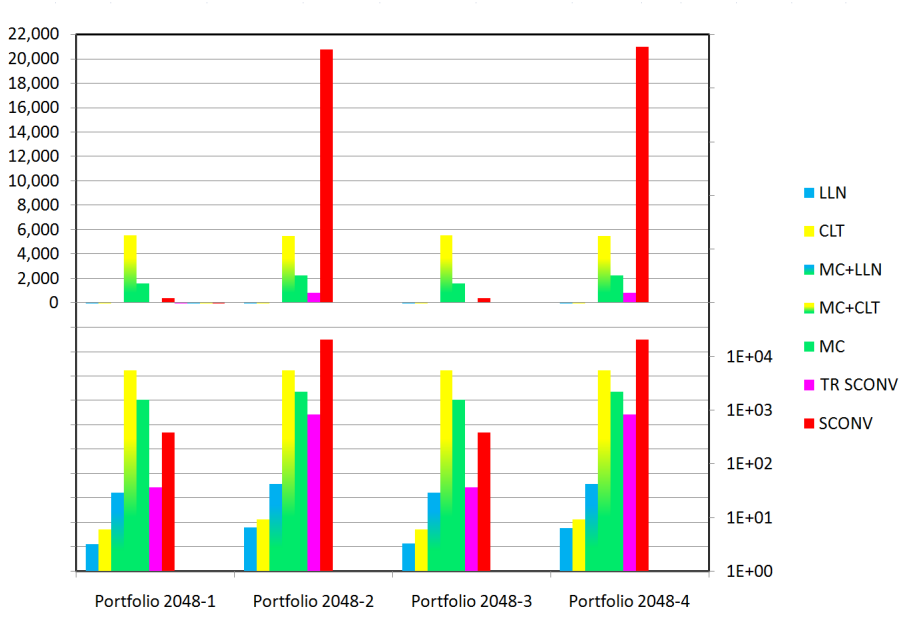
(a) CPU time: Portfolio 64-i



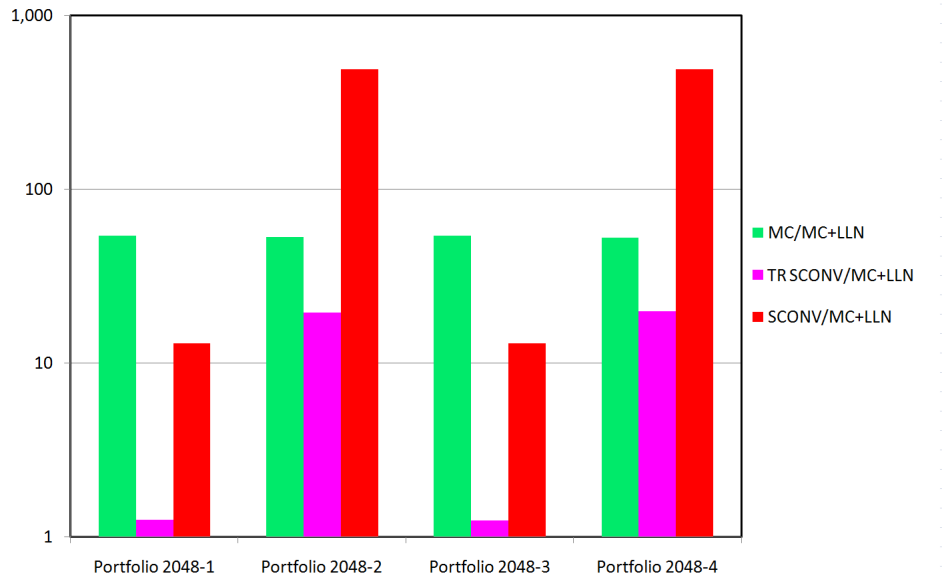
(b) Speedup: Portfolio 64-i



(c) CPU time: Portfolio 2048-i



(d) Speedup: Portfolio 2048-i



6.4.2 Comparison for Computing VaR

For Portfolios 2048-1 and 2048-2, we compute VaR at the confidence levels $\gamma = 95\%$, 99%, 99.9% and 99.98% by the following methods: LLN, CLT, Hybrid MC+LLN (Algorithm 6.7), Hybrid MC+CLT (Algorithm 6.7), MC, Hybrid TR SCONV+LLN (Algorithm 6.8), Hybrid TR SCONV+CLT (Algorithm 6.8), TR SCONV, Hybrid SCONV+LLN, Hybrid SCONV+CLT and SCONV method. As in Subsection

6.4.1, for the TR SCONV method, we set the threshold for errors in the cumulative loss probabilities to $Tol = 10^{-4}$, and set ϵ by (5.1.51). For the MC method, we set the sample size to $U = 2^{14}$, $V = 2^{12}$.

6.4.2.1 Accuracy

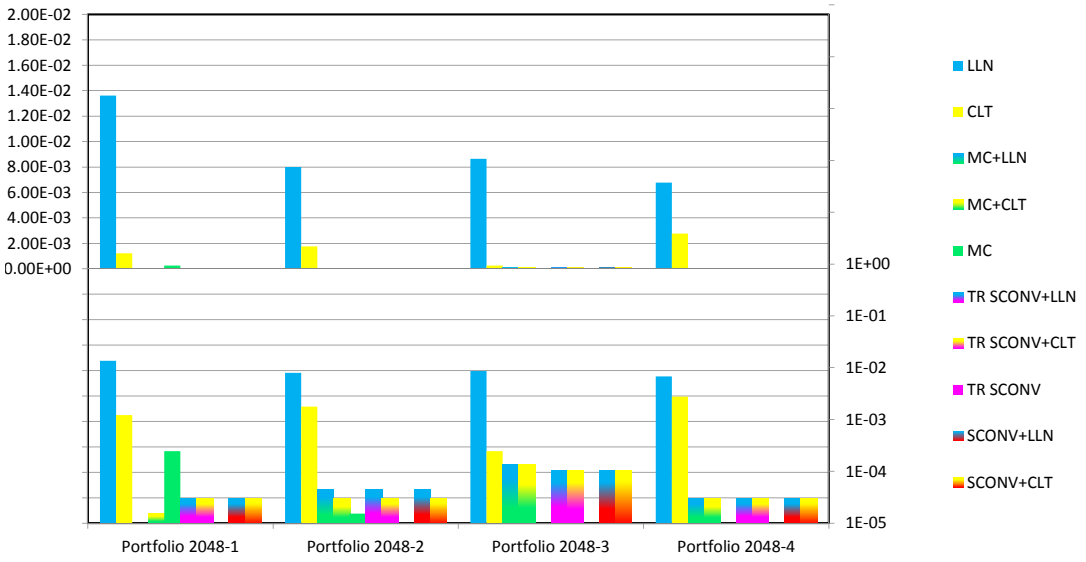
To assess the accuracy of these methods, we use VaRs calculated by the SCONV method as the benchmark, and compute the errors in VaRs computed by other methods by

$$\delta_{\gamma}^{VaR,method} = \left| \mathrm{VaR}_{\gamma}^{method} - \mathrm{VaR}_{\gamma}^{SCONV} \right|.$$

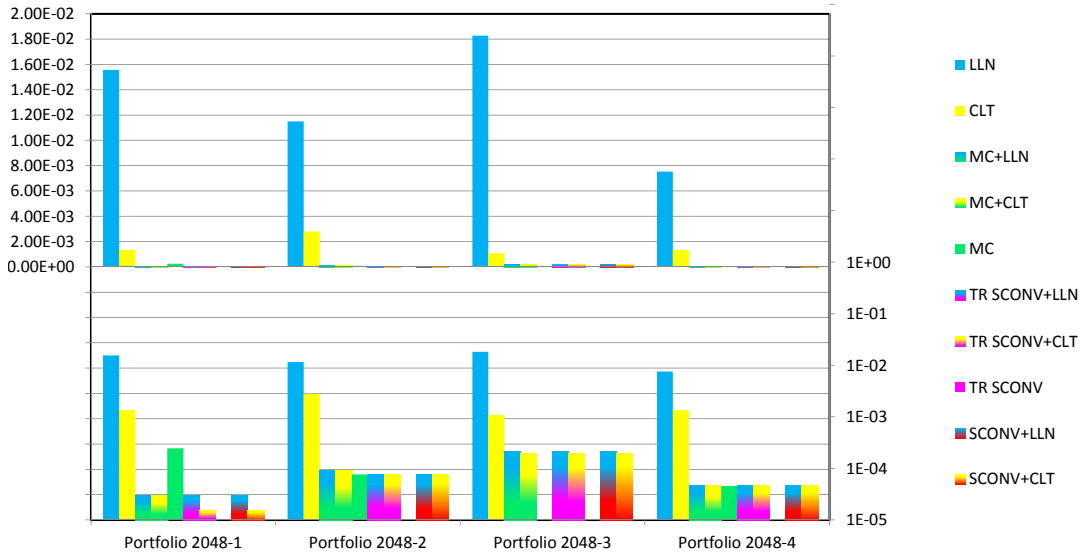
The testing results are presented in Figure 6.14, where we plot the errors on a semi-log scale in the bottom part of each subfigure to better distinguish errors of different methods. From the numerical results, it is clear that the errors in the computed VaR associated with all hybrid methods are comparable, and are significantly smaller than the asymptotic methods. This shows that the hybrid method is effective to reduce the asymptotic errors in VaR calculation for lumpy portfolios. On the other hand, there are still small but acceptable errors associated with the hybrid methods in all testing cases, while the errors associated with the exact methods or the MC method are zero in most testing cases. This shows that, to calculate VaR, the exact methods and the MC method are more accurate than the hybrid methods, which is as expected since the hybrid methods suffer from asymptotic errors associated with the homogeneous sub-portfolio.

Figure 6.14: Comparison of the errors in computing VaR

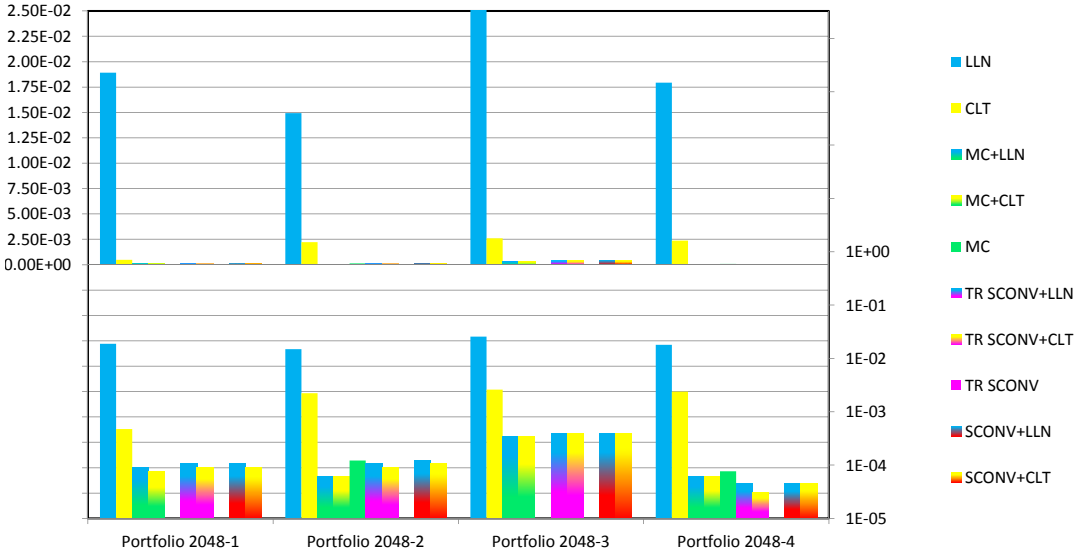
(a) Confidence level: 95%



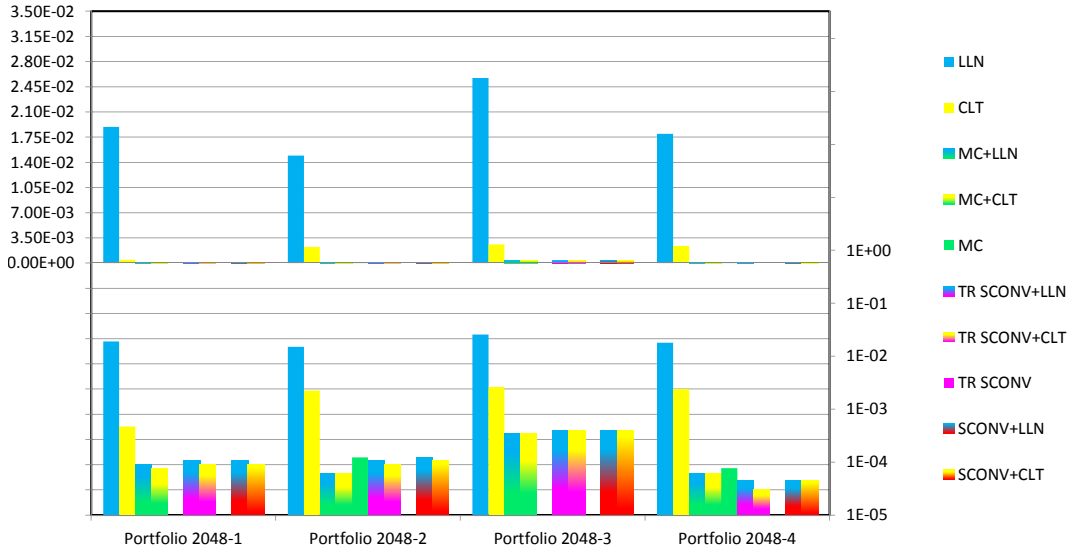
(b) Confidence level: 99%



(c) Confidence level: 99.9%



(d) Confidence level: 99.98%



6.4.2.2 Efficiency

Figure 6.14 compares the average CPU time in seconds, to compute VaR at different confidence levels, required by the tested methods, as well as the speedup of the hybrid methods relative to the pure MC, TR SCONV and SCONV methods. In the bottom part of Figure 6.14a, we plot the CPU time on a semi-log scale to better distinguish the efficiency of different methods. The following observations can be made.

- The MC+LLN method and the MC+CLT method are comparably efficient to compute VaR, and

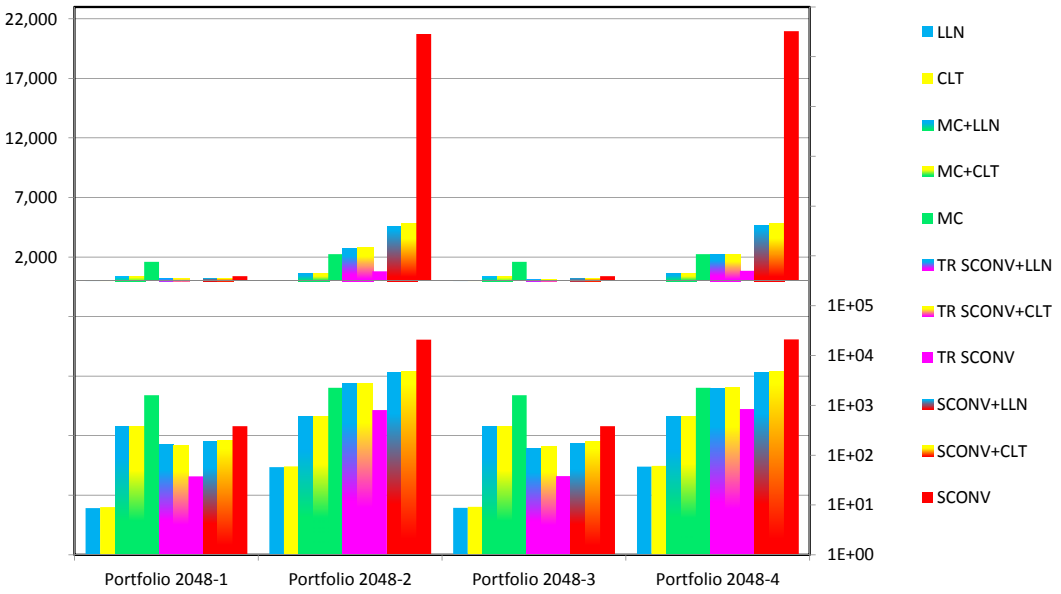
they are approximately four times faster than the MC method. However, notice that the CPU time required by the MC method to compute the loss distribution is comparable with that to compute VaR, and as we observed in Subsection 6.4.1.2, the MC+LLN method is around 50 times faster than the MC method to compute the loss distribution for the same testing portfolios. Moreover, given the loss distribution, it takes little additional computational time to compute VaR. Therefore, we recommend using Algorithm 6.1, rather than Algorithm 6.7, to compute VaR if the MC+LLN method is selected.

- The SCONV+LLN method and the SCONV+CLT methods are comparably efficient to compute VaR, and they are approximately 5-10 times faster than the SCONV method. However, the SCONV+LLN and SCONV+CLT methods are significantly slower than the MC+LLN method. Given these three methods have approximately the same accuracy, we recommend using the MC+LLN method over the SCONV+LLN and SCONV+CLT method to compute VaR.

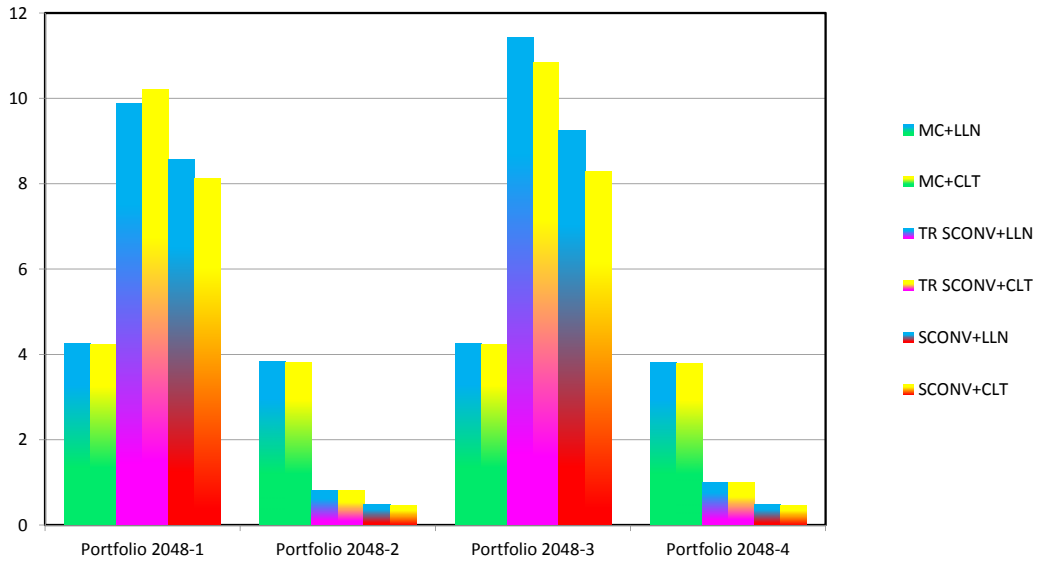
- Though the TR SCONV+LLN method and TR SCONV+CLT method are faster than the SCONV+LLN method and the SCONV+CLT method, the TR SCONV+LLN method and the TR SCONV+CLT method are slower than the TR SCONV method. This proves again that dropping extremely small probabilities applied in the TR SCONV method is a very effective approach to boost the performance of the SCONV method. Figure 6.13d shows that the MC+LLN method (Algorithm 6.7) is around 10 time slower than the TR SCONV method for portfolios with a simple rating system (portfolio 2048-1 and portfolio 2048-3), and approximated as fast as the TR SCONV method for portfolios with a more complex rating system (portfolio 2048-2 and portfolio 2048-4). However, as we discussed in Subsection 6.4.1.2, if Algorithm 6.1 is applied to calculate VaR, the MC+LLN method would be around 10 times faster than the TR SCONV method to compute VaR, which shows a good balance between accuracy and efficiency.

Figure 6.14: Comparison of efficiency in computing VaR

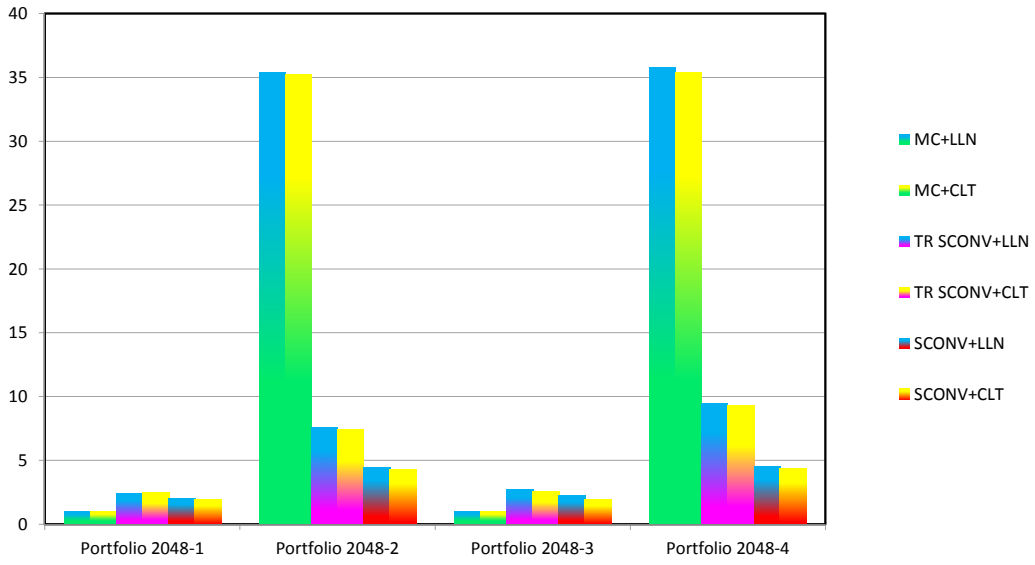
(a) CPU time



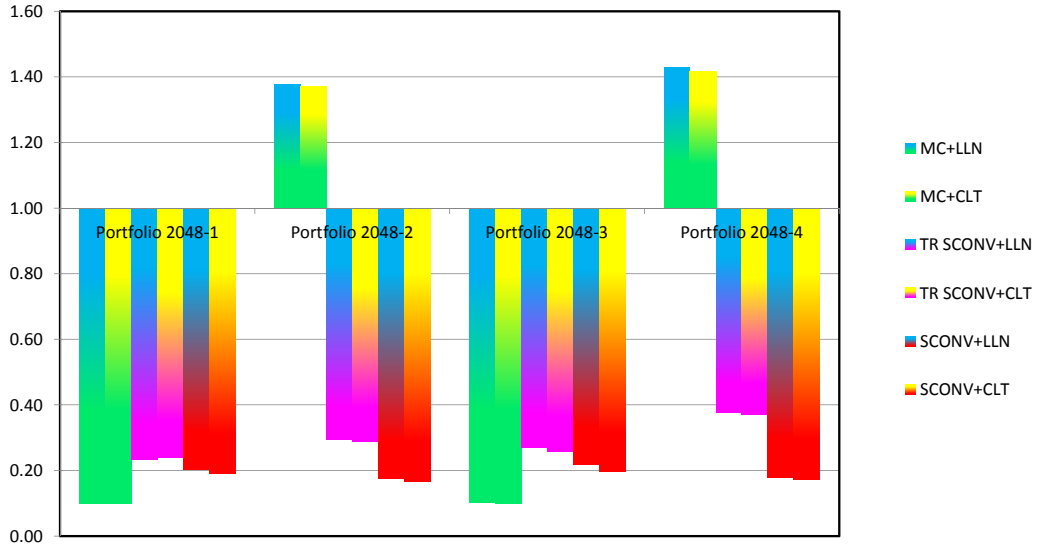
(b) Speedup over MC



(c) Speedup over SCONV



(d) Speedup over TR SCONV



Chapter 7

Conclusion and Future Work

In this thesis, we develop several new asymptotic and exact methods to compute conditional loss probabilities of credit portfolios in the CreditMetrics framework.

We propose an asymptotic approximation based on the CLT. We prove that both the conditional and unconditional loss distributions converge to our CLT limiting distribution under conditions similar to those used by Gordy to prove the convergence of the LLN approximation. To assess the accuracy of the CLT approximation, we use Berry-Esseen-type results to bound the error incurred in the CLT approximation.

We improve the efficiency of the exact methods by exploiting the sparsity that often occurs in the obligors' conditional losses. We develop a sparse convolution method which enjoys a speedup between $\Omega\left(\alpha^2\right)$ and $\Omega\left(\alpha^2 C^N\right)$ compared with the straightforward convolution method. To further accelerate the computation, we introduce a truncated sparse convolution method, which is subject to some additional truncation error to a user-specified level, but gives a significant speedup compared to the sparse convolution method. Moreover, we develop a sparse FFT method that enjoys a speedup of $\Omega\left(K/N\right)$ in the best case and $\Omega\left(K/\left(N\log_2 C\right)\right)$ in the worst case. We also construct a truncated sparse FFT method to further improve its efficiency, with an optimal exponential windowing approach used to balance aliasing errors and roundoff errors.

For lumpy portfolios, we introduce hybrid methods which combine an asymptotic approximation (CLT or LLN) and MC simulation or an exact method to achieve a good balance between accuracy and efficiency. Several algorithms are proposed to apply hybrid methods to compute the loss distribution and VaR.

We conduct several numerical experiments to test our methods. For asymptotic methods, we have

the following conclusions from our numerical results. The CLT approximation generates more accurate approximations to the loss distribution than does the LLN approximation in most cases, especially for coarse-grained heterogeneous portfolios. Also the CLT approximation consistently outperforms the LLN in the high probability tail of the loss distribution, which makes the CLT approximation more attractive as a method to compute downside risk measures, such as VaR and ES. Moreover, the CLT approximation is almost as efficient as the LLN approximation, both of which are significantly more efficient than the MC approximation.

For exact methods, numerical results in the best/worst cases agree with the theoretical speedup of the SCONV method and the SFFT method, and errors in the TR SCONV method are significantly smaller than the predefined error bound. In our tests with synthetic portfolios, we find that

- Both the SCONV method and the SFFT method produce excellent accuracy, and are faster than the FCONV method. The TR SCONV and (EW) TR SFFT methods greatly improve the efficiency of the SCONV and SFFT methods, respectively, at the cost of larger errors. However, it is easy to keep the errors in the TR SCONV method below a user specified threshold. The TR SFFT suffers significant aliasing errors, but, by using exponential windowing, the EW TR SFFT method can reduce the aliasing errors to approximately the same level as the errors of the TR SCONV method at negligible computational cost.

- The sparsity of the conditional probability vectors (or the number of rating classes) has an impact on sparse methods and truncated sparse methods. In general, as C increases, the efficiency decreases, but sparse FFT methods are less sensitive to the sparsity of portfolios than sparse convolution methods. However, one exception is the EW TR SFFT method, which might run faster as C increases due to the increase in the length of the truncated conditional probability vectors.

- The heterogeneity of portfolios is another major factor which affects the efficiency of sparse convolution methods (SCONV and TR SCONV). When C is small, sparse convolution methods run faster as the portfolio becomes more homogeneous. However, when C is large, the impact of the heterogeneity of portfolios on the efficiency of sparse convolution methods is very limited. The impact of the heterogeneity of portfolios is not significant on sparse FFT methods (SFFT and (EW) TR SCONV).

- Comparing the sparse convolution methods (SCONV and TR SCONV) with the sparse FFT methods (SFFT and (EW) TR SFFT), the sparse convolution methods are much more efficient than the sparse FFT methods when C is small. However, sparse FFT methods are much less sensitive

to sparsity and heterogeneity of portfolios than sparse convolution methods, and as the portfolio becomes less sparse and less homogeneous, the advantage in efficiency of sparse convolution methods is gradually reduced and even reversed. Moreover, the length of the truncated vector of probabilities has significant impact on the efficiency of the (EW) TR SFFT method. As the length of the truncated vector of probabilities increases, the efficiency of the (EW) TR SFFT method decreases.

- Compared to the MC method with a reasonably large sample size to achieve acceptable accuracy, the TR SCONV method is more efficient and more accurate. In addition, compared to the random errors incurred in the MC method, the errors incurred in the TR SCONV method are deterministic and can be controlled by the user specified threshold.

For lumpy portfolios, numerical results show that both of the asymptotic approximations (CLT and LLN) are inaccurate. For the calculation of the loss distribution and VaR, we conduct numerical experiments to compare the efficiency and accuracy of different hybrid methods to those of exact methods and asymptotic methods. Hybrid approximations produce much more accurate loss distributions than asymptotic approximations for lumpy portfolios, especially in the high probability tail of the loss distributions. The MC+LLN method is $3-50$ times faster than the MC approximation, as much as 10 times faster than the TR SCONV method, and as much as 700 times faster than the SCONV method, which shows that the MC+LLN method provides a good balance between efficiency and accuracy.

In the future, we hope to perform the following improvements on the methods discussed in this thesis:

- Improve the efficiency of exact methods by allowing discretization errors.
- Further improve further simulation-based methods by developing importance sampling (IS) techniques.
- Develop efficient methods for multi-period models.

For the exact methods, this thesis assumes that a perfect discretization grid is used. That is, each possible loss is located exactly on our discretization grid. Hence, there is no discretization error involved in our exact methods. It is worth mentioning that, subject to user specified discretization error tolerance, we could round some losses to a coarser grid to reduce the total number of points on the grid and consequently to accelerate the exact methods (especially the SFFT and TR SFFT methods). We could combine this coarsening approach with other techniques described in this thesis to achieve a more efficient, but less accurate, approximation to the conditional portfolio loss.

Glasserman et al. [17, 18, 19] developed IS schemes for both the inner-level and the outer-level MC simulations. For the inner-level simulation, their methods are quite effective. However, we believe that it may be possible to improve their outer-level simulation. In their work, the IS scheme shifts the mean of the normal distribution of systematic risk factors to approximate the IS zero-variance distribution. The shifted mean is chosen by matching the mode of the shifted normal distribution and the mode of an upper bound on the IS zero-variance distribution. This IS scheme suffers from the following problems:

1. The shifted normal distribution may not be close to the IS zero-variance distribution. Indeed, we have found that the shape of the IS zero-variance distribution can be quite different from a normal distribution. For example, the IS zero-variance distribution has more than one mode in many cases.

2. The IS zero-variance distribution and its upper bound may have very different modes. Therefore, the shifted mean may be very far away from the mode of the IS zero-variance distribution.

3. Their IS scheme does not match the variance. Instead, it takes the variance to be $\mathbf{I}$. However, The variance of the IS zero-variance distribution may be very different from $\mathbf{I}$.

Currently, we are working on an improvement over Glasserman's method. Our plan is to use a mixture of Gaussians to approximate the IS zero-variance distribution to solve the first and last problems outlined above. For the second problem, from the discussion in Section 4.2, the conditional portfolio loss probabilities can be approximated by the CLT, hence we can use the CLT approximation to find a good estimate of the IS zero-variance distribution. By minimizing the difference of the density function of the CLT estimate of the IS zero-variance distribution and the density of the Gaussian mixture, we may be able to determine the parameters in the Gaussian mixture. Once this is done, we can use the measure corresponding to the Gaussian mixture to compute the unconditional portfolio loss distribution. Since our CLT method approximates the conditional portfolio loss probabilities quite well, at least for fine-grained portfolios, the CLT estimate of the IS zero-variance distribution should be close to the IS zero-variance distribution. Therefore, the variance reduction in our IS scheme should be good.

All methods introduced in this thesis are based on a single-period Merton's model. That is, we assume all credit event can only occur at the end of the considered time-horizon. In recent years, multi-period models have become more attractive to both regulators and financial institutions. In a multi-period model, the considered time-horizon is divided into several periods, and it is assumed that credit events can happen at the end of each period. If the assumption of a constant level of risk is applied (i.e., positions with changed credit quality at the end of a period are replaced with positions having similar

credit quality to what they had at the start of the period), then the portfolio loss distribution in each period can be assumed to be the same. Moreover, portfolio losses in different periods are independent, hence the portfolio loss probabilities in the considered time-horizon can be computed by a multi-fold convolution of portfolio loss probabilities in each period. Portfolio loss probabilities in each period can be computed using any method introduced in this thesis. In the more complicated cases, for which the assumption of constant level of risk is not applied, the portfolio loss probabilities in the considered time-horizon can be computed by either a grouping method, for which the computation is based on grouping all possible credit rating sequences by their corresponding losses, or a multi-period CLT approximation. We are working on the methods for multi-period models in both cases.

We had originally planned to develop the improved IS scheme, as well as methods for multi-period models in this thesis. However, we believe that this thesis already contains more than enough results. Therefore, we plan to develop the improved IS scheme and the methods for multi-period models as two separate projects.

Appendix A

Proof of Theorem 4.1

We use Linderberg's version of CLT to prove Theorem 4.1. The proof is based on the following two lemmas.

Lemma A.1. *If $\mathcal{X}_1, \ldots, \mathcal{X}_N$ are independent with finite second moments, and*

$$\forall\, \epsilon > 0,\ \frac{1}{\left(\sigma^{(N)}\right)^2} \sum_{n=0}^{N-1} \int_{\left\{|x-\mu_n| \geq \epsilon \sigma^{(N)}\right\}} (x-\mu_n)^2\, dF_n(x) \to 0 \text{ as } N \to \infty, \tag{A.0.1}$$

then

$$\frac{\sum_{n=0}^{N-1} \mathcal{X}_n - \mu^{(N)}}{\sigma^{(N)}} \xrightarrow{d} \mathcal{N}(0,1),\ \textit{as}\ N \to \infty,$$

where $\left(\sigma^{(N)}\right)^2 = \sum_{n=0}^{N-1} \mathbb{V}\left[\mathcal{X}_n\right]$, $\mu_n = \mathbb{E}\left[\mathcal{X}_n\right]$, $\mu^{(N)} = \sum_{n=0}^{N-1} \mu_n$, $F_n(x)$ *is the CDF of* $\mathcal{X}_n$.

Proof. The proof of this lemma can be found in [56, pp. 328-337]. □

Lemma A.2. *Under the model assumptions and conditions of Theorem 4.1, the conditional loss rate of each obligor, $L_n(\boldsymbol{z}, \boldsymbol{\mathcal{E}})$, is not a degenerate random variable. That is, $\exists\, \sigma_{min}(\boldsymbol{z})$ such that $\mathbb{V}\left[L_n(\boldsymbol{z}, \boldsymbol{\mathcal{E}})\right] \geq \sigma^2_{min}(\boldsymbol{z}) > 0$.*

Proof. To begin, notice that, since $\boldsymbol{\mathcal{Z}} = \boldsymbol{z} < \infty$, there exists $M > 0$ such that

$$\boldsymbol{z} \in \mathbb{D}^S \triangleq [-M, M] \times \cdots \times [-M, M]. \tag{A.0.2}$$

Let $p_n^c(\boldsymbol{z})$ be the probability conditional on $\boldsymbol{\mathcal{Z}} = \boldsymbol{z}$ that obligor n is in credit state c at time $t = t_1$ given

that it is in credit state $c(n)$ at time $t = t_0$. It follows from (3.1.7) and (3.3.2) that

$$\begin{aligned} p_n^c(\boldsymbol{z}) &= \mathbb{P}\left\{ \frac{H_{c(n)}^{c-1} - \boldsymbol{\beta}_n^T \boldsymbol{z}}{\sqrt{1-\boldsymbol{\beta}_n^T \boldsymbol{\beta}_n}} \leq \mathcal{E}_n < \frac{H_{c(n)}^{c} - \boldsymbol{\beta}_n^T \boldsymbol{z}}{\sqrt{1-\boldsymbol{\beta}_n^T \boldsymbol{\beta}_n}} \right\} \\ &= \Phi\left(\frac{H_{c(n)}^{c} - \boldsymbol{\beta}_n^T \boldsymbol{z}}{\sqrt{1-\boldsymbol{\beta}_n^T \boldsymbol{\beta}_n}} \right) - \Phi\left(\frac{H_{c(n)}^{c-1} - \boldsymbol{\beta}_n^T \boldsymbol{z}}{\sqrt{1-\boldsymbol{\beta}_n^T \boldsymbol{\beta}_n}} \right). \end{aligned} \tag{A.0.3}$$

Thus, the variance of $L_n(\boldsymbol{z}, \boldsymbol{\mathcal{E}})$ is

$$\begin{aligned} \mathbb{V}\left[L_n(\boldsymbol{z}, \boldsymbol{\mathcal{E}})\right] &= \sum_{c=0}^{C-1} \left(\mathrm{LGC}_n^c\right)^2 p_n^c(\boldsymbol{z}) - \left(\sum_{c=0}^{C-1} \mathrm{LGC}_n^c p_n^c(\boldsymbol{z}) \right)^2 \\ &= \frac{1}{2} \left(\sum_{a=0}^{C-1} \left(\mathrm{LGC}_n^a\right)^2 p_n^a(\boldsymbol{z}) \sum_{b=0}^{C-1} p_n^b(\boldsymbol{z}) + \sum_{b=0}^{C-1} \left(\mathrm{LGC}_n^b\right)^2 p_n^b(\boldsymbol{z}) \sum_{a=0}^{C-1} p_n^a(\boldsymbol{z}) \right. \\ &\quad \left. - 2 \sum_{a=0}^{C-1} \sum_{b=0}^{C-1} \mathrm{LGC}_n^a \mathrm{LGC}_n^b p_n^a(\boldsymbol{z}) p_n^b(\boldsymbol{z}) \right) \\ &= \frac{1}{2} \sum_{a=0}^{C-1} \sum_{b=0}^{C-1} \left(\mathrm{LGC}_n^a - \mathrm{LGC}_n^b\right)^2 p_n^a(\boldsymbol{z}) p_n^b(\boldsymbol{z}) \\ &= \sum_{a>b} \left(\mathrm{LGC}_n^a - \mathrm{LGC}_n^b\right)^2 p_n^a(\boldsymbol{z}) p_n^b(\boldsymbol{z}), \end{aligned} \tag{A.0.4}$$

where we use $\sum_{c=0}^{C-1} p_n^c(\boldsymbol{z}) = 1$ in the second equation.

Next we prove that, under our model assumptions in Chapter 4 and (A.0.2), for $p_n^c(\boldsymbol{z})$ defined by (A.0.3), the following statement is true:

$$\forall n,\ \exists i, j \in \{0, \ldots, C-1\},\ i \neq j, \text{ such that } p_n^i(\boldsymbol{z}) > 0 \text{ and } p_n^j(\boldsymbol{z}) > 0. \tag{A.0.5}$$

Suppose (A.0.5) is not true, then there must be a credit state c^* such that $p_n^{c^*}(\boldsymbol{z}) = 1$ and $p_n^c(\boldsymbol{z}) = 0$ for all $c \neq c^*$. From (A.0.3), we see that this is equivalent to

$$\begin{cases} \frac{H_{c(n)}^{c^*} - \boldsymbol{\beta}_n^T \boldsymbol{z}}{\sqrt{1-\boldsymbol{\beta}_n^T \boldsymbol{\beta}_n}} = +\infty, \\ \frac{H_{c(n)}^{c^*-1} - \boldsymbol{\beta}_n^T \boldsymbol{z}}{\sqrt{1-\boldsymbol{\beta}_n^T \boldsymbol{\beta}_n}} = -\infty. \end{cases} \tag{A.0.6}$$

From assumption 6 in Chapter 4

$$\boldsymbol{\beta}_n^T \boldsymbol{\beta}_n \leq \beta_{max} < 1,$$

and from (A.0.2), we see that (A.0.6) is equivalent to

$$\begin{cases} H_{c(n)}^{c^*} = +\infty, \\ H_{c(n)}^{c^*-1} = -\infty. \end{cases} \tag{A.0.7}$$

Since

$$H_{c(n)}^{c} = \Phi^{-1}\left(\sum_{\gamma \le c} P_{c(n)}^{\gamma}\right), \tag{A.0.8}$$

(A.0.7) is equivalent to

$$\begin{cases} \sum_{\gamma \le c^*} P_{c(n)}^{\gamma} = 1, \\ \sum_{\gamma \le c^*-1} P_{c(n)}^{\gamma} = 0, \end{cases}$$

which implies $P_{c(n)}^{c^*} = 1$ and $P_{c(n)}^{c} = 0$ for all $c \neq c^*$. However, this contradicts our model assumptions 4 and 5 in Chapter 4 that ensure that for all $n \in \{1, \ldots, N\}$, all $c(n) \in \{1, \ldots, C-1\}$, and all $c \in \{0, \ldots, C-1\}$, $0 \le P_{c(n)}^{c} \le P_{max} < 1$. Therefore, (A.0.5) is true.

Next, we show that the conditional loss probabilities in (A.0.5) are bounded below by a positive constant that is independent of n. That is, for i, j in (A.0.5),

$$\exists \eta > 0 \text{ such that } p_n^i(\boldsymbol{z}) \ge \eta \text{ and } p_n^j(\boldsymbol{z}) \ge \eta. \tag{A.0.9}$$

If $H_{c(n)}^{i-1} = -\infty$, then (A.0.2), (A.0.3) and model assumption 6 in Chapter 4 give that

$$\begin{aligned} p_n^i(\boldsymbol{z}) &= \Phi\left(\frac{H_{c(n)}^{i} - \boldsymbol{\beta}_n^T \boldsymbol{z}}{\sqrt{1 - \boldsymbol{\beta}_n^T \boldsymbol{\beta}_n}}\right) \\ &= \Phi\left(\frac{\Phi^{-1}\left(P_{c(n)}^{i}\right) - \boldsymbol{\beta}_n^T \boldsymbol{z}}{\sqrt{1 - \boldsymbol{\beta}_n^T \boldsymbol{\beta}_n}}\right). \end{aligned} \tag{A.0.10}$$

From (A.0.5) we see that for such i,

$$P_{c(n)}^{i} \ge P_{min}^{+} > 0, \tag{A.0.11}$$

where

$$P_{min}^{+} = \min_{c_0, c_1 \in \{0, \ldots, C-1\}} \left\{ P_{c_0}^{c_1} : \; P_{c_0}^{c_1} > 0 \right\}.$$

Otherwise, we would have $P_{c(n)}^{i} = 0$, implying $p_n^i(\boldsymbol{z}) = 0$, which contradicts (A.0.5). Thus, (A.0.2),

(A.0.10), (A.0.11) and model assumption 6 in Chapter 4 ensures that

$$p_n^i(\boldsymbol{z}) \geq \eta_1 > 0, \tag{A.0.12}$$

where

$$\eta_1 = \Phi\left(\frac{\Phi^{-1}\left(P_{min}^+\right) - SM}{\alpha}\right),$$

$$\alpha = \begin{cases} 1, & \text{if } \Phi^{-1}\left(P_{min}^+\right) \geq SM, \\ \sqrt{1-\beta_{max}} & \text{if } \Phi^{-1}\left(P_{min}^+\right) < SM, \end{cases}$$

S is the number of systematic factors.

On the other hand, if $H_{c(n)}^{i-1} > -\infty$, then (A.0.3) and the mean value theorem gives that

$$p_n^i(\boldsymbol{z}) = \phi\left(\frac{x - \boldsymbol{\beta}_n^T\boldsymbol{z}}{\sqrt{1-\boldsymbol{\beta}_n^T\boldsymbol{\beta}_n}}\right)\frac{H_{c(n)}^i - H_{c(n)}^{i-1}}{\sqrt{1-\boldsymbol{\beta}_n^T\boldsymbol{\beta}_n}}$$

for some $x \in \left[H_{c(n)}^{i-1}, H_{c(n)}^i\right]$, where $\phi(x)$ is the PDF of the standard normal distribution. Since $\phi(x)$ is always positive and decreases as $|x|$ increases, it follows from (A.0.2) and model assumption 6 in Chapter 4 that

$$\phi\left(\frac{x - \boldsymbol{\beta}_n^T\boldsymbol{z}}{\sqrt{1-\boldsymbol{\beta}_n^T\boldsymbol{\beta}_n}}\right) \geq \phi\left(\theta\right) > 0,$$

where

$$\theta = \begin{cases} \frac{H_{c(n)}^i + SM}{\sqrt{1-\beta_{max}}}, \text{ if } \phi\left(H_{c(n)}^i\right) \leq \phi\left(H_{c(n)}^{i-1}\right), \\ \frac{H_{c(n)}^{i-1} - SM}{\sqrt{1-\beta_{max}}}, \text{ if } \phi\left(H_{c(n)}^i\right) > \phi\left(H_{c(n)}^{i-1}\right). \end{cases}$$

Consequently,

$$p_n^i(\boldsymbol{z}) \geq \phi\left(\theta\right)\left(H_{c(n)}^i - H_{c(n)}^{i-1}\right). \tag{A.0.13}$$

Notice that, by (A.0.8) and the mean value theorem, we have

$$\begin{aligned} H_{c(n)}^i - H_{c(n)}^{i-1} &= \Phi^{-1}\left(\sum_{\gamma \leq i} P_{c(n)}^\gamma\right) - \Phi^{-1}\left(\sum_{\gamma \leq i-1} P_{c(n)}^\gamma\right) \\ &= \frac{d}{dx}\Phi^{-1}\left(\sum_{\gamma \leq i-1} P_{c(n)}^\gamma + \kappa\right) P_{c(n)}^i, \end{aligned} \tag{A.0.14}$$

where $\kappa \in \left[0, P^i_{c(n)}\right]$. Notice that, since

$$\frac{d}{dx}\Phi\left(\Phi^{-1}(x)\right) = \frac{d}{dx}x = 1,$$

and

$$\frac{d}{dx}\Phi\left(\Phi^{-1}(x)\right) = \phi\left(\Phi^{-1}(x)\right)\frac{d}{dx}\Phi^{-1}(x),$$

thus we have

$$\frac{d}{dx}\Phi^{-1}(x) = \frac{1}{\phi\left(\Phi^{-1}(x)\right)} \geq \frac{1}{\phi\left(0\right)} = \sqrt{2\pi}.$$

Therefore,

$$\frac{d}{dx}\Phi^{-1}\left(\sum_{\gamma \leq i-1} P^{\gamma}_{c(n)} + \kappa\right) \geq \sqrt{2\pi}. \tag{A.0.15}$$

For $P^i_{c(n)}$, if $P^i_{c(n)} = 0$, then $H^i_{c(n)} = H^{i-1}_{c(n)}$ implying $p^i_n(\boldsymbol{z}) = 0$, which contradicts (A.0.5). Therefore, (A.0.11) also holds for the case $H^{i-1}_{c(n)} > -\infty$. By (A.0.11), (A.0.14) and (A.0.15), we have

$$H^i_{c(n)} - H^{i-1}_{c(n)} \geq \sqrt{2\pi}P^+_{min} > 0. \tag{A.0.16}$$

Consequently, (A.0.13) and (A.0.16) give

$$p^i_n(\boldsymbol{z}) \geq \eta_2 > 0, \tag{A.0.17}$$

where $\eta_2 \triangleq \phi\left(\theta\right)\sqrt{2\pi}P^+_{min} > 0$.

Let $\eta = \min\left(\eta_1, \eta_2\right)$. Then (A.0.12) and (A.0.17) give that, there exists $\eta > 0$, such that $p^i_n(\boldsymbol{z}) \geq \eta$. Similarly, we can show $p^j_n(\boldsymbol{z}) \geq \eta > 0$. Therefore, (A.0.9) is true. As a consequence of (A.0.4), (A.0.9), and model assumption 3 in Chapter 4 which ensures that $\left|\mathrm{LGC}^i_n - \mathrm{LGC}^j_n\right| \geq \xi > 0$ if $i \neq j$ for all $n \in \{0, \ldots . N-1\}$, we have

$$\begin{aligned}
\mathbb{V}\left[L_n(\boldsymbol{z}, \boldsymbol{\mathcal{E}})\right] &\geq \left(\mathrm{LGC}^i_n - \mathrm{LGC}^j_n\right)^2 p^i_n(\boldsymbol{z}) p^j_n(\boldsymbol{z}) \\
&\geq \xi^2\eta^2 \\
&> 0.
\end{aligned}$$

Therefore, we conclude that there exists $\sigma^2_{min}(\boldsymbol{z}) = \xi^2\eta^2$ such that $\mathbb{V}\left[L_n(\boldsymbol{z}, \boldsymbol{\mathcal{E}})\right] \geq \sigma^2_{min}(\boldsymbol{z}) > 0$. □

Next we use the two lemmas above to prove Theorem 4.1. First, note that

$$\begin{aligned}\mathbb{E}\left[\widetilde{L}_n^2(\boldsymbol{z},\boldsymbol{\mathcal{E}})\right] &= \omega_n^2\left(\sum_{c=0}^{C-1}\left(\mathrm{LGC}_n^c\right)^2 p_n^c\left(\boldsymbol{z}\right)\right)\\ &= \left(\frac{\mathrm{EAD}_n}{\sum_{n=0}^{N-1}\mathrm{EAD}_n}\right)^2\left(\sum_{c=0}^{C-1}\left(\mathrm{LGC}_n^c\right)^2 p_n^c\left(\boldsymbol{z}\right)\right). \qquad \text{(A.0.18)}\end{aligned}$$

From

$$p_n^c\left(\boldsymbol{z}\right) = \Phi\left(\frac{H_{c(n)}^c - \boldsymbol{\beta}_n^T\boldsymbol{z}}{\sqrt{1-\boldsymbol{\beta}_n^T\boldsymbol{\beta}_n}}\right) - \Phi\left(\frac{H_{c(n)}^{c-1} - \boldsymbol{\beta}_n^T\boldsymbol{z}}{\sqrt{1-\boldsymbol{\beta}_n^T\boldsymbol{\beta}_n}}\right) \in [0,1]$$

and model assumptions 1, 2 and 7, which ensure that $\mathrm{LGC}_n^c \leq \mathrm{LGC}_{max} < \infty$, $|\mathrm{EAD}_n| \leq \mathrm{EAD}_{max} < \infty$, and $\sum_{n=1}^{N}\mathrm{EAD}_n \neq 0$, we see that (A.0.18) is finite. That is, $\widetilde{L}_n(\boldsymbol{z},\boldsymbol{\mathcal{E}})$ has a finite second moment.

Next, we show that Linderberg's condition (A.0.1) is satisfied with $\mathcal{X}_n = \widetilde{L}_n(\boldsymbol{z},\boldsymbol{\mathcal{E}})$. To this end, choose any $\epsilon > 0$ and let

$$\begin{aligned}K(N) &= \frac{1}{\left(\sigma^{(N)}(\boldsymbol{z})\right)^2}\sum_{n=0}^{N-1}\int_{\left\{|x-\widetilde{\mu}_n(\boldsymbol{z})|\geq\epsilon\sigma^{(N)}(\boldsymbol{z})\right\}}\left(x-\widetilde{\mu}_n(\boldsymbol{z})\right)^2 dF_n(x)\\ &= \frac{1}{\left(\sigma^{(N)}(\boldsymbol{z})\right)^2}\sum_{n=0}^{N-1}\mathbb{E}\left[\left(\widetilde{L}_n(\boldsymbol{z},\boldsymbol{\mathcal{E}})-\widetilde{\mu}_n(\boldsymbol{z})\right)^2\mathbb{I}_{\left\{|\widetilde{L}_n(\boldsymbol{z},\boldsymbol{\mathcal{E}})-\widetilde{\mu}_n(\boldsymbol{z})|\geq\epsilon\sigma^{(N)}(\boldsymbol{z})\right\}}\right],\end{aligned}$$

where

$$\begin{aligned}\widetilde{\mu}_n\left(\boldsymbol{z}\right) &= \mathbb{E}\left[\widetilde{L}_n(\boldsymbol{z},\boldsymbol{\mathcal{E}})\right] = \omega_n\left(\sum_{c=0}^{C-1}\mathrm{LGC}_n^c p_n^c\left(\boldsymbol{z}\right)\right),\\ \left(\sigma^{(N)}(\boldsymbol{z})\right)^2\left(\boldsymbol{z}\right) &= \mathbb{V}\left[L^{(N)}(\boldsymbol{z},\boldsymbol{\mathcal{E}})\right] = \sum_{n=0}^{N-1}\omega_n^2\mathbb{V}\left[L_n(\boldsymbol{z},\boldsymbol{\mathcal{E}})\right].\end{aligned}$$

Since

$$\begin{aligned}\left|\widetilde{L}_n(\boldsymbol{z},\boldsymbol{\mathcal{E}})\right| &= |\omega_n|\left|\sum_{c=0}^{C-1}\mathrm{LGC}_n^c\mathbb{I}_{\left\{\frac{H_{c(n)}^{c-1}-\beta_n z_{s(n)}}{\sqrt{1-\beta_n^2}}\leq\mathcal{E}_n<\frac{H_{c(n)}^{c}-\beta_n z_{s(n)}}{\sqrt{1-\beta_n^2}}\right\}}\right|\\ &\leq \sup_n\left\{|\omega_n|\right\}\mathrm{LGC}_{max},\end{aligned}$$

and

$$\begin{aligned}|\widetilde{\mu}_n\left(\boldsymbol{z}\right)| &= |\omega_n|\left|\sum_{c=0}^{C-1}\mathrm{LGC}_n^c p_n^c\left(\boldsymbol{z}\right)\right|\\ &\leq \sup_n\left\{|\omega_n|\right\}\mathrm{LGC}_{max},\end{aligned}$$

it follows that

$$\begin{aligned}\left|\widetilde{L}_n(\boldsymbol{z},\boldsymbol{\mathcal{E}})-\widetilde{\mu}_n(\boldsymbol{z})\right| &\leq \left|\widetilde{L}_n(\boldsymbol{z},\boldsymbol{\mathcal{E}})\right|+|\widetilde{\mu}_n(\boldsymbol{z})| \\ &\leq 2\sup_n\{|\omega_n|\}\,\mathrm{LGC}_{max.}\end{aligned}$$

Therefore,

$$\begin{aligned}K(N) &\leq \frac{4\mathrm{LGC}_{max}^2\left(\sup_n\{|\omega_n|\}\right)^2}{\left(\sigma^{(N)}(\boldsymbol{z})\right)^2}\sum_{n=0}^{N-1}\mathbb{E}\left[\mathbb{I}_{\left\{\left|\widetilde{L}_n(\boldsymbol{z},\boldsymbol{\mathcal{E}})-\widetilde{\mu}_n(\boldsymbol{z})\right|\geq\epsilon\sigma^{(N)}(\boldsymbol{z})\right\}}\right] \\ &= \frac{4\mathrm{LGC}_{max}^2\left(\sup_n\{|\omega_n|\}\right)^2}{\left(\sigma^{(N)}(\boldsymbol{z})\right)^2}\sum_{n=0}^{N-1}\mathbb{P}\left\{\left|\widetilde{L}_n(\boldsymbol{z},\boldsymbol{\mathcal{E}})-\widetilde{\mu}_n(\boldsymbol{z})\right|\geq\epsilon\sigma^{(N)}(\boldsymbol{z})\right\}.\end{aligned}$$

By the generalized Chebyshev's inequality,

$$\forall\,\epsilon>0,\ \mathbb{P}\left\{\left|\widetilde{L}_n(\boldsymbol{z},\boldsymbol{\mathcal{E}})-\widetilde{\mu}_n(\boldsymbol{z})\right|\geq\epsilon\sigma^{(N)}(\boldsymbol{z})\right\}\leq\frac{\mathbb{V}\left[\widetilde{L}_n(\boldsymbol{z},\boldsymbol{\mathcal{E}})\right]}{\epsilon^2\left(\sigma^{(N)}(\boldsymbol{z})\right)^2}.$$

Thus,

$$\begin{aligned}K(N) &\leq \frac{4\mathrm{LGC}_{max}^2\left(\sup_n\{|\omega_n|\}\right)^2}{\epsilon^2\left(\sigma^{(N)}(\boldsymbol{z})\right)^2}\sum_{n=0}^{N-1}\frac{\mathbb{V}\left[\widetilde{L}_n(\boldsymbol{z},\boldsymbol{\mathcal{E}})\right]}{\left(\sigma^{(N)}(\boldsymbol{z})\right)^2} \\ &= \frac{4\mathrm{LGC}_{max}^2\left(\sup_n\{|\omega_n|\}\right)^2}{\epsilon^2\left(\sigma^{(N)}(\boldsymbol{z})\right)^2}.\end{aligned}$$

Lemma A.2 gives that $\left(\sigma^{(N)}(\boldsymbol{z})\right)^2\geq\sigma_{min}^2(\boldsymbol{z})\sum_{n=0}^{N-1}\omega_n^2$, whence

$$K(N)\leq\frac{4\mathrm{LGC}_{max}^2}{\epsilon^2\sigma_{min}^2(\boldsymbol{z})}\cdot\frac{\left(\sup_n\{|\omega_n|\}\right)^2}{\sum_{n=1}^{N}\omega_n^2}.$$

Using Cauchy-Schwartz inequality, we have

$$\sum_{n=0}^{N-1}\omega_n^2\cdot\sum_{n=0}^{N-1}1^2\geq\left(\sum_{n=0}^{N-1}\omega_n\cdot 1\right)^2,$$

from which it follows that

$$\sum_{n=0}^{N-1}\omega_n^2\geq\frac{1}{N},$$

since $\sum_{n=0}^{N-1}\omega_n=1$. Hence,

$$K(N)\leq\frac{4\mathrm{LGC}_{max}^2}{\epsilon^2\sigma_{min}^2(\boldsymbol{z})}\left(\sup_n\{|\omega_n|\}\right)^2 N.$$

The hypothesis of Theorem 4.1 ensures that there exists a $\delta > 0$ such that $\sup_n \{|\omega_n|\} = O\left(N^{-(1/2+\delta)}\right)$. This implies that $\left(\sup_n \{|\omega_n|\}\right)^2 N = O\left(N^{-2\delta}\right)$ for some positive δ, whence

$$K(N) \leq \frac{4\mathrm{LGC}_{max}^2}{\epsilon^2 \sigma_{min}^2(\boldsymbol{z})} O\left(N^{-2\delta}\right).$$

Since $\mathrm{LGC}_{max} < \infty$ and $\sigma_{min}^2(\boldsymbol{z}) > 0$, for any fixed $\epsilon > 0$, we have

$$K(N) \to 0 \text{ as } N \to \infty.$$

Thus we have shown that Linderberg's condition (A.0.1) holds. Therefore, as a consequence of Lemma A.1, we have

$$\frac{L^{(N)}(\boldsymbol{z}, \boldsymbol{\mathcal{E}}) - \mathbb{E}\left[L^{(N)}(\boldsymbol{z}, \boldsymbol{\mathcal{E}})\right]}{\sqrt{\mathbb{V}\left[L^{(N)}(\boldsymbol{z}, \boldsymbol{\mathcal{E}})\right]}} \xrightarrow{d} \mathcal{N}(0,1), \text{ as } N \to +\infty.$$

Appendix B

Proof of Theorem 4.2

We prove Theorem 4.2 for the 1-dimensional case, for which there is only one systematic risk factor ($S = 1$). That is, for all $l \in \mathbb{R}$,

$$\mathbb{P}\left\{L^{(N)}(\mathcal{Z}, \boldsymbol{\mathcal{E}}) \leq l\right\} - \int_{\mathbb{R}} \Phi\left(\frac{l - \mu^{(N)}(z)}{\sigma^{(N)}(z)}\right) d\Phi(\boldsymbol{z}) \to 0 \text{ as } N \to \infty, \tag{B.0.1}$$

where $\Phi(\boldsymbol{z})$ is the CDF of the standard normal distribution. The proof can be generalized easily to the multi-dimensional case.

To show (B.0.1), we prove below that

$$\lim_{N\to\infty} \int_{-\infty}^{+\infty} f^{(N)}(z) d\Phi(\boldsymbol{z}) = 0,$$

where

$$\begin{aligned} f^{(N)}(z) &= \mathbb{P}\left\{ L^{(N)}(\mathcal{Z}, \boldsymbol{\mathcal{E}}) \leq l \middle| \mathcal{Z} = z \right\} - \Phi\left(\frac{l - \mu^{(N)}(z)}{\sigma^{(N)}(z)}\right) \\ &= \mathbb{P}\left\{L^{(N)}(z, \boldsymbol{\mathcal{E}}) \leq l\right\} - \Phi\left(\frac{l - \mu^{(N)}(z)}{\sigma^{(N)}(z)}\right). \end{aligned}$$

More specifically, we show below that

$$\forall\, l \in \mathbb{R},\ \forall\, \delta > 0,\ \exists\, N^* > 0 \text{ such that } N \geq N^* \Rightarrow \left| \int_{-\infty}^{+\infty} f^{(N)}(z) d\Phi(\boldsymbol{z}) \right| < \delta.$$

First, note that, for all $M \in [0, \infty)$, we have

$$\begin{aligned}\left|\int_{-\infty}^{+\infty} f^{(N)}(z) d\Phi(\boldsymbol{z})\right| &= \left|\int_{+M}^{+\infty} f^{(N)}(z) d\Phi(\boldsymbol{z}) + \int_{-M}^{+M} f^{(N)}(z) d\Phi(\boldsymbol{z}) + \int_{-\infty}^{-M} f^{(N)}(z) d\Phi(\boldsymbol{z})\right| \\ &\leq \int_{+M}^{+\infty} \left|f^{(N)}(z)\right| d\Phi(\boldsymbol{z}) + \int_{-M}^{+M} \left|f^{(N)}(z)\right| \Phi(\boldsymbol{z}) + \int_{-\infty}^{-M} \left|f^{(N)}(z)\right| d\Phi(\boldsymbol{z})\end{aligned}$$

Since both of $\mathbb{P}\left\{L^{(N)}(z, \boldsymbol{\mathcal{E}}) < l\right\}$ and $\Phi\left(\frac{l-\mu^{(N)}(z)}{\sigma^{(N)}(z)}\right)$ are probability measures, $\left|f^{(N)}(z)\right| \leq 1$. Hence,

$$\int_{+M}^{+\infty} \left|f^{(N)}(z)\right| d\Phi(\boldsymbol{z}) \leq \int_{+M}^{+\infty} d\Phi(\boldsymbol{z}) = 1 - \Phi(M).$$

Similarly, $\int_{-\infty}^{-M} \left|f^{(N)}(z)\right| d\Phi(z) \leq 1 - \Phi(M)$. Thus,

$$\left|\int_{-\infty}^{+\infty} f^{(N)}(z) d\Phi(z)\right| \leq 2\left(1 - \Phi(M)\right) + \int_{-M}^{+M} \left|f^{(N)}(z)\right| d\Phi(z).$$

For any $\delta > 0$, we can choose $M > 0$ such that $1 - \Phi(M) < \delta/4$. For such an M, Theorem 4.1 implies that $\left|f^{(N)}(z)\right| \to 0$ point-wise on the closed interval $[-M, M]$. Also, $\left|f^{(N)}(z)\right| \leq g(z) = 1$ and $\int_{-M}^{M} g(z) d\Phi(z) = 2\Phi(M) - 1 < \infty$. Thus, according to the Dominated Convergence Theorem, we have

$$\lim_{N\to\infty} \int_{-M}^{+M} \left|f^{(N)}(z)\right| d\Phi(z) = \int_{-M}^{+M} \lim_{N\to\infty} \left|f^{(N)}(z)\right| d\Phi(z) = 0.$$

Therefore, $\forall\ \delta' = \delta/2 > 0,\ \exists\ N^* > 0$ such that $\forall\ N \geq N^*$

$$\int_{-M}^{+M} \left|f^{(N)}(z)\right| d\Phi(z) \leq \delta' = \frac{\delta}{2}.$$

Hence, $\forall\ l \in \mathbb{R},\ \forall\ \delta > 0,\ \exists\ M \in [0, +\infty),\ \exists\ N^* > 0$ such that $\forall\ N \geq N^*$

$$\begin{aligned}\left|\int_{-\infty}^{+\infty} f^{(N)}(z) d\Phi(z)\right| &\leq 2\left(1 - \Phi(M)\right) + \int_{-M}^{+M} \left|f^{(N)}(z)\right| d\Phi(z) \\ &\leq 2 \cdot \frac{\delta}{4} + \frac{\delta}{2} \\ &= \delta.\end{aligned}$$

Therefore, we conclude that $\mathbb{P}\left\{L^{(N)}(\mathcal{Z}, \boldsymbol{\mathcal{E}}) \leq l\right\} - \int_{\mathbb{R}} \Phi\left(\frac{l-\mu^{(N)}(z)}{\sigma^{(N)}(z)}\right) d\Phi(\mathrm{z}) \to 0,$ as $N \to +\infty$.

Appendix C

Proof of Theorem 4.3

Let $\mathcal{X} = X(\boldsymbol{\mathcal{Z}}) = \mathbb{P}\left\{ L^{(N)}(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{E}}) \leq l \middle| \boldsymbol{\mathcal{Z}} \right\}$, $\boldsymbol{\mathcal{X}} = \left(X\left(\boldsymbol{\mathcal{Z}}^{(1)}\right), \ldots, X\left(\boldsymbol{\mathcal{Z}}^{(U)}\right)\right)^T$, and $\mu_{\mathcal{X}} = \mathbb{E}[X(\boldsymbol{\mathcal{Z}})]$. Then $\mathbb{P}\{L(\boldsymbol{\mathcal{Z}}, \boldsymbol{\mathcal{E}}) \leq l\} = \mu_{\mathcal{X}}$. Since $X\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)$ are independent and identically-distributed, the confidence interval, $[P_\alpha^-(\boldsymbol{\mathcal{X}}), P_\alpha^+(\boldsymbol{\mathcal{X}})]$, for $\mu_{\mathcal{X}}$ at level $1-\alpha$, can be computed as

$$P_\alpha^{\pm}(\boldsymbol{\mathcal{X}}) = \bar{\boldsymbol{\mathcal{X}}} \pm t_{U-1,1-\alpha/2}\sqrt{S^2_{\boldsymbol{\mathcal{X}}}/U}, \tag{C.0.1}$$

where $\bar{\boldsymbol{\mathcal{X}}} = \sum_{u=1}^{U} X\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)/U$, $t_{U-1,1-\alpha/2}$ is the $1-\alpha/2$ critical value for the t distribution with $U-1$ degrees of freedom, and $S^2_{\boldsymbol{\mathcal{X}}}$ is the sample variance of $X(\boldsymbol{\mathcal{Z}})$ defined by

$$S^2_{\boldsymbol{\mathcal{X}}} = \frac{1}{U-1}\sum_{u=1}^{U}\left(X\left(\boldsymbol{\mathcal{Z}}^{(u)}\right) - \bar{\boldsymbol{\mathcal{X}}}\right)^2.$$

That is,

$$\mathbb{P}\left\{\mu_{\mathcal{X}} \in \left[P_\alpha^-(\boldsymbol{\mathcal{X}}), P_\alpha^+(\boldsymbol{\mathcal{X}})\right]\right\} = 1-\alpha.$$

Since we use the CLT approximation to estimate the conditional loss probability $X\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)$,

$$X\left(\boldsymbol{\mathcal{Z}}^{(u)}\right) \approx \Phi\left(\frac{l - \mu^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)}{\sigma^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)}\right), \tag{C.0.2}$$

there is an asymptotic error associated with (C.0.2), $\delta\left(l, \boldsymbol{\mathcal{Z}}^{(u)}\right)$, mentioned earlier in (4.2.4). Therefore,

$$X\left(\boldsymbol{\mathcal{Z}}^{(u)}\right) = \Phi\left(\frac{l - \mu^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)}{\sigma^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)}\right) + \delta\left(l, \boldsymbol{\mathcal{Z}}^{(u)}\right).$$

By (4.2.7), the error function $\delta\left(l, \boldsymbol{\mathcal{Z}}^{(u)}\right)$ can be bounded:

$$\left|\delta\left(l, \boldsymbol{\mathcal{Z}}^{(u)}\right)\right| \leq B\left(\frac{l-\mu^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)}{\sigma^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)}\right).$$

Therefore,

$$X\left(\boldsymbol{\mathcal{Z}}^{(u)}\right) \geq \Phi\left(\frac{l-\mu^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)}{\sigma^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)}\right) - B\left(\frac{l-\mu^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)}{\sigma^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)}\right),$$

and

$$X\left(\boldsymbol{\mathcal{Z}}^{(u)}\right) \leq \Phi\left(\frac{l-\mu^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)}{\sigma^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)}\right) + B\left(\frac{l-\mu^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)}{\sigma^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)}\right).$$

Let

$$X^{\pm}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right) = \Phi\left(\frac{l-\mu^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)}{\sigma^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)}\right) \pm B\left(\frac{l-\mu^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)}{\sigma^{(N)}\left(\boldsymbol{\mathcal{Z}}^{(u)}\right)}\right),$$

$$\boldsymbol{\mathcal{X}}^{\pm} = \left[X^{\pm}\left(\boldsymbol{\mathcal{Z}}^{(1)}\right), \ldots, X^{\pm}\left(\boldsymbol{\mathcal{Z}}^{(U)}\right)\right]^{T},$$

and

$$u_{\alpha}(\boldsymbol{\mathcal{X}}) = P_{\alpha}^{-}\left(\boldsymbol{\mathcal{X}}^{-}\right),\ v_{\alpha}(\boldsymbol{\mathcal{X}}) = P_{\alpha}^{+}\left(\boldsymbol{\mathcal{X}}^{+}\right).$$

Han [23] has shown that both $P_{\alpha}^{+}(\boldsymbol{\mathcal{X}})$ and $P_{\alpha}^{-}(\boldsymbol{\mathcal{X}})$ are non-decreasing functions with respect to each element of $\boldsymbol{\mathcal{X}}$. Hence

$$u_{\alpha}(\boldsymbol{\mathcal{X}}) \leq P_{\alpha}^{-}(\boldsymbol{\mathcal{X}}) \leq P_{\alpha}^{+}(\boldsymbol{\mathcal{X}}) \leq v_{\alpha}(\boldsymbol{\mathcal{X}}).$$

That is, we find an observable interval $[u_{\alpha}(\boldsymbol{\mathcal{X}}), v_{\alpha}(\boldsymbol{\mathcal{X}})]$ bracketing the interval $[P_{\alpha}^{-}(\boldsymbol{\mathcal{X}}), P_{\alpha}^{+}(\boldsymbol{\mathcal{X}})]$:

$$\left[P_{\alpha}^{-}(\boldsymbol{\mathcal{X}}), P_{\alpha}^{+}(\boldsymbol{\mathcal{X}})\right] \subseteq \left[u_{\alpha}(\boldsymbol{\mathcal{X}}), v_{\alpha}(\boldsymbol{\mathcal{X}})\right]. \tag{C.0.3}$$

Since the set $E = \{\mu_{\mathcal{X}} \in [u_{\alpha}(\boldsymbol{\mathcal{X}}), v_{\alpha}(\boldsymbol{\mathcal{X}})]\}$ can be partitioned as $E = E_1 \cup E_2$, $E_1 \cap E_2 = \emptyset$, where

$$E_1 = \left\{\mu_{\mathcal{X}} \in \left[u_{\alpha}(\boldsymbol{\mathcal{X}}), v_{\alpha}(\boldsymbol{\mathcal{X}})\right], \mu_{\mathcal{X}} \in \left[P_{\alpha}^{-}(\boldsymbol{\mathcal{X}}), P_{\alpha}^{+}(\boldsymbol{\mathcal{X}})\right]\right\},$$

$$E_2 = \left\{\mu_{\mathcal{X}} \in \left[u_{\alpha}(\boldsymbol{\mathcal{X}}), v_{\alpha}(\boldsymbol{\mathcal{X}})\right], \mu_{\mathcal{X}} \notin \left[P_{\alpha}^{-}(\boldsymbol{\mathcal{X}}), P_{\alpha}^{+}(\boldsymbol{\mathcal{X}})\right]\right\},$$

we have

$$\begin{aligned}\mathbb{P}\left\{\mu_{\mathcal{X}} \in [u_\alpha(\boldsymbol{\mathcal{X}}), v_\alpha(\boldsymbol{\mathcal{X}})]\right\} &= \mathbb{P}\left\{E_1\right\} + \mathbb{P}\left\{E_2\right\} \\ &\geq \mathbb{P}\left\{E_1\right\}.\end{aligned}$$

Due to (C.0.3), we have $\mathbb{P}\left\{E_1\right\} = \mathbb{P}\left\{\mu_{\mathcal{X}} \in [P_\alpha^-(\boldsymbol{\mathcal{X}}), P_\alpha^+(\boldsymbol{\mathcal{X}})]\right\}$. Thus

$$\begin{aligned}\mathbb{P}\left\{\mu_{\mathcal{X}} \in [u_\alpha(\boldsymbol{\mathcal{X}}), v_\alpha(\boldsymbol{\mathcal{X}})]\right\} &\geq \mathbb{P}\left\{\mu_{\mathcal{X}} \in \left[P_\alpha^-(\boldsymbol{\mathcal{X}}), P_\alpha^+(\boldsymbol{\mathcal{X}})\right]\right\} \\ &= 1 - \alpha.\end{aligned}$$

Appendix D

Proof of Theorem 5.4

To prove Theorem 5.4, we need the following lemmas.

Lemma D.1. *Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be vectors of length K_x and K_y, respectively, where, without loss of generality, we assume $K_x \geq K_y$. Then*

$$\boldsymbol{x} * \boldsymbol{y}[k] = \begin{cases} \displaystyle\sum_{j=0}^{k} x[j]y[k-j] = \sum_{j=0}^{k} x[k-j]y[j], & k = 0, \ldots, K_y - 1, \\ \displaystyle\sum_{j=k-(K_y-1)}^{k} x[j]y[k-j] = \sum_{j=0}^{K_y-1} x[k-j]y[j], & k = K_y, \ldots, K_x - 1, \\ \displaystyle\sum_{j=k-(K_y-1)}^{K_x-1} x[j]y[k-j] = \sum_{j=k-(K_x-1)}^{K_y-1} x[k-j]y[j], & k = K_x, \ldots, (K_x-1)+(K_y-1). \end{cases} \tag{D.0.1}$$

Consequently, if $K_x = K_y = K$, then

$$\boldsymbol{x} * \boldsymbol{y}[k] = \begin{cases} \displaystyle\sum_{j=0}^{k} x[j]y[k-j] = \sum_{j=0}^{k} x[k-j]y[j], & k = 0, \ldots, K-1, \\ \displaystyle\sum_{j=k-(K-1)}^{K-1} x[j]y[k-j] = \sum_{j=k-(K-1)}^{K-1} x[k-j]y[j], & k = K, \ldots, 2(K-1). \end{cases} \tag{D.0.2}$$

Proof. First, we prove

$$\boldsymbol{x} * \boldsymbol{y}[k] = \sum_{j=k-\min\{k,K_y-1\}}^{\min\{k,K_x-1\}} x[j]y[k-j]. \tag{D.0.3}$$

The discrete linear convolution (5.1.10) is equivalent to

$$\boldsymbol{x} * \boldsymbol{y}[k] = \sum_{j=0}^{K_x-1} x[j]y[k-j] \tag{D.0.4}$$

for $k = 0, \ldots, (K_x - 1) + (K_y - 1)$ with the added constraint that the indices j and $K - j$ do not go out of range. That is,

$$\begin{cases} 0 \leq j \leq K_x - 1 \\ 0 \leq k - j \leq K_y - 1 \end{cases} \tag{D.0.5}$$

Since the second pair of inequalities above is equivalent to

$$k - (K_y - 1) \leq j \leq k, \tag{D.0.6}$$

(D.0.6) together with the first pair of inequalities in (D.0.5) imply

$$\max\{0, k - (K_y - 1)\} \leq j \leq \min\{k, K_x - 1\},$$

or equivalently,

$$k - \min\{k, K_y - 1\} \leq j \leq \min\{k, K_x - 1\}.$$

Therefore, (D.0.4) together with the constraints (D.0.5) on its indices is equivalent to

$$\boldsymbol{x} * \boldsymbol{y}[k] = \sum_{j=k-\min\{k,K_y-1\}}^{\min\{k,K_x-1\}} x[j]y[k-j]. \tag{D.0.7}$$

Similarly, we can prove

$$\boldsymbol{x} * \boldsymbol{y}[k] = \sum_{j=k-\min\{k,K_x-1\}}^{\min\{k,K_y-1\}} x[k-j]y[j]. \tag{D.0.8}$$

Finally, (D.0.7) and (D.0.8) are equivalent to (D.0.1) and (D.0.2). □

Lemma D.2. *Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be vectors of length K, then*

$$\boldsymbol{x} \star \boldsymbol{y}[k] = \begin{cases} \displaystyle\sum_{j=0}^{k} x[j]y[k-j] + \sum_{j=k+1}^{K-1} x[j]y[K+k-j], & k = 0, \ldots, K-2, \\ \displaystyle\sum_{j=0}^{K-1} x[j]y[(K-1)-j], & k = K-1. \end{cases} \tag{D.0.9}$$

Proof. For convenience, we repeat the definition of the discrete circular convolution (5.2.1):

$$\boldsymbol{x} \star \boldsymbol{y}[k] = \sum_{j=0}^{K-1} x[j]y\left[(k-j) \bmod K\right]$$

for $k = 0, \dots K-1$. For $j = 0, \dots, k$, since $0 \le k - j < K$,

$$(k-j) \bmod K = k - j. \tag{D.0.10}$$

For $j = k+1, \dots, K-1$, since $-K < k - j < 0$,

$$(k-j) \bmod K = K + k - j. \tag{D.0.11}$$

Therefore, for $k = 0, \dots, K-2$,

$$\begin{aligned} \boldsymbol{x} \star \boldsymbol{y}[k] &= \sum_{j=0}^{k} x[j] y\left[(k-j) \bmod K\right] + \sum_{j=k+1}^{K-1} x[j] y\left[(k-j) \bmod K\right] \\ &= \sum_{j=0}^{k} x[j] y[k-j] + \sum_{j=k+1}^{K-1} x[j] y[K+k-j], \end{aligned}$$

and for $k = K-1$,

$$\begin{aligned} \boldsymbol{x} \star \boldsymbol{y}[k] &= \sum_{j=0}^{K-1} x[j] y\left[(k-j) \bmod K\right] \\ &= \sum_{j=0}^{K-1} x[j] y[(K-1)-j]. \end{aligned}$$

Next we prove Theorem 5.4 by induction for $K > N$. One can use similar arguments to prove this theorem for $K \le N$.

First notice that, for all $n \in \mathbb{N}^+ = \{n \in \mathbb{N} : \ n \ge 1\}$,

$$\begin{aligned} U_k^n &= \max\left\{u \in \mathbb{N} \,|0 \le uK + k \le (n+1)(K-1)\right\} \\ &= \max\left\{u \in \mathbb{N} \,\middle|\, 0 \le u \le (n+1) - \frac{(n+1)+k}{K}\right\} \\ &= (n+1) - \left\lceil \frac{(n+1)+k}{K} \right\rceil. \end{aligned} \tag{D.0.12}$$

For the base case of the induction, let $N = 2$. Since $K > N$,

$$U_k^{N-1} = 2 - \left\lceil \frac{2+k}{K} \right\rceil = \begin{cases} 1, & \text{for } k = 0, \dots, K-2, \\ 0, & \text{for } k = K-1. \end{cases}$$

Thus we need to prove

$$\left(\overset{1}{\underset{n=0}{\circledast}} \boldsymbol{x}_n\right)[k] = \begin{cases} \left(\overset{1}{\underset{n=0}{\circledast}} \boldsymbol{x}_n\right)[k] + \left(\overset{1}{\underset{n=0}{\circledast}} \boldsymbol{x}_n\right)[K+k], & \text{for } k = 0, \ldots, K-2, \\ \left(\overset{1}{\underset{n=0}{\circledast}} \boldsymbol{x}_n\right)[k], & \text{for } k = K-1. \end{cases} \tag{D.0.13}$$

Applying Lemma D.2, we have

$$\begin{aligned} \left(\overset{1}{\underset{n=0}{\circledast}} \boldsymbol{x}_n\right)[k] &= \boldsymbol{x}_0 \star \boldsymbol{x}_1[k] \\ &= \begin{cases} \displaystyle\sum_{j=0}^{k} x_0[j]x_1[k-j] + \sum_{j=k+1}^{K-1} x_0[j]x_1[K+k-j], & k = 0, \ldots, K-2, \\ \displaystyle\sum_{j=0}^{K-1} x_0[j]x_1[(K-1)-j], & k = K-1. \end{cases} \end{aligned} \tag{D.0.14}$$

For $k = 0, \ldots, K-1$, we have

$$\left(\overset{1}{\underset{n=0}{\circledast}} \boldsymbol{x}_n\right)[k] = \boldsymbol{x}_0 * \boldsymbol{x}_1[k] = \sum_{j=0}^{k} x_0[j]x_1[k-j] \tag{D.0.15}$$

by the first equation in (D.0.2). For $k = 0, \ldots, K-2$, we have $K+k = K, \ldots, 2(K-1)$, whence, by the second equation in (D.0.2),

$$\left(\overset{1}{\underset{n=0}{\circledast}} \boldsymbol{x}_n\right)[K+k] = \boldsymbol{x}_0 * \boldsymbol{x}_1[K+k] = \sum_{j=k+1}^{K-1} x_0[j]x_1[K+k-j]. \tag{D.0.16}$$

Substituting (D.0.15) and (D.0.16) into (D.0.14), we obtain (D.0.13). Therefore, (5.2.5) is true for $N = 2$.

For the induction step, suppose (5.2.5) is true for some $N \geq 2$ and $N < K$. That is

$$\tilde{x}^{\star}_{N-1}[k] = \sum_{u=0}^{U_k^{N-1}} \tilde{x}_{N-1}[uK+k]. \tag{D.0.17}$$

We prove below that (5.2.5) is true for $N+1$. Denote $\tilde{\boldsymbol{x}}_{N-1} \doteq \overset{N-1}{\underset{n=0}{\circledast}} \boldsymbol{x}_n$ and $\tilde{\boldsymbol{x}}^{\star}_{N-1} \doteq \overset{N-1}{\underset{n=0}{\circledast}} \boldsymbol{x}_n$. Since

$$\overset{(N+1)-1}{\underset{n=0}{\circledast}} \boldsymbol{x}_n = \tilde{\boldsymbol{x}}^{\star}_{N-1} \star \boldsymbol{x}_N,$$

and both $\tilde{\boldsymbol{x}}^{\star}_{N-1}$ and $\boldsymbol{x}_N$ are of length K, by Lemma D.2, we have

$$\left(\mathop{\circledast}_{n=0}^{(N+1)-1} \boldsymbol{x}_n\right)[k] = \begin{cases} \displaystyle\sum_{j=0}^{k} \tilde{x}^{\star}_{N-1}[j] x_N[k-j] + \sum_{j=k+1}^{K-1} \tilde{x}^{\star}_{N-1}[j] x_N[K+k-j], & k = 0, \ldots, K-2, \\ \displaystyle\sum_{j=0}^{K-1} \tilde{x}^{\star}_{N-1}[j] x_N[(K-1)-j], & k = K-1. \end{cases}$$

For $k = 0, \ldots, K-N-1$,

$$\left(\mathop{\circledast}_{n=0}^{(N+1)-1} \boldsymbol{x}_n\right)[k] = \sum_{j=0}^{k} \tilde{x}^{\star}_{N-1}[j] x_N[k-j] + \sum_{j=k+1}^{K-N} \tilde{x}^{\star}_{N-1}[j] x_N[K+k-j] + \sum_{j=K-N+1}^{K-1} \tilde{x}^{\star}_{N-1}[j] x_N[K+k-j].$$

Substituting (D.0.17) into the equation above, we have

$$\begin{aligned} \left(\mathop{\circledast}_{n=0}^{(N+1)-1} \boldsymbol{x}_n\right)[k] = & \sum_{j=0}^{k} \sum_{u=0}^{U_j^{N-1}} \tilde{x}_{N-1}[uK+j] x_N[k-j] \\ & + \sum_{j=k+1}^{K-N} \sum_{u=0}^{U_j^{N-1}} \tilde{x}_{N-1}[uK+j] x_N[K+k-j] \\ & + \sum_{j=K-N+1}^{K-1} \sum_{u=0}^{U_j^{N-1}} \tilde{x}_{N-1}[uK+j] x_N[K+k-j]. \end{aligned}$$

From (D.0.12), we have

$$U_j^{N-1} = N - \left\lceil \frac{N+j}{K} \right\rceil = \begin{cases} N-1, & j = 0, \ldots, K-N, \\ N-2, & j = K-N+1, \ldots, K-1. \end{cases}$$

Therefore,

$$\left(\circledast_{n=0}^{(N+1)-1} \boldsymbol{x}_n\right)[k] = \sum_{u=0}^{N-1}\sum_{j=0}^{k} \tilde{x}_{N-1}[uK+j]x_N[k-j] \tag{D.0.18}$$

$$+ \sum_{u=0}^{N-1}\sum_{j=k+1}^{K-N} \tilde{x}_{N-1}[uK+j]x_N[K+k-j] \tag{D.0.19}$$

$$+ \sum_{u=0}^{N-2}\sum_{j=K-N+1}^{K-1} \tilde{x}_{N-1}[uK+j]x_N[K+k-j]. \tag{D.0.20}$$

Make the change variable $j^* = k - j$ for (D.0.18) and $j^* = K + k - j$ for (D.0.19) and (D.0.20) to get

$$\begin{aligned}
\left(\circledast_{n=0}^{(N+1)-1} \boldsymbol{x}_n\right)[k] &= \sum_{u=0}^{N-1}\sum_{j^*=0}^{k} \tilde{x}_{N-1}[uK+k-j^*]x_N[j^*] \\
&\quad + \sum_{u=0}^{N-1}\sum_{j^*=N+k}^{K-1} \tilde{x}_{N-1}[(u+1)K+k-j^*]x_N[j^*] \\
&\quad + \sum_{u=0}^{N-2}\sum_{j^*=k+1}^{N+k-1} \tilde{x}_{N-1}[(u+1)K+k-j^*]x_N[j^*] \\
&= \sum_{u=0}^{N-1}\sum_{j=0}^{k} \tilde{x}_{N-1}[uK+k-j]x_N[j] \\
&\quad + \sum_{u=1}^{N}\sum_{j=N+k}^{K-1} \tilde{x}_{N-1}[uK+k-j]x_N[j] \\
&\quad + \sum_{u=1}^{N-1}\sum_{j=k+1}^{N+k-1} \tilde{x}_{N-1}[uK+k-j]x_N[j] \\
&= \sum_{j=0}^{k} \tilde{x}_{N-1}[k-j]x_N[j] \\
&\quad + \sum_{u=1}^{N-1}\sum_{j=0}^{K-1} \tilde{x}_{N-1}[uK+k-j]x_N[j] \\
&\quad + \sum_{j=N+k}^{K-1} \tilde{x}_{N-1}[NK+k-j]x_N[j].
\end{aligned} \tag{D.0.21}$$

Similarly, for $k = K - N$, we have

$$
\begin{aligned}
&\left(\circledast_{n=0}^{(N+1)-1} \boldsymbol{x}_n\right)[k] \\
&= \sum_{j=0}^{K-N} \tilde{x}^{\star}_{N-1}[j] x_N[(K-N)-j] + \sum_{j=K-N+1}^{K-1} \tilde{x}^{\star}_{N-1}[j] x_N[K+(K-N)-j] \\
&= \sum_{u=0}^{N-1} \sum_{j=0}^{K-N} \tilde{x}_{N-1}[uK+j] x_N[(K-N)-j] + \sum_{u=0}^{N-2} \sum_{j=K-N+1}^{K-1} \tilde{x}_{N-1}[uK+j] x_N[K+(K-N)-j] \\
&= \sum_{u=0}^{N-1} \sum_{j^*=0}^{K-N} \tilde{x}_{N-1}[uK+(K-N)-j^*] x_N[j^*] + \sum_{u=1}^{N-1} \sum_{j^*=(K-N)+1}^{K-1} \tilde{x}_{N-1}[uK+(K-N)-j^*] x_N[j^*] \\
&= \sum_{u=0}^{N-1} \sum_{j=0}^{K-N} \tilde{x}_{N-1}[uK+(K-N)-j] x_N[j] + \sum_{u=1}^{N-1} \sum_{j=(K-N)+1}^{K-1} \tilde{x}_{N-1}[uK+(K-N)-j] x_N[j] \\
&= \sum_{j=0}^{K-N} \tilde{x}_{N-1}[(K-N)-j] x_N[j] + \sum_{u=1}^{N-1} \sum_{j=0}^{K-1} \tilde{x}_{N-1}[uK+(K-N)-j] x_N[j], \qquad \text{(D.0.22)}
\end{aligned}
$$

and for $k = K-N+1, \ldots, K-2$, we have

$$
\begin{aligned}
&\left(\circledast_{n=0}^{(N+1)-1} \boldsymbol{x}_n\right)[k] \\
&= \sum_{j=0}^{K-N} \tilde{x}^{\star}_{N-1}[j] x_N[k-j] + \sum_{j=K-N+1}^{k} \tilde{x}^{\star}_{N-1}[j] x_N[k-j] \\
&\quad + \sum_{j=k+1}^{K-1} \tilde{x}^{\star}_{N-1}[j] x_N[K+k-j] \\
&= \sum_{u=0}^{N-1} \sum_{j=0}^{K-N} \tilde{x}_{N-1}[uK+j] x_N[k-j] + \sum_{u=0}^{N-2} \sum_{j=K-N+1}^{k} \tilde{x}_{N-1}[uK+j] x_N[k-j] \\
&\quad + \sum_{u=0}^{N-2} \sum_{j=k+1}^{K-1} \tilde{x}_{N-1}[uK+j] x_N[K+k-j] \\
&= \sum_{u=0}^{N-1} \sum_{j^*=N-K+k}^{k} \tilde{x}_{N-1}[uK+k-j^*] x_N[j^*] + \sum_{u=0}^{N-2} \sum_{j^*=0}^{N-K+k-1} \tilde{x}_{N-1}[uK+k-j^*] x_N[j^*] \\
&\quad + \sum_{u=1}^{N-1} \sum_{j^*=k+1}^{K-1} \tilde{x}_{N-1}[uK+k-j^*] x_N[j^*] \\
&= \sum_{j=0}^{k} \tilde{x}_{N-1}[k-j] x_N[j] \\
&\quad + \sum_{u=1}^{N-2} \sum_{j=0}^{K-1} \tilde{x}_{N-1}[uK+k-j] x_N[j] \\
&\quad + \sum_{j=N-K+k}^{K-1} \tilde{x}_{N-1}[(N-1)K+k-j] x_N[j], \qquad \text{(D.0.23)}
\end{aligned}
$$

and for $k = K-1$, we have

$$
\begin{aligned}
&\left(\overset{(N+1)-1}{\underset{n=0}{\circledast}} \boldsymbol{x}_n\right)[k] \\
&= \sum_{j=0}^{K-N} \tilde{x}^{\star}_{N-1}[j] x_N[(K-1)-j] + \sum_{j=K-N+1}^{K-1} \tilde{x}^{\star}_{N-1}[j] x_N[(K-1)-j] \\
&= \sum_{u=0}^{N-1} \sum_{j=0}^{K-N} \tilde{x}_{N-1}[uK+j] x_N[(K-1)-j] + \sum_{u=0}^{N-2} \sum_{j=K-N+1}^{K-1} \tilde{x}_{N-1}[uK+j] x_N[(K-1)-j] \\
&= \sum_{u=0}^{N-1} \sum_{j^*=N-1}^{K-1} \tilde{x}_{N-1}[uK+(K-1)-j^*] x_N[j^*] + \sum_{u=0}^{N-2} \sum_{j^*=0}^{N} \tilde{x}_{N-1}[uK+(K-1)-j^*] x_N[j^*] \\
&= \sum_{u=0}^{N-2} \sum_{j^*=0}^{K-1} \tilde{x}_{N-1}[uK+(K-1)-j] x_N[j] + \sum_{j^*=N-1}^{K-1} \tilde{x}_{N-1}[(N-1)K+(K-1)-j] x_N[j] \\
&= \sum_{u=0}^{N-2} \sum_{j=0}^{K-1} \tilde{x}_{N-1}[uK+(K-1)-j] x_N[j] + \sum_{j=N-1}^{K-1} \tilde{x}_{N-1}[NK-1-j] x_N[j]. \qquad \text{(D.0.24)}
\end{aligned}
$$

On the other hand, consider

$$
\sum_{u=0}^{U_k^{(N+1)-1}} \left(\overset{N}{\underset{n=0}{\circledast}} \boldsymbol{x}_n\right)[uK+k] = \sum_{u=0}^{U_k^N} \tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[uK+k].
$$

For $k = 0, \ldots, K-N-1$,

$$
U_k^N = (N+1) - \left\lceil \frac{(N+1)+k}{K} \right\rceil,
$$

Since $N < K$, then

$$
\begin{aligned}
\left\lceil \frac{(N+1)+k}{K} \right\rceil &\leq \left\lceil \frac{(N+1)+K-N-1}{K} \right\rceil = 1, \\
\left\lceil \frac{(N+1)+k}{K} \right\rceil &\geq \left\lceil \frac{N+1}{K} \right\rceil = 1.
\end{aligned}
$$

Therefore, $U_k^N = N$ and

$$
\begin{aligned}
&\sum_{u=0}^{U_k^{(N+1)-1}} \left(\overset{N}{\underset{n=0}{\circledast}} \boldsymbol{x}_n\right)[uK+k] \\
&= \sum_{u=0}^{N} \tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[uK+k] \\
&= \tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[k] + \sum_{u=1}^{N-1} \tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[uK+k] + \tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[NK+k]. \qquad \text{(D.0.25)}
\end{aligned}
$$

Notice that, since $\tilde{\boldsymbol{x}}_{N-1}$ has length $K_x = N(K-1)+1$, $\boldsymbol{x}_N$ has length $K_y = K$, and $0 \leq k \leq K-N-1 \leq$

K_y, by the first equality in (D.0.1),

$$\tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[k] = \sum_{j=0}^{k} \tilde{x}_{N-1}[k-j]x_N[j]. \tag{D.0.26}$$

Also, for $u = 1, \ldots, N-1$,

$$uK + k \leq (N-1)K + K - N - 1 = N(K-1) - 1 < K_x - 1,$$

and

$$uK + k \geq K = K_y,$$

thus by the second equality in (D.0.1),

$$\sum_{u=1}^{N-1} \tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[uK + k] = \sum_{u=1}^{N-1} \sum_{j=0}^{K-1} \tilde{x}_{N-1}[uK + k - j]x_N[j]. \tag{D.0.27}$$

Similarly,

$$NK + k \leq NK + K - N - 1 = (N+1)(K-1) = (K_x - 1) + (K_y - 1),$$

and

$$NK + k \geq NK \geq N(K-1) + 1 = K_x,$$

hence by the third equality in (D.0.1),

$$\begin{aligned} \tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[NK + k] &= \sum_{j=NK+k-N(K-1)}^{K-1} \tilde{x}_{N-1}[NK + k - j]x_N[j] \\ &= \sum_{j=N+k}^{K-1} \tilde{x}_{N-1}[NK + k - j]x_N[j] \end{aligned} \tag{D.0.28}$$

Substituting (D.0.26), (D.0.27) and (D.0.28) into (D.0.21), we obtain

$$\begin{aligned}\sum_{u=0}^{U_k^{(N+1)-1}} \left(\circledast_{n=0}^{N} \boldsymbol{x}_n\right)[uK+k] &= \sum_{j=0}^{k} \tilde{x}_{N-1}[k-j]x_N[j] \\ &\quad + \sum_{u=1}^{N-1}\sum_{j=0}^{K-1} \tilde{x}_{N-1}[uK+k-j]x_N[j] \\ &\quad + \sum_{j=N+k}^{K-1} \tilde{x}_{N-1}[NK+k-j]x_N[j],\end{aligned}$$

which is the same as (D.0.21), implying

$$\left(\circledast_{n=0}^{(N+1)-1} \boldsymbol{x}_n\right)[k] = \sum_{u=0}^{U_k^{(N+1)-1}} \left(\circledast_{n=0}^{N} \boldsymbol{x}_n\right)[uK+k]$$

for $k = 0, \ldots, K-N-1$.

For $k = K-N$,

$$U_k^N = (N+1) - \left\lceil \frac{(N+1)+(K-N)}{K} \right\rceil = (N+1) - \left\lceil \frac{K+1}{K} \right\rceil = N-1,$$

thus,

$$\begin{aligned}&\sum_{u=0}^{U_k^{(N+1)-1}} \left(\circledast_{n=0}^{N} \boldsymbol{x}_n\right)[uK+(K-N)] \\ &= \sum_{u=0}^{N-1} \tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[uK+(K-N)] \\ &= \tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[K-N] + \sum_{u=1}^{N-1} \tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[uK+(K-N)]. \end{aligned} \tag{D.0.29}$$

Since $0 \le K-N < K_y - 1$, by the first equality in (D.0.1), we have

$$\tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[K-N] = \sum_{j=0}^{K-N} \tilde{x}_{N-1}[(K-N)-j]x_N[j], \tag{D.0.30}$$

Also, for $u = 1, \ldots, N-1$,

$$uK + k \le (N-1)K + (K-N) = N(K-1) < K_x - 1,$$

and

$$K + (K - N) \geq K = K_y,$$

thus, by the second equality in (D.0.1),

$$\sum_{u=1}^{N-1} \tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[uK + (K-N)] = \sum_{u=1}^{N-1} \sum_{j=0}^{K-1} \tilde{x}_{N-1}[uK + (K-N) - j] x_N[j]. \tag{D.0.31}$$

Substituting (D.0.30) and (D.0.31) into (D.0.29), we obtain

$$\begin{aligned}\sum_{u=0}^{U_k^{(N+1)-1}} \left(\circledast_{n=0}^{N} \boldsymbol{x}_n \right)[uK + (K-N)] = &\sum_{j=0}^{K-N} \tilde{x}_{N-1}[(K-N) - j] x_N[j] \\ &+ \sum_{u=1}^{N-1} \sum_{j=0}^{K-1} \tilde{x}_{N-1}[uK + (K-N) - j] x_N[j],\end{aligned}$$

which is the same as (D.0.22), implying

$$\left(\circledast_{n=0}^{(N+1)-1} \boldsymbol{x}_n \right)[k] = \sum_{u=0}^{U_k^{(N+1)-1}} \left(\circledast_{n=0}^{N} \boldsymbol{x}_n \right)[uK + k]$$

for $k = K - N$.

For $k = K - N + 1, \ldots, K - 2$, $U_k^N = (N+1) - \left\lceil \frac{(N+1)+k}{K} \right\rceil$. Since $K > N$,

$$\begin{aligned}\left\lceil \frac{(N+1)+k}{K} \right\rceil &\leq \left\lceil \frac{(N+1)+K-2}{K} \right\rceil = 1 + \left\lceil \frac{N-1}{K} \right\rceil = 2, \\ \left\lceil \frac{(N+1)+k}{K} \right\rceil &\geq \left\lceil \frac{(N+1)+K-N+1}{K} \right\rceil = 1 + \left\lceil \frac{2}{K} \right\rceil = 2.\end{aligned}$$

Therefore $U_k^N = N - 1$, and

$$\begin{aligned}&\sum_{u=0}^{U_k^{(N+1)-1}} \left(\circledast_{n=0}^{N} \boldsymbol{x}_n \right)[uK + k] \\ &= \sum_{u=0}^{N-1} \tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[uK + k] \\ &= \tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[k] + \sum_{u=1}^{N-2} \tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[uK + k] + \tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[(N-1)K + k].\end{aligned} \tag{D.0.32}$$

Since $0 \leq k \leq K - 2 \leq K_y$, by the first equality in (D.0.1),

$$\tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[k] = \sum_{j=0}^{k} \tilde{x}_{N-1}[k-j]x_N[j]. \tag{D.0.33}$$

Also, for $u = 1, \ldots, N-2$,

$$uK + k \leq (N-2)K + K - 2 = NK - K - 2 \leq N(K-1) = K_x - 1,$$

and

$$uK + k \geq K + (K - N + 1) \geq K = K_y,$$

thus by the second equality in (D.0.1),

$$\sum_{u=1}^{N-2} \tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[uK + k] = \sum_{u=1}^{N-2} \sum_{j=0}^{K-1} \tilde{x}_{N-1}[uK + k - j]x_N[j]. \tag{D.0.34}$$

Similarly, since $K > N$,

$$\begin{aligned}(N-1)K + k &\leq (N-1)K + K - 2 \\ &= NK - 2 \leq NK + (K - N) - 1 \\ &= N(K-1) + (K-1) \\ &= (K_x - 1) + (K_y - 1),\end{aligned}$$

and

$$(N-1)K + k \geq (N-1)K + K - N + 1 = N(K-1) + 1 = K_x,$$

hence, by the third equality in (D.0.1),

$$\begin{aligned}&\tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[(N-1)K + k] \\ &= \sum_{j=(N-1)K+k-N(K-1)}^{K-1} \tilde{x}_{N-1}[(N-1)K + k - j]x_N[j] \qquad \text{(D.0.35)} \\ &= \sum_{j=N-K+k}^{K-1} \tilde{x}_{N-1}[(N-1)K + k - j]x_N[j]. \qquad \text{(D.0.36)}\end{aligned}$$

Substituting (D.0.33), (D.0.34) and (D.0.35) into (D.0.21), we obtain

$$\begin{aligned}\sum_{u=0}^{U_k^{(N+1)-1}} \left(\circledast_{n=0}^{N} \boldsymbol{x}_n\right)[uK+k] &= \sum_{j=0}^{k} \tilde{x}_{N-1}[k-j]x_N[j] \\ &\quad + \sum_{u=1}^{N-2}\sum_{j=0}^{K-1} \tilde{x}_{N-1}[uK+k-j]x_N[j] \\ &\quad + \sum_{j=N-K+k}^{K-1} \tilde{x}_{N-1}[(N-1)K+k-j]x_N[j],\end{aligned}$$

which is the same as (D.0.23), implying

$$\left(\circledast_{n=0}^{(N+1)-1} \boldsymbol{x}_n\right)[k] = \sum_{u=0}^{U_k^{(N+1)-1}} \left(\circledast_{n=0}^{N} \boldsymbol{x}_n\right)[uK+k]$$

for $k = K-N+1, \ldots, K-2$.

For $k = K-1$, since $K > N$,

$$U_k^N = (N+1) - \left\lceil \frac{(N+1)+(K-1)}{K} \right\rceil = N - \left\lceil \frac{N}{K} \right\rceil = N-1,$$

and

$$\begin{aligned}&\sum_{u=0}^{U_k^{(N+1)-1}} \left(\circledast_{n=0}^{N} \boldsymbol{x}_n\right)[uK+(K-1)] \\ &= \sum_{u=0}^{N-1} \tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[uK+(K-1)] \\ &= \tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[K-1] + \sum_{u=1}^{N-2} \tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[uK+(K-1)] + \tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[(N-1)K+(K-1)]. \\ &= \tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[K-1] + \sum_{u=1}^{N-2} \tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[uK+(K-1)] + \tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[NK-1]. \qquad \text{(D.0.37)}\end{aligned}$$

Since $0 \le K-1 < K_y - 1$, by the first equality in (D.0.1), we have

$$\tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[K-1] = \sum_{j=0}^{K-1} \tilde{x}_{N-1}[(K-1)-j]x_N[j], \qquad \text{(D.0.38)}$$

Also, for $u = 1, \ldots, N-2$,

$$uK + k \le (N-2)K + (K-1) = NK - K - 1 < N(K-1) = K_x - 1,$$

and

$$K + (K-1) \geq K = K_y,$$

thus by the second equality in (D.0.1),

$$\sum_{u=1}^{N-2} \tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[uK + (K-1)] = \sum_{u=1}^{N-2} \sum_{j=0}^{K-1} \tilde{x}_{N-1}[uK + (K-1) - j] x_N[j]. \tag{D.0.39}$$

Also, since $K > N$

$$NK - 1 \leq NK + (K-N) - 1 = N(K-1) + (K-1) = (K_x - 1) + (K_y - 1),$$

and since $N \geq 2$

$$NK - 1 \geq NK - (N-1) = N(K-1) + 1 = K_y,$$

hence, by the third equality in (D.0.1),

$$\begin{aligned} \tilde{\boldsymbol{x}}_{N-1} * \boldsymbol{x}_N[NK-1] &= \sum_{j=NK-1-N(K-1)}^{K-1} \tilde{x}_{N-1}[NK-1-j] x_N[j] \\ &= \sum_{j=N-1}^{K-1} \tilde{x}_{N-1}[NK-1-j] x_N[j]. \end{aligned} \tag{D.0.40}$$

Substituting (D.0.38), (D.0.39) and (D.0.40) into (D.0.37), we obtain

$$\begin{aligned} &\sum_{u=0}^{U_k^{(N+1)-1}} \left(\circledast_{n=0}^{N} \boldsymbol{x}_n \right) [uK + (K-N)] \\ &= \sum_{u=0}^{N-2} \sum_{j=0}^{K-1} \tilde{x}_{N-1}[uK + (K-1) - j] x_N[j] \\ &\quad + \sum_{j=N-1}^{K-1} \tilde{x}_{N-1}[NK - 1 - j] x_N[j], \end{aligned}$$

which is the same as (5.2.6), implying

$$\left(\circledast_{n=0}^{(N+1)-1} \boldsymbol{x}_n \right) [k] = \sum_{u=0}^{U_k^{(N+1)-1}} \left(\circledast_{n=0}^{N} \boldsymbol{x}_n \right) [uK + k]$$

for $k = K - 1$.

Therefore, we have proved that

$$\left(\overset{(N+1)-1}{\underset{n=0}{\circledast}} \boldsymbol{x}_n\right)[k] = \sum_{u=0}^{U_k^{(N+1)-1}} \left(\overset{N}{\underset{n=0}{\circledast}} \boldsymbol{x}_n\right)[uK+k]$$

for $k = 0, \ldots, K-1$, which implies (5.2.5) is true for $N+1$. This completes the proof. □

Bibliography

[1] A. Akavia. Deterministic sparse Fourier approximation via fooling arithmetic progressions. *Proc. 23rd Conf. Learning Theory (COLT)*, pages 381–393, 2010,.

[2] Haitham Al-Hassanieh. The sparse Fourier transform : theory & practice. *Thesis: Ph. D., Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science*, 2016.

[3] Renzo G. Avesani, Kexue Liu, Alin Mirestean, and Jean Salvati. Review and implementation of credit risk models of the financial sector assessment program. *IMF Working Paper*, 2006.

[4] Andrew C. Berry. The accuracy of the Gaussian approximation to the sum of independent variates. *Transactions of the American Mathematical Society*, 49:122–136, 1941.

[5] Algimantas Bikelis. Estimates of the remainder term in the central limit theorem. *Litovsk. Mat. Sb.*, 6:323–346, 1966.

[6] Christian Bluhm, Ludger Overbeck, and Christoph Wagner. *An Introduction to Credit Risk Modelling*. Chapman & Hall, 2003.

[7] P. Boufounos, V. Cevher, A. C. Gilbert, Y. Li, and M. J. Strauss. What's the frequency, Kenneth? sublinear Fourier sampling off the grid. *Proc. RANDOM/ APPROX*, 2012.

[8] Gordon E. Carlson. *Signal and Linear System Analysis*, chapter 17, page 683. Houghton Mifflin Co, Boston, 1992.

[9] Po-Ning Chen. Asymptotic refinement of the Berry-Esseen constant. *Unpublished manuscript*, 2002.

[10] Clare Chu. The fast Fourier transform on hypercube parallel computers. *Ph.D. Thesis, Center for Applied Mathematics, Cornell University*, 1988.

[11] Peter Crosbie and Jeff Bohn. Modeling default risk. *Moody's KMV Company White Paper*, 2003.

[12] Micheal Crouhy, Dan Galai, and Robert Mark. A comparative analysis of current credit risk models. *Journal of Banking and Finance*, 24:59–117, 2000.

[13] Ludovic Dubrana. Credit risk modeling through the use of an extended and numerically stable version of CreditRisk$^+$ and a Merton model. *Working Paper*, 2010.

[14] Carl-Gustav Esseen. Fourier analysis of distribution functions. A mathematical study of the Laplace-Gaussian law. *Acta Mathematica*, 77:1–125, 1945.

[15] Christopher Finger. Conditional approaches for creditmetrics portfolio distributions. *CreditMetrics Monitor*, pages 14–32, April, 1999.

[16] B. Ghazi, H. Hassanieh, P. Indyk, D. Katabi, E. Price, and L. Shi. Sample-optimal average-case sparse Fourier transform in two dimensions. *Proc. Allerton*, 2013.

[17] Paul Glasserman, Wanmo Kang, and Perwez Shahabuddin. Large deviations in multifactor portfolio credit risk. *Mathematical Finance*, 17:345–379, 2007.

[18] Paul Glasserman, Wanmo Kang, and Perwez Shahabuddin. Fast simulation of multifactor portfolio credit risk. *Operations Research*, 56(5):1200–1217, 2008.

[19] Paul Glasserman and Jingyi Li. Importance sampling for portfolio credit risk. *Management Science*, 51(11):1643–1656, 2005.

[20] Michael B. Gordy. A risk-factor model foundation for ratings-based bank capital rules. *Journal of Financial Intermediation*, 12:199–232, 2003.

[21] Michael B. Gordy. Granularity adjustment in portfolio credit risk measurement. In Giorgio Szegö, editor, *Risk Measures for the 21st Century*, pages 109–122. Wiley, 2004.

[22] Greg M. Gupton, Christopher C. Finger, and Mickey Bhatia. The benckmark for understanding credit risk. *CreditMetrics Technical Document*, 1997.

[23] Meng Han. Credit portfolio: Loss distribution and optimization. *Working Notes*, 2010.

[24] Meng Han. Loss distribution and optimization based on the structural model for credit portfolios. *University of Toronto Research Paper*, 2011.

[25] Meng Han, Kenneth R. Jackson, and Alex Kreinin. Approximations for portfolio credit risk in the Gaussian copula factor model. *Working Notes*, 2011.

[26] Fredric J. Harris. On the use of windows for harmonic analysis with the discrete Fourier transform. *Proceedings of the IEEE*, 66(1):51–83, 1978.

[27] H. Hassanieh, F. Adib, D. Katabi, and P. Indyk. Faster GPS via the sparse Fourier transform. *Proc. 18th Annu. Int. Conf. Mobile Networking and Computing (MOBICOM)*, 2012.

[28] H. Hassanieh, P. Indyk, D. Katabi, and E. Price. Near-optimal algorithm for sparse Fourier transform. *Proc. 44th Annu. ACM Symp. Theory of Computing (STOC)*, 2012.

[29] H. Hassanieh, P. Indyk, D. Katabi, and E. Price. Simple and practical algorithm for sparse Fourier transform. *Proc. 23rd Annu. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 2012.

[30] S. Heider, S. Kunis, D. Potts, and M. Veit. A sparse Prony FFT. *Proc. 10th Int. Conf. Sampling Theory and Applications (SAMPTA)*, 2013.

[31] Peter Henrici. *Discrete Variable Methods in Ordinary Differential Equations*, chapter xi, page 407. John Wiley, New York, 1962.

[32] Peter Henrici. *Error Propagation for Difference Methods*, chapter vi, page 73. John Wiley, New York, 1963.

[33] Peter Henrici. *Elements of Numerical Analysis*, chapter xv, page 73. John Wiley, New York, 1964.

[34] Thomas E. Hull and J. Richard Swenson. Tests of probabilistic models for propagation of roundoff errors. *Comminications of the ACM*, 9(2):108–113, 1966.

[35] P. Indyk, M. Kapralov, and E. Price. (Nearly) sample-optimal sparse Fourier transform. *Proc. ACM-SIAM Symp. on Discrete Algorithms*, 2014.

[36] I. Iscoe and A. Kreinin. Large pool approximations. *Encyclopedia of Quantitative Finance*, 2010.

[37] M. A. Iwen. Combinatorial sublinear-time Fourier algorithms. *Found. Computat. Math.*, 10:303–338, 2010.

[38] M. A. Iwen. Improved approximation guarantees for sublinear-time Fourier algorithms. *Appl. Computat. Harm. Anal.*, 34:57–82, 2013.

[39] Toyohisa Kaneko and Liu Bede. Accumulation of round-off error in fast Fourier transforms. *Journal of the ACM*, 17(4):637–654, 1970.

[40] Richard Martin and Tom Wilde. Unsystematic credit risk. *Risk*, 15(11):123–128, 2002.

[41] Mario R. Melchiori. *CreditRisk*$^+$ by FFT. *Working Paper*, 2004.

[42] Sami Merhi, Ruochuan Zhang, Mark A. Iwen, and Andrew Christlieb. A new class of fully discrete sparse Fourier transforms: Faster stable implementations with guarantees. *eprint arXiv:1706.02740*, https://arxiv.org/abs/1706.02740, 2017.

[43] Sandro Merino and Mark Nyfeler. Calculating portfolio loss. *Risk*, pages 82–86, August, 2002.

[44] Robert C. Merton. On the pricing of corporate debt: The risk structure of interest rates. *Journal of Finance*, 29:449–470, 1973.

[45] Reinhard Michel. On the constant in the nonuniform version of the Berry-Esseen theorem. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 55:109–117, 1981.

[46] Sergey V. Nagaev. Some limit theorems for large deviations. *Theory of Probability and its Applications*, 10:214–235, 1965.

[47] Albert H. Nuttall. Some windows with very good sidelobe behavior. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 29(1), 1981.

[48] Ludwig Paditz. Ber die annherung der verteilungsfunktionen von summen unabhngiger zufallsgrben gegen unberrenzt teilbare verteilungsfunktionen unter besonderer berchtung der verteilungsfunktion de standarddisierten normalver-teilung. *Dissertation, Technische Universität Dresden*, 1977.

[49] Ludwig Paditz. On the analytical structure of the constant in the nonuniform version of the Esseen inequality. *Statistics*, 20:453–464, 1989.

[50] S. Pawar and K. Ramchandran. Computing a k-sparse n-length discrete Fourier transform using at most 4k samples and o(k log k) complexity. *Proc. IEEE Int. Symp. Information Theory (ISIT)*, 2013.

[51] Michael Pykhtin. Multi-factor adjustment. *Risk*, 17(3):85–90, 2004.

[52] D. Saunders, C. Xiouros, and C. Zenios. Credit risk optimization using factor models. *Annals of Operations Research*, 152(1):49–77, 2007.

[53] Peter Schaller and Grigory Temnov. Efficient and precise computation of convolutions: applying FFT to heavy tailed distributions. *Journal of Computational Methods in Applied Mathematics*, 8(2):187–200, 2008.

[54] R. Scheibler, S. Haghighatshoar, and M. Vetterli. A sparse sub-linear hadamard transform. *Proc. Allerton*, 2013.

[55] I. S. Shiganov. Refinement of the upper bound of the constant in the central limit theorem. *Journal of Soviet Mathematics*, pages 2545–2550, 1986.

[56] Albert N. Shiryaev. *Probability*. Springer, 1995.

[57] Charles F. Van Loan. *Computational frameworks for the fast Fourier transform*. Frontiers in applied mathematics. SIAM, Philadelphia, 1992.

[58] Oldrich Vasicek. Probability of loss on loan portfolio. *Moody's KMV Company Research Paper*, 1987.

[59] Paul von Beek. An application of Fourier methods to the problem of sharpening the Berry-Esseen inequality. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 23:187–196, 1972.

[60] Tom Wilde. Probing granularity. *Risk*, 14(8):103–106, 2001.