DISCRETE SAMPLING METHODS FOR VECTOR MONTE CARLO CODES

by

Riyanarto Sarno

TR92-072 August 1992

This is an unaltered version of the author's
Ph.D.(CS) Thesis

Faculty of Computer Science
University of New Brunswick
P.O. Box 4400
Fredericton, N.B. E3B 5A3

Phone: (506) 453-4566
Fax: (506) 453-3566

Discrete Sampling Methods For Vector Monte Carlo Codes

by

Riyanarto Sarno

Ir(Electrical Eng.)–Bandung Institute of Technology–1983

Drs(Economics)–University of Padjadjaran–1985

MSc(Computer Science)–University of New Brunswick–1988

A Thesis Submitted in Partial Fulfillment of

the Requirements for the Degree of

Doctor of Philosophy

in the

Faculty of Computer Science

This thesis is accepted.

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

August, 1992

Abstract

Sampling from an arbitrary discrete distribution in Monte Carlo calculations often requires a considerable amount of computing time on sequential computers as it involves table lookup computation. The vectorization of some existing discrete sampling methods is investigated, in an attempt to speed them up. These methods are applied to various problems including a large sparse system of linear equations and neutron transport problems. The codes are written in VS Fortran and implemented on the IBM 3090-180VF. Their performance for scalar and vector processing is evaluated based on both statistical and computational criteria.

To overcome the computational drawbacks of these sampling methods, this thesis proposes a discrete sampling method, named the weighted sampling method, which is especially suited for vector processing. This method utilizes a uniform distribution to construct samples from a probability table. Each sample is subsequently adjusted so that unbiased estimates are obtained. The proposed method enhances the vectorizability of the vector Monte Carlo codes, and achieves better performance for the scalar as well as vector processing in comparison to those of the other sampling methods for the examined problems.

Four variants of the weighted sampling method are also developed. These variants involve stretching of a probability table, sampling from known nonuniform distributions and the combination of two sampling methods. For this purpose, the study of vectorizing the random number generation from binomial and geometric distributions is carried out. It is demonstrated that these variants significantly increase the efficiency of Monte Carlo solutions by reducing the sample variance and decreasing the processing time through vectorization.

Contents

List of Tables

List of Figures

Acknowledgements

I gratefully acknowledge my thesis supervisors Dr. V.C. Bhavsar (Faculty of Computer Science) and Dr. E.M.A. Hussein (Department of Mechanical Engineering) for their continuous guidance, advice and constructive criticism during the course of this research work. I also acknowledge Dr. P.K. Banerjee from Department of Statistics & Mathematics for many helpful discussions.

I wish to thank the Government of Indonesia for giving me the opportunity to study as well as the financial support, and to World University Service of Canada for administering the program. I am also grateful to the Faculty of Computer Science, University of New Brunswick, for awarding me the Graduate Teaching Assistantship, and to Dr. V.C. Bhavsar and Dr. E.M.A. Hussein for giving me the Graduate Research Assistantship through their Natural Sciences and Engineering Research Council of Canada operating grants.

My deepest thanks go to my dearest wife Winta, who always encouraged me with great enthusiasm and determination, and to my beloved children Çisya, Candia and Atrasina, who refreshed the situation in the difficult time.

Finally, I would like to express my sincere feeling of gratitude to my late father Sarno, my mother Riyati, and my parents-in-law Ismanoe and Darmi Susanti for their wholehearted moral support during my study.

Dedication

In the name of God (Allah), Most Gracious, Most Merciful.
Praise be to God, the Cherisher and Sustainer of the worlds,
who teaches mankind new knowledge at every given moment.
This thesis is dedicated to God for
all the favours He has bestowed upon me;
in creating and bringing me to this world.
I pray to God to make this effort sincere for His sake,
in agreement with His wish, and useful for His people.

Nomenclature

$\mathbf{A}$	a matrix
$\mathbf{A}_{ij}$	i-th row and j-th column element of matrix $\mathbf{A}$
ALIAS	code incorporating the alias method
$\mathbf{b}$	a column vector
$\mathbf{b}_i$	i-th element of vector $\mathbf{b}$
$\mathbf{C}$	cumulative probability matrix corresponding $\mathbf{P}$
DISCS	code incorporating Brown's method
$E[X]$	expectation of random variable X
EQUIP	code incorporating the equiprobable method
FSD	fractional standard deviation, defined on page 18
$f(.)$	probability density function
$F(.)$	cumulative probability distribution function
$\binom{n}{j}$	binomial coefficient, defined by $\frac{n!}{j!(n-j)!}$
$\mathbf{F}$	fundamental matrix corresponding $\mathbf{P}$, defined on page 72

$\mathbf{I}$	identity matrix
INVER	code incorporating the inverse method
INWS	code incorporating the weighted sampling with the inverse method
INWSST	code incorporating the weighted sampling using a stretched table with the inverse method
k	number of samples
$\log_2 n$	logarithm of n to the base 2
n	the length of a probability table
$n \times n$	the size of a square matrix
$\mathbf{P}$	transition probability matrix
$p(.)$	probability mass function
$\mathbf{Q}$	substochastic matrix of $\mathbf{P}$, defined on page 72
S^2	sample variance
$\Theta(f(n))$	computational complexity is exactly $f(n)$ function
$Var[X]$	variance of random variable X
V_{max}	maximum vectorization speedup, see page 26
V_{min}	minimum vectorization speedup, see page 26
$\overline{X}$	sample mean
$x(.)$	mass points of random variable X
η_{max}	maximum relative efficiency, see page 19

η_{min}	minimum relative efficiency, see page 19
$\in$	belonging to a set
μ	mean of a probability distribution
$\rho(\mathbf{A})$	spectral radius of matrix $\mathbf{A}$
σ	standard deviation of a probability distribution
σ^2	variance of a probability distribution
WGHTS	code incorporating the weighted sampling method
WSNU	code incorporating the weighted sampling with a nonuniform distribution
WSST	code incorporating the weighted sampling with a stretched table
$\lceil x \rceil$	ceiling of x
$\lfloor x \rfloor$	flooring of x
ξ	a random number uniformly distributed in the interval $(0, 1)$
$\square$	end of proof
$\diamond$	end of example

Notations for the computer codes have the same symbols as the mathematical ones, however, their font types are different. Some notations are as follows.

k	denotes k

`n` denotes n

`p` denotes p

`x` denotes x

`w` denotes w

Chapter 1

Introduction

1.1 Introduction

The Monte Carlo method is generally defined as representing the solution of a problem by a statistical parameter of a hypothetical distribution, and obtaining the estimate of the parameter by drawing random samples from the distribution [32]. This method is used extensively in many scientific and engineering applications [8,17,32,59,77,85]. However, a large computing time is required to obtain estimates within a reasonable statistical confidence interval.

Random sampling is at the core of a Monte Carlo method. The samples may be constructed from continuous or discrete distributions. Sampling from discrete distributions is called discrete sampling [11]. Most of the distributions provided by experimental data are arbitrary discrete distributions, which cannot be described using closed forms of mathematical expressions. A probability table is then used to digitally represent this distribution as a set of mass points with associated probabilities. Therefore, sampling from an arbitrary discrete distribution often requires a considerable amount of computing time on sequential computers as it involves table lookup computation.

Nowadays, many high performance computers use vector processing to speed up

computation [25,61,63,83,93]. These computers have up to thousands of vector processors. In a vector processor, a functional unit, such as an adder or a multiplier, is segmented into a sequence of nearly independent subtasks which are then executed concurrently in a pipelined manner. In such a processor, a high computation rate is achieved by executing identical operations on vectors. This can be done if there are no vectorization inhibitors, such as conditional statements involving indirect addressing and recurrences [45]. In general, the efficiency of vector processing increases as the vector length increases. Thus, these computers can only achieve their best performance when the programs utilize their architectural features.

In view of the availability of vector processors, this thesis carries out the study of the discrete sampling methods for arbitrary discrete distributions and their suitability for vector processing.

1.2 Motivation and Approach

Several methods are presently used to construct samples from arbitrary discrete distributions [11,54,56,60,99]. These sampling methods have computational drawbacks, such as the involvement of step-by-step table searching, the requirement of a large storage, and the complicated procedure to set up the tables for constructing samples. Moreover, these methods do not aim at reducing the sample variances of their solutions.

Many Monte Carlo methods involve random walks, such as Monte Carlo solutions of linear equations [77], particle transport [17,59], statistical physics [8], and operations research [32]. The scalar Monte Carlo codes (the codes developed for sequential computers) are usually ill suited for vector processing [10,90]. Therefore, global and local restructuring of the scalar code is required in order to achieve high vectorizability. This code is referred to as a vector Monte Carlo code, in which the attributes of random walks are stored as vectors and many random walks are then carried out

concurrently [14,79]. In a vector Monte Carlo code, the existing sampling methods do not enhance the vectorizability of the code since they contain vectorization inhibitors, such as a loop containing a `go to`-statement, and conditional statements with indirect addressing. Some of the existing methods [11,60] are suited for vector processing, they require however a large storage or a relatively complex procedure. The performance of these existing sampling methods has not been fully evaluated. That is, at least to our knowledge, no extensive study has been done on discrete sampling methods and their vectorizability.

In this context, this thesis investigates some of the existing discrete sampling methods for sampling from probability tables and asseses their potential for vector processing. We also develop new methods for sampling from probability tables, which are especially suited for vector processing.

First, the above various sampling methods are applied to problems involving one-dimensional probability tables. These methods are implemented in VS Fortran on the IBM 3090-180 with a Vector Facility of the University of New Brunswick. The Monte Carlo efficiency is used as a measure, which combines both the statistical and computational performance.

As an application to problems involving two-dimensional probability tables [17, 32,33,59,77], we consider the Monte Carlo solutions of a large sparse system of linear equations. This particular application is chosen due to the existence of numerical solutions and the tractability of their time complexities. Moreover, similar methods are applicable to solving partial differential equations and integral equations. The discrete sampling methods are applied and their performance is evaluated based on the scalar as well as vector processing time and the efficiency.

The sampling methods are incorporated into the MORSE (Multigroup Oak Ridge Stochastic Experiment) code for solving three neutron transport problems [66]. This thesis examines whether the incorporation of different sampling methods can speed up the computation.

We also develop four variants of the weighted sampling method to enhance the efficiencies of the Monte Carlo solutions, and apply them to problems involving one-dimensional probability tables and applications of the MORSE code.

1.3 Thesis Outline

The next chapter examines the existing methods for sampling from probability tables, and investigates their suitability to scalar as well as vector processing. These methods are used to sample from a one-dimensional probability table. It is shown that the existing sampling methods have some disadvantages when implemented on scalar and vector processors.

A new sampling method, named the weighted sampling method, is proposed in Chapter 3. It is proved that this method is statistically unbiased. Further, based on applications to one-dimensional tables, we demonstrate that it requires simpler coding and shorter execution time on both scalar and vector processors compared with the existing methods.

In Chapter 4, the Monte Carlo methods employing absorbing and ergodic Markov chains for solving a large sparse system of linear equations are investigated. The weighted sampling method and other sampling methods are incorporated into these Monte Carlo methods, and their time complexity is analyzed. Their performance for scalar as well as vector processing is evaluated.

Chapter 5 examines the applications of Monte Carlo methods for solving neutron transport problems using the MORSE code. The weighted sampling method and two other sampling methods are incorporated into this code and their performance is analyzed.

In Chapter 6, four variants of the weighted sampling method are proposed to improve the performance by reducing the sample variance and utilizing vector processing. These variants are used to reexamine the problems discussed in Chapters 3

and 5 for which the weighted sampling method resulted in low efficiencies. It is demonstrated that the use of these variants can improve the efficiencies of the Monte Carlo solutions.

Finally, the contributions of this thesis are summarized in Chapter 7, and some suggestions for future work are given.

Chapter 2

Existing Sampling Methods

2.1 Introduction

This chapter[1] reviews some of the existing methods for sampling from arbitrary discrete distributions, which are represented as probability tables. The procedures of such sampling methods are examined. Their codes are investigated for the suitability in scalar as well as vector processing. For this purpose a one-dimensional probability table is used to demonstrate the statistical results and the processing time of the sampling methods. Further, the codes are implemented on an IBM 3090-180 computer with a vector facility using VS FORTRAN. Some features of this computer are given in Appendix I. Finally, some concluding remarks of this chapter is given in the last section. We will first start by briefly reviewing some definitions in probability theory.

2.2 Preliminaries

A random variable is a rule of correspondence between each possible outcome of an experiment and a numerical value assigned to it [72]. According to the sample space, a random variable is further classified into *discrete* and *continuous* random variables.

[1] An earlier version of this chapter is published in [82].

A random variable X is said to be discrete if it assigns values to at most a countable number of outcomes [56, p. 137]. The probability that the discrete random variable X takes on the outcome x_j is given as

$$P(X = x_j) = p(x_j), \quad \text{for } j = 1, 2, \cdots \tag{2.1}$$

and

$$\sum_{j=1}^{\infty} p_j = 1. \tag{2.2}$$

The *probability mass function* of a discrete random variable X is defined as a function $f(x)$ which describes the probabilities for all outcomes. Further, the cumulative probabilities of the outcomes are described by the *cumulative distribution function* of X, designated as $F(x)$. Similar to the probability mass function for a discrete random variable, the *probability density function* is associated with a continuous random variable.

An arbitrary discrete distribution can be represented by a probability table. The probability table digitally stores the distribution as a set of mass points with associated probabilities. In order to elaborate the procedures of the existing sampling methods, we use Table 2.1 as an example of the probability table.

Table 2.1: An example of a probability table.

Mass points (x)	10	20	30	40	50
Probabilities (p)	0.40	0.20	0.30	0.08	0.02

The objective of sampling from a probability table, which is also called *table lookup sampling*, is to randomly select mass points, x_j , according to the associated probabilities, p_j, for $j = 1, 2, \cdots, n$, with $\sum_{j=1}^{n} p_j = 1$. Several methods exist for such sampling and some of these methods are reviewed in the ensuing sections.

2.3 The Inverse Method

The inverse transform method [56] is the most common method used for generating samples from a discrete distribution as well as a continuous distribution [54]. Let X be a random variable with cumulative distribution function $F(x)$. The method evaluates $X = F^{-1}(\xi)$, where ξ is a uniform random number in the interval $[0, 1]$, to generate samples. It is obvious why this method is called the inverse method since it evaluates the inverse function of the cumulative distribution function.

For the discrete distribution, the inverse method selects $X = x_j$ if $F^{-1}(\xi) = x_j$; that is, $X = x_j$ is generated iff $\sum_{i=1}^{j-1} p_i < \xi \leq \sum_{i=1}^{j} p_i$. Therefore, in the discrete case, the inverse method involves searching for the interval in which the uniformly distributed variable U is located.

The speed of the inverse method depends on the speed of the search, which is affected by two factors: the length of the table (n) and the search procedure. Since the length of the table affects the number of comparisons in the searching, the speed of the inverse method decreases as the value of n increases. In addition, the search procedure can significantly affect the speed. The commonly used procedures for this purpose are the linear search and the binary search. For example, for n mass points distributed uniformly, the average number of comparisons in a linear search is $(n + 1)/2$, whereas, the binary search requires only $\log_2 n$ comparisons [86].

There are two techniques for implementing the inverse method. In the first technique, the probability table is processed before generating samples. Such a processed table is called a *generation table*, which consists of a set of arrays, and is used for generating samples. The procedure for deriving the generation table from a probability table depends on the sampling method. For the inverse method, the generation table is constructed by calculating the cumulative probability distribution. The second technique consists of calculation of the cumulative probability distribution from the probability table while search is being done for generating a sample.

The first technique is more efficient since the cumulative distribution probability is calculated only once; whereas, the second technique calculates the cumulative probability distribution k times, where k is the number of samples. Nevertheless, the second technique is still being used in a large code employing large two-dimensional probability tables because it can utilize the original probability tables, as discussed in Chapter 5.

Table 2.2: The generation table of the inverse method.

x	10	20	30	40	50
c	0.40	0.60	0.90	0.98	1.00

In this chapter, the first technique was chosen. The generation table for sampling from Table 2.1 using the inverse method is given in Table 2.2. This table is constructed by calculating the cumulative probability and storing it into array c. Array x contains the mass points.

```
      do 10 i = 1,k
        r1 = RNUNF()
        do 20 j = 1,n
          if(c(j) .ge. r1) then
            rv(i) = x(j)
            go to 10
          endif
20      continue
10    continue
```

Figure 2.1: Scalar code of the inverse method.

The scalar code of the inverse method, shown in Figure 2.1, generates k samples from a probability table of length n and stores the samples into array rv. The RNUNF function used in this code is an IMSL [46] routine, which generates uniform

random numbers in the interval $(0,1)$.

It should be noted that the notations for the computer codes have the same symbols as the mathematical ones, however, their font types are different. For example k and p are the codes for k and p.

```
            call SURAND(seed1,k,rn1)
            do 10 i = 1,k
              do 20 j = 1,n
                if(c(j) .ge. rn1(i)) then
                  rv(i) = x(j)
                  go to 10
                endif
      20      continue
      10    continue
```

(a) Version 1: the inner loop is vectorizable.

```
                          call SURAND(seed1,k,rn1)
UNAN                      do 10 i = 1,k
VECT +--------              do 20 j = 1,n
     |--------                 if(c(j) .ge. rn1(i)) then
                10          continue
```

(b) The vector compiler report.

Figure 2.2: Vector code for the inverse method (Version 1).

The inverse method can be implemented in two versions of vector codes, as shown in Figure 2.2 (a) and Figure 2.3 (a).

Under VS FORTRAN Version 2 Release 3 [44] and its earlier releases, the vector code shown in Figure 2.2 (a) was not vectorized due to the presence of the go to-statement in the inner loop. However using VS FORTRAN Version 2 Release 4 [45], the inner loop gets vectorized, as shown in Figure 2.2 (b). In the vector compiler

report, UNAN means that the associated loop cannot be vectorized since it contains an unanalyzable exit branch. VECT denotes that the associated loop is vectorized.

The vector codes use a vectorized subroutine, called `SURAND`, to generate uniform random numbers on $(0,1)$. This routine is available in ESSL (Engineering and Scientific Subroutine Library) [43], which contains highly tuned vector subroutines for IBM 3090-180VF. `SURAND(seed1,k,rn1)` uses `seed1` as a seed to generate k random numbers which are stored in array `rn1`.

By restructuring the code, as shown in Figure 2.3 (a), the outer and inner loops become vectorizable. Since k is usually much larger than n, the compiler chooses the outer loop to vectorize in order to result in less processing time. Vector compiler report, shown Figure 2.3 (b), demonstrates that the outer loop is vectorized, while the inner loop is eligible for vectorization (designated by ELIG). This inner loop is not vectorized since the compiler can only choose one loop, which results in less processing time. In vector code Version 2 however, some extra time is required since each sample requires n iterations of the inner loop regardless of the location of an interval in the cumulative distribution in which a random number lies.

Since the searching process in the inverse method is time consuming many efforts have been made to devise alternative methods which do not involve search. Such methods are discussed in the ensuing sections.

```
          call SURAND(seed1,k,rn1)
          do 10 i = 1,k
            if(rn1(i) .le. c(1)) rv(i)=x(1)
            do 20 j = 1,n-1
              if(rn1(i) .gt. c(j)) rv(i)=x(j+1)
    20        continue
    10      continue
```

(a) Version 2: the two loops are vectorizable.

```
                        call SURAND(seed1,k,rn1)
VECT +--------           do 10 i = 1,k
     |                     if(rn1(i) .le. c(1)) rv(i)=x(1)
ELIG |+------             do 20 j = 1,n-1
     ||______               if(rn1(i) .gt. c(j)) rv(i)=x(j+1)
     |_______
```

(b) The vector compiler report.

Figure 2.3: Vector code for the inverse method (Version 2).

2.4 The Equiprobable Method

Marsaglia [60] has proposed an alternative method which does not require searching. We refer to this method as the *equiprobable method* since the generation table is constructed by expanding the probabilities into a larger set of probabilities having equal values. For example, if the smallest probability requires a 3-digit decimal number to represent it then each p_j can be expressed by 0.001 m_j, where m_j is an integer and $m_j \in \{0, 1, ..., 1000\}$. In order to expand all probabilities, p_j for $j = 1, 2, \cdots, n$, the generation table requires $\sum_{j=1}^{n} m_j$ storage locations. The maximum value of this summation is 1000. In general, if the smallest probability is of the form a b-digit decimal, the inverse method requires 10^b extra storage locations to set up the generation table.

Table 2.3: The generation table of the equiprobable method.

x	10(20×)	20(10×)	30(15×)	40(4×)	50(1×)
p	0.02(20×)	0.02(10×)	0.02(15×)	0.02(4×)	0.02(1×)

The generation table of the equiprobable method is shown in Table 2.3. This table is constructed by expanding the probabilities in Table 2.1 into a larger set of equal probabilities. In Table 2.3, $(m\times)$-term denotes that the value is repeated m times.

```
      do 10 i = 1,k
         r1  = RNUNF()
         j   = int(r1 * n) + 1
         rv(i)=x(j)
10    continue
```

Figure 2.4: Scalar code of the equiprobable method.

The scalar and vector codes of the equiprobable method are depicted in Figures 2.4 and 2.5 (a), respectively. Figure 2.5 (b) shows the vector compiler report, which demonstrates that the vector code is fully vectorized.

```
      call SURAND(seed1,k,rn1)
      do 10 i = 1,k
         j    = int(rn1(i) * n) + 1
         rv(i) = x(j)
10    continue
```

(a) The vector code.

```
                      call SURAND(seed1,k,rn1)
VECT +-------         do 10 i = 1,k
     |                   j    = int(rn1(i) * n) + 1
     |_______            rv(i) = x(j)
```

(b) The vector compiler report.

Figure 2.5: Vector code of the equiprobable method.

2.5 The Alias Method

The alias method, proposed by Walker [99], does not require searching and requires only one comparison regardless of the number of the mass points. The algorithm for generating tables of cutoffs and aliases is given in Kronmal [54], and the Fortran code is given by Walker [99]. The alias method requires $2n$ extra storage locations to store the cutoff and alias tables. The correctness of the alias method is based on the following theorem [54] *"Any discrete distribution with a finite number (n) of mass points can be represented as an equiprobable mixture of n distributions, each of them having two mass points."*

Table 2.4: The generation table of the alias method.

x	10	20	30	40	50
a	10	20	10	30	10
f	0.0	0.0	0.9	0.4	0.1

The generation table of the alias method for Table 2.1 is depicted in Table 2.4. The transformation from the probability table to the generation table is carried out systematically according to the above theorem. Array `x` stores the mass points of the first elements of the five two-point distributions, while array `f` stores the associated probabilities. Array `a` stores the mass points of the second elements of the five two-point distributions.

The scalar and vector codes of the alias method are shown respectively in Figure 2.6 and Figure 2.7 (a). The `if`-statement in the vector code is not vectorizable since array `a` is used in conditionally executed code and has non-inductive subscript expressions (indirect addressing). Thus, the vector codes are not fully vectorized as demonstrated by the vector compiler report in Figure 2.7 (b). UNSP denotes that the associated code is linked to some unsupportable statements through mutual dependencies.

```
      do 10 i = 1,k
         r1   = RNUNF()
         r2   = RNUNF()
         j    = int(r1 * n) + 1
         rv(i)= x(j)
         if(r2 .gt. f(j)) rv(i) = a(j)
10    continue
```

Figure 2.6: Scalar code of the alias method.

```
      call SURAND(seed1,k,rn1)
      call SURAND(seed2,k,rn2)
      do 10 i = 1,k
         j(i)  = int(rn1(i) * n) + 1
         rv(i) = x(j(i))
         if(rn2(i) .gt. f(j(i))) rv(i) = a(j(i))
10    continue
```

(a) The vector code.

```
                      call SURAND(seed1,k,rn1)
                      call SURAND(seed2,k,rn2)
VECT +-------         do 10 i = 1,k
     |                   j(i)  = int(rn1(i) * n) + 1
     |_______            rv(i) = x(j(i))

UNSP +-------         do 10 i = 1,k
     |_______            if(rn2(i) .gt. f(j(i))) rv(i) = a(j(i))
```

(b) The vector compiler report.

Figure 2.7: Vector code of the alias method.

2.6 Brown's Method

The discrete sampling method proposed by Brown et al. [11] refers to the same theorem as the alias method; the difference is in the procedure for setting up the tables of n two-point distributions.

Table 2.5: The generation table of Brown's method.

y	10	30	10	10	20
b	50	40	30	10	20
f	0.9	0.6	0.1	0.0	0.0

d	10	50	30	40	10	30	10	10	20	20

Table 2.5 shows the generation table for Brown's method derived from Table 2.1. The construction of the generation table employs the same approach as the alias method; however, the procedure is different. This generation table requires $3n$ extra storage locations to store the probability tables of n two-point distributions. Array `y` stores the mass points of the first elements of the five two-point distributions, while array `f` stores the associated probabilities. Array `b` stores the mass points of the second elements of the five two-point distributions. In order to avoid the use of `if`-statements, as in the case of the alias method, the generation of samples uses array `d` which combines arrays `y` and `b`. The first elements of `y` and `b` arrays are stored into the first and second elements of array `d`. The second elements of `y` and `b` are stored into the third and fourth elements of array `d`, and so on.

The scalar and vector codes of Brown's method are shown in Figure 2.8 and Figure 2.9 (a), respectively. Figure 2.9 (b) shows the vector compiler report, which demonstrates that the vector code is vectorized entirely.

Brown's method requires two random numbers. In order to reduce the computation time, only one random number is generated by a random number generator.

Subsequently, the second random number can be obtained from the first one (see the scalar or vector codes). Since this technique relies on the randomness of the lower order digits of the first random number, it is not recommended for a large number of mass points (see reference [24, page 108]).

2.7 Performance Criteria

In this thesis, it is important to identify specific aspects of a Monte Carlo simulation in order to measure its performance. These aspects include the solution and its variance, the processing time, the scalar and vector speedup, and the efficiency in obtaining the solution.

To measure the variation of a solution we use a fractional standard deviation (FSD) [66], which is defined as

$$\begin{aligned} \text{FSD} &= \frac{\sqrt{\frac{S^2}{k}}}{\overline{X}}, \text{with} \\ S^2 &= \frac{1}{k-1}\sum_{i=1}^{k}(X_i - \overline{X})^2, \text{and} \\ \overline{X} &= \frac{\sum_{i=1}^{k} X_i}{k}, \end{aligned} \tag{2.3}$$

where k is a sample size; $\overline{X}$ and S^2 denote the estimators of the distribution mean and variance, respectively. They are commonly called sample mean and sample variance, respectively [56, p. 145]. Note that we also use these terms in this thesis. The FSD is used since it contains information about the true parameter of the distribution [56, p. 178].

In a Monte Carlo simulation, a respectably high efficiency in obtaining the solution is one of the performance criteria. Hammersley and Handscomb [33] have proposed that the efficiency of a Monte Carlo simulation be defined as a constant inversely proportional to the product of the variance of $\overline{X}$ (which is S^2/k) and the processing time required in obtaining an estimated value (an estimate) of the distribution mean.

This definition is referred to in discussing efficiency in this thesis. In order to increase the efficiency of a Monte Carlo simulation the sample variance (S^2) as well as the processing time (T) should be reduced. Thus, the efficiency combines the measure of statistical and computational performance. The efficiency of Method A relative to Method B is given by

$$\eta = \frac{S_B^2 \times T_B}{S_A^2 \times T_A} \tag{2.4}$$

Based on Equation (2.4), we further define the maximum relative efficiency (η_{max}) if Method B achieves the highest efficiency among those of the other methods. Also, the minimum relative efficiency (η_{min}) if Method B results in the lowest efficiency.

```
      do 10 i = 1,k
         r1   = RNUNF()
         r    = r1 * n + 1.0
         ir   = int(r)
         j    = ir + r - f(ir)
         rv(i)= d(j)
10    continue
```

Figure 2.8: Scalar code of Brown's method.

```
      call SURAND(seed1,k,rn1)
      do 10 i = 1,k
         r    = rn1(i) * n + 1.0
         ir   = int(r)
         j    = ir + r - f(ir)
         rv(i)= d(j)
10    continue
```

(a) The vector code.

```
                    call SURAND(seed1,k,rn1)
VECT +--------      do 10 i = 1,k
     |                 r    = rn1(i) * n + 1.0
     |                 ir   = int(r)
     |                 j    = ir + r - f(ir)
     |_______          rv(i)= d(j)
```

(b) The vector compiler report.

Figure 2.9: Vector code of Brown's method.

2.8 Results and Discussion

The inverse method, the equiprobable method, the alias method, and Brown's method have been implemented on the IBM 3090-180 Vector Facility computer of the University of New Brunswick to sample from a simple probability table. We also examine a scalar routine supported by IMSL, named RNGDA, which applies the alias method. These methods are used to estimate the mean and variance of the distribution.

VS FORTRAN Version 2 Release 4 was used to code the methods. The programs were compiled using optimization OPTION(3), the highest optimization option for scalar and vector processing. Besides reporting the statistical results, this section also demonstrates the processing time, the vectorization speedups, and the efficiencies of the sampling methods examined.

Notations for the scalar codes are as follows.

1. INVER: scalar code of the inverse method,
2. EQUIP: scalar code of the equiprobable method,
3. ALIAS: scalar code of the alias method,
4. DISCS: scalar code of Brown's method, and
5. IMSL: employs the IMSL routine RNGDA, in which the alias method is implemented.

The following notations are for the vector codes.

1. INVR1: vector code of the inverse method, shown in Figure 2.2,
2. INVR2: vector code of the inverse method, shown in Figure 2.3,
3. EQUIP: vector code of the equiprobable method,
4. ALIAS: vector code of the alias method, and
5. DISCS: vector code of Brown's method.

2.8.1 Estimation of Distribution Mean and Variance

This subsection reports the estimated values for the distribution mean, obtained by different sampling methods. The probability table used for sampling is shown in Table 2.6. The distribution mean of this table is 87.431. This probability table was chosen as it results in a relatively large sample variance; which enables verification of the validity of the estimators.

Table 2.6: The probability table used in computation.

Mass points	100	90	70	50	20	15	10	5	2	1
Probabilities	.600	.200	.100	.030	.025	.016	.013	.010	.005	.001

$\mu = 87.431$, $\sigma^2 = 555.99$

Note that the scalar and vector codes produce the same results; therefore, we use only the scalar code notations to report the results. The estimates of the distribution mean together with the associated FSDs obtained using different codes are summarized in Table 2.7. The percentage FSD (%FSD) is the product of FSD and 100.

Table 2.7: Mean estimates and their percentage FSDs.

Sample size	INVER $\bar{X}$(%FSD)	EQUIP $\bar{X}$(%FSD)	ALIAS $\bar{X}$(%FSD)	DISCS $\bar{X}$(%FSD)	IMSL $\bar{X}$(%FSD)
20,000	87.44(0.19)	87.44(0.19)	87.40(0.19)	87.40(0.19)	87.39(0.19)
40,000	87.42(0.14)	87.42(0.14)	87.58(0.13)	87.58(0.13)	87.57(0.13)
60,000	87.41(0.11)	87.41(0.11)	87.56(0.11)	87.56(0.11)	87.56(0.11)
80,000	87.39(0.10)	87.39(0.10)	87.57(0.10)	87.57(0.09)	87.55(0.09)
100,000	87.43(0.09)	87.43(0.09)	87.50(0.09)	87.50(0.08)	87.50(0.08)

It is found, as shown in Table 2.7, that all of the estimated values for the distribution mean obtained using various methods are unbiased, i.e. they estimate a value of the mean almost equal to that of the distribution mean. The FSDs of the existing

methods are relatively equal. For these sample sizes, the estimates of $\overline{X}$ and S^2 are close to μ and σ^2, respectively. Therefore, for each sample size the FSD is almost equal to $\sigma/(\mu\sqrt{k})$. It should be noted that the equiprobable method results in the same estimates as the inverse method. This is due to the fact that the equiprobable method is essentially another version of the inverse transform method.

Table 2.8: Variance estimates and their percentage FSDs.

Sample size	INVER S^2(%FSD)	EQUIP S^2(%FSD)	ALIAS S^2(%FSD)	DISCS S^2(%FSD)	IMSL S^2(%FSD)
20,000	557.9(1.8)	557.9(1.8)	550.8(1.8)	550.1(1.8)	549.6(1.8)
40,000	559.8(1.3)	559.8(1.3)	543.4(1.3)	539.9(1.3)	539.8(1.3)
60,000	558.0(1.0)	558.0(1.0)	547.4(1.0)	541.1(1.0)	541.0(1.0)
80,000	558.9(0.9)	558.9(0.9)	548.9(0.9)	541.1(0.9)	540.9(0.9)
100,000	557.1(0.8)	557.1(0.8)	552.4(0.8)	543.8(0.8)	543.8(0.8)

The distribution variance of the probability table shown in Table 2.6 is 555.99. Table 2.8 summarizes the estimated values of the distribution variance obtained using different methods. As Table 2.8 shows, all sample variances evaluated by the different methods are almost equal to the distribution variance. The estimates of the distribution variance are therefore unbiased.

2.8.2 Processing Time

In order to examine the effects of a probability table's length on the sampling time, we utilize two probability tables with different length. Probability tables of length 10 and 200 ($n = 10$ and $n = 200$) are used for evaluating the processing time of scalar and vector codes. The distribution of table with length 200 is uniform. The sample size ranges from 20,000 to 100,000.

Table 2.9: Scalar processing time for $n = 10$, in milliseconds.

Samples	INVER	EQUIP	ALIAS	DISCS	IMSL
20,000	100.58	83.28	159.33	98.28	94.55
40,000	201.25	166.51	318.64	196.55	188.41
60,000	302.24	249.97	478.46	294.98	282.85
80,000	402.52	333.11	637.26	392.81	378.37
100,000	504.30	416.58	796.89	491.25	471.45

Table 2.9 shows the processing time of the scalar codes for $n = 10$. In this table, the least processing time is for the EQUIP code, followed by the IMSL code; then the processing time increases respectively for the DISCS, INVER, and ALIAS codes. The inverse method (INVER) needs less processing time than the alias method, since INVER requires relatively few comparisons.

Table 2.10: Scalar processing time for $n = 200$, in milliseconds.

Samples	INVER	EQUIP	ALIAS	DISCS	IMSL
20,000	382.25	83.35	159.94	97.90	94.42
40,000	766.99	166.80	320.13	195.93	189.40
60,000	1151.52	250.45	479.60	294.13	283.49
80,000	1538.74	333.99	640.09	392.59	378.68
100,000	1920.76	416.79	798.24	490.12	473.07

Table 2.10 demonstrates the processing time of the scalar codes for $n = 200$. It shows that the least processing time is still for the EQUIP code, followed by the IMSL,

DISCS, and ALIAS codes. The INVER code here requires the largest processing time, since it needs a relatively large number of comparisons. The INVER code does not perform well in the long probability table. The processing time of the EQUIP, IMSL, DISCS and ALIAS codes for $n = 10$ and $n = 200$ are almost equal; whereas, the processing time of the INVER code depends on the value of n.

Table 2.11: Vector processing time for $n = 10$, in milliseconds.

Samples	INVR1	INVR2	EQUIP	ALIAS	DISCS
20,000	34.77	18.23	7.02	20.83	10.98
40,000	69.63	36.38	13.93	41.65	21.83
60,000	104.49	54.72	20.90	62.46	32.74
80,000	139.17	72.75	27.89	83.28	43.65
100,000	174.38	90.94	34.80	104.40	54.53

The processing times of the vector codes for $n = 10$ is shown in Table 2.11. It shows that the least processing time is for the EQUIP code, followed by the DISCS code; then the processing time increases respectively for the INVR2, ALIAS and INVR1 codes. The INVR2 requires less processing time than INVR1, since the outer loop of the INVR2 code gets vectorized and the extra time spent by the inner loop is relatively small. The inner loop of the INVR1 code is vectorizable, but it was not vectorized since the maximum number of loops is only 10.

Table 2.12: Vector processing time for $n = 200$, in milliseconds.

Samples	INVR1	INVR2	EQUIP	ALIAS	DISCS
20,000	190.68	283.67	7.16	21.06	11.00
40,000	381.60	566.98	14.27	42.03	21.93
60,000	573.15	850.28	21.41	62.95	32.89
80,000	763.00	1132.60	28.34	83.87	43.91
100,000	955.29	1418.00	35.41	104.99	54.77

Table 2.12 exhibits the processing times of the vector codes for $n = 200$. In this table, the least processing time is still for the EQUIP code, followed by the DISCS and

ALIAS codes. The EQUIP codes are about 1.6 times faster than the DISCS codes for both $n = 10$ and $n = 200$. The inverse methods (INVR1 and INVR2) consume much larger processing time than the other codes, since the number of comparisons required is large and the benefit of vector processing still cannot compensate it.

Moreover, the INVR1 code requires less processing time than the INVR2 code. It is due to the fact that for $n = 200$ the inner loop of the INVR1 code gets vectorized. Also, the overhead for the INVR2 code is larger than the benefit of the vectorization of its outer loop. INVR2 involves an overhead processing time, because the inner loop is always iterated n times regardless of the location of an interval in the cumulative distribution in which a random number lies.

These processing time tables show that in vector processing, as the case in scalar processing, the equiprobable method, the alias and Brown's methods are not affected by the length of the probability table.

2.8.3 Speedup

This subsection presents *minimum* and *maximum* vectorization speedups. The minimum speedup (V_{min}) is obtained according to processing time of vector codes relative to the smallest processing time of a scalar code, while the maximum speedup is (V_{max}) calculated relative to the largest scalar execution time. The advantage of these results is to demonstrate a possible speedup one can achieve for a given sampling method through vectorization.

Table 2.13 demonstrates the minimum speedups of the vector codes relative to the scalar EQUIP code, which requires the smallest scalar processing time. The speedups of the ALIAS, DISCS and the EQUIP codes are about 4, 7.6 and 12, respectively. The speedup of the INVR1 is about 2.4, while INVR2 is about 4.6.

The maximum speedups of the vector codes (shown in Table 2.14) are calculated relative to the processing time of the scalar INVER code, which exhibits the largest

Table 2.13: Minimum vectorization speedups (V_{min}) for $n = 10$.

Samples	INVR1	INVR2	EQUIP	ALIAS	DISCS
20,000	2.395	4.568	11.863	3.998	7.585
40,000	2.391	4.577	11.953	3.998	7.628
60,000	2.392	4.568	11.960	4.002	7.635
80,000	2.394	4.579	11.944	4.000	7.631
100,000	2.389	4.581	11.971	3.990	7.639

Table 2.14: Maximum vectorization speedups (V_{max}) for $n = 10$.

Samples	INVR1	INVR2	EQUIP	ALIAS	DISCS
20,000	4.582	8.740	22.697	7.649	14.511
40,000	4.576	8.759	22.874	7.650	14.596
60,000	4.579	8.744	22.893	7.660	14.614
80,000	4.579	8.760	22.849	7.652	14.599
100,000	4.570	8.763	22.899	7.633	14.614

scalar execution time. The maximum speedups here are about twice of the corresponding minimum speedups.

Table 2.15: Minimum vectorization speedups (V_{min}) for $n = 200$.

Samples	INVR1	INVR2	EQUIP	ALIAS	DISCS
20,000	0.437	0.294	11.641	3.958	7.577
40,000	0.437	0.294	11.689	3.969	7.606
60,000	0.437	0.295	11.698	3.979	7.615
80,000	0.438	0.295	11.785	3.982	7.606
100,000	0.436	0.294	11.770	3.970	7.610

For $n = 200$, the minimum and maximum vectorization speedups are shown in Tables 2.15 and 2.16, respectively. The minimum speedup of the INVR1 is about 0.4, while INVR2 is about 0.3. Note that the speedups of the INVR1 and INVR2 codes are about the same when $n = 100$; the speedup is about 0.6.

The minimum speedups of EQUIP, ALIAS and DISCS for different probability

Table 2.16: Maximum vectorization speedups (V_{max}) for $n = 200$.

Samples	INVR1	INVR2	EQUIP	ALIAS	DISCS
20,000	2.005	1.348	53.387	18.151	34.750
40,000	2.010	1.353	53.748	18.249	34.974
60,000	2.009	1.354	53.784	18.293	35.011
80,000	2.017	1.359	54.296	18.347	35.043
100,000	2.011	1.355	54.243	18.295	35.070

table length ($n = 10$ and $n = 200$) are relatively equal. It is due to the fact that the scalar smallest execution time, produced by EQUIP, is not affected by the table length.

The maximum speedups for this table are about 4.6 times of the corresponding minimum speedups; it is not the case in $n = 10$. The maximum speedups are much higher than the minimum ones, since the largest scalar time produced by INVER code increases drastically.

2.8.4 Efficiency

This subsection reports the efficiencies of different scalar codes relative to the scalar INVER code, and those of different vector codes relative to the vector INVR1 code. The inverse method is used as a reference in calculating the relative efficiency, because it is the most common method used in the applications; such as in Monte Carlo solutions of a system of linear equations (discussed in Chapter 4), and in Monte Carlo solutions of particle transport problems (presented in Chapter 5).

As discussed in Section 2.7, the relative efficiency of two sampling methods compares their processing time and sample variances. Moreover, the efficiencies demonstrated here can also be interpreted as the speedups of different codes relative to the code for the inverse method in the same processing mode, because their sample variances are relatively equal. For example: in the scalar processing, the efficiencies

also represents the scalar speedups of different scalar codes over the INVER code. Similarly, in the vector processing, the efficiencies demonstrate the speedups of different vector codes relative to the INVR1 code. Therefore, the results presented here support the discussion about speedups in Subsection 2.8.3.

Tables 2.17 and 2.18 demonstrate the efficiencies of different scalar codes relative to the corresponding scalar INVER code for $n = 10$ and $n = 200$, respectively. Since their FSDs are almost equal, these tables also exhibit the scalar speedups of different sampling methods relative to the inverse method for $n = 10$ and $n = 200$, respectively.

Tables 2.19 and 2.20 show the efficiencies of different vector codes relative to the corresponding vector INVR1 code for $n = 10$ and $n = 200$, respectively. Note that these tables also demonstrate the vector speedups of different sampling methods relative to the inverse method for $n = 10$ and $n = 200$, respectively.

Table 2.17: Efficiencies of different scalar codes relative to that of scalar INVER for $n = 10$.

Samples	EQUIP	ALIAS	DISCS	IMSL
20,000	1.208	0.632	1.024	1.065
40,000	1.209	0.730	1.183	1.235
60,000	1.209	0.630	1.021	1.065
80,000	1.208	0.629	1.260	1.309
100,000	1.211	0.632	1.297	1.352

Table 2.18: Efficiencies of different scalar codes relative to that of scalar INVER for $n = 200$.

Samples	EQUIP	ALIAS	DISCS	IMSL
20,000	4.586	2.392	3.908	4.053
40,000	4.598	2.768	4.523	4.680
60,000	4.598	2.393	3.902	4.048
80,000	4.607	2.394	4.819	4.998
100,000	4.608	2.402	4.952	5.130

Table 2.19: Efficiencies of different vector codes relative to that of vector INVR1 for $n = 10$.

Samples	INVR2	EQUIP	ALIAS	DISCS
20,000	1.909	4.953	1.671	3.170
40,000	2.212	5.383	1.932	3.687
60,000	1.903	5.000	1.667	3.181
80,000	1.905	4.990	2.055	3.922
100,000	1.914	5.011	2.111	4.041

Table 2.20: Efficiencies of different vector codes relative to that of vector INVR1 for $n = 200$.

Samples	INVR2	EQUIP	ALIAS	DISCS
20,000	0.673	26.631	9.062	17.354
40,000	0.778	28.798	10.491	20.112
60,000	0.672	26.770	9.074	17.367
80,000	0.671	26.923	11.185	21.374
100,000	0.673	26.978	11.497	22.040

2.9 Concluding Remarks

In this chapter, we have examined the existing methods for sampling from a probability table. The investigation was carried out by measuring several aspects, including processing time, speedup and efficiency.

All of the existing methods result in unbiased estimates. Their estimated values for the distribution mean are almost equal, and as the case for their estimated values of the distribution variance. The inverse method is time consuming when the probability table is long, since the number of comparisons is large. This method is not suited for vectorization either as it usually involves sequential step-by-step searching.

The equiprobable method requires the least processing time in scalar as well as in vector processing, compared to the other methods. If the smallest probability is of the form a b-digit decimal, this method requires 10^b extra storage locations to set up tables for generating samples. The equiprobable method, therefore, requires a huge amount of storage when a very small probability exists in the probability table.

The alias method requires two uniform random numbers and one comparison to generate a random variable regardless the length of a probability table. However, the alias method is not fully vectorizable since it requires `if`-statement involving indirect addressing.

Brown's method is based on the same theory as the alias method. This method can reduce the processing time by eliminating the comparison and calling a random number generator only once for each sample, since the second random number is formed by utilizing the lower order digits of the first one. Therefore, Brown's method is more suitable for vectorization since it avoids the use of `if`-statement. However, Brown's method, as the case of the alias method, requires a relatively complicated procedures to construct the generation tables for sampling. The generation table of Brown's method as well as the alias method requires extra storage of size three times the probability table's size.

Thus, these existing sampling methods have some disadvantages, and only the equiprobable and Brown's methods are suited for vectorization. Moreover, all the existing sampling methods do not aim at reducing the sample variances of their solutions. These facts necessitate the investigation of new sampling methods, which are suited for vector processing and reduce sample variances. For this reason, we propose a new sampling method in Chapter 3.

Chapter 3

Weighted Sampling Method

3.1 Introduction

The existing sampling methods were reviewed and their suitability for vector processing was investigated in the previous section. It was found that the existing sampling methods have some disadvantages in utilizing vector processing facility and in preprocessing the tables for generating samples. Since Monte Carlo methods often require sampling from probability tables, in this chapter therefore, we introduce a vectorizable sampling method, called *weighted sampling method*. It is for constructing samples from arbitrary discrete distributions, which are represented as probability tables.

In this chapter, two probability tables having different characteristics are investigated. It is demonstrated that this method requires simpler coding and shorter execution time for both scalar and vector processing, when compared with the existing methods. The vectorization speedups of the proposed method are demonstrated on an IBM 3090-180 machine with a vector facility. Further, we also exhibit the relative efficiencies of different scalar and vector codes.

This chapter[1] will start by introducing the concept of the weighted sampling method and examining its scalar and vector codes. Subsequently, the statistical

[1] An abridged version of this chapter is published in [83].

analysis of this method is demonstrated in Section 3.3. Sections 3.4 and 3.5 examine two probability tables with different characteristics. In these sections, we discuss some performance criteria including the solutions and their fractional standard deviations, the processing time, the speedups, and the efficiencies of the sampling methods. Finally, some concluding remarks of this chapter are given in Section 3.6.

3.2 The Weighted Sampling Method

This method is based on the concept of importance sampling [77], which is often used in Monte Carlo simulations to minimize the sample variance associated with the estimated quantity. This is achieved by altering the original distribution such that an unbiased estimate of the quantity of interest is obtained with minimum sample variance. Thus, the objective here is solely for variance reduction.

In a similar way, the proposed method alters the given distribution by employing a uniform distribution, which requires a smaller execution time both with scalar and vector processing. However, the main objective of the weighted sampling method is to increase the efficiencies of Monte Carlo solutions by speeding up the generation of samples and reducing the sample variance. This method may not preserve the mass points of the original distribution. It results in, however, unbiased estimated values for the mean and variance of the original distribution.

There are basically two different techniques for implementing the weighted sampling method. First, samples are selected directly from the probability table, and subsequently multiplied by the corresponding adjustment factor. Second, the generation table is constructed by multiplying every mass point with the corresponding adjustment factor. Then, samples are taken from this generation table. It should be noted here that both techniques result in the same estimates.

Let us consider an arbitrary discrete distribution, which is represented by the probability table as follows. The mass points are x_j, for $j = 1, 2, \cdots, n$, while

the associated probabilities are p_j, for $j = 1, 2, \cdots, n$, where $\sum_{j=1}^{n} p_j = 1$. In the first technique, the procedure for generating k samples from this probability is the following. For the i-th sample, the mass point x_i , where $x_i \in \{x_j, j = 1, 2, \cdots, n\}$, is selected from the probability table according to uniform distribution in the interval $[1, n]$. Subsequently, the sampled mass point is multiplied by an adjustment factor, which is defined as the product of the table length (n) and the probability of the selected mass point (p_i), where $p_i \in \{p_j, j = 1, 2, \cdots, n\}$. Thus, the random samples generated are equal to $x_i p_i n$, for $i = 1, 2, \cdots, k$. Thus, the total operations required to generate k samples is $2k$ fetch operations and $2k$ multiplications.

In the second technique, the generation table is a new discrete distribution, which has mass points: $x_j p_j n$, for $j = 1, 2, \cdots, n$, and the associated probability for every mass point: $1/n$. Thus, the mass points in this generation table are distributed uniformly. Therefore, samples can be fetched directly from the generation table according to uniform distribution in the interval $[1, n]$. To construct the generation table, every mass point requires two multiplication operations. The advantage of this technique is that only one fetch operation is required for generating a sample. The number of operations required to construct the generation table from a probability table of length n is $2n$ fetch operations and $2n$ multiplications, while the number of operations needed to generate k samples is only k fetch operations.

The second technique generally requires less processing time than the first one, since k is usually much larger than n. However, it is much easier to incorporate the first technique into a large developed code since the original probability table can be utilized directly. In this chapter, we implement both techniques in order to demonstrate their processing time required for sampling.

For the second technique, the generation table of the weighted sampling is shown in Table 3.1. It is constructed from a probability table demonstrated in Table 2.1. In constructing the generation table, zero probabilities are excluded and the table length is accordingly adjusted. The mass points are multiplied by the adjustment factors:

$x_j p_j n$, for $j = 1, 2, \cdots, 5$, and subsequently stored into an array w, as shown in Table 3.1. The new mass points in array w are distributed uniformly; therefore, the probabilities are not really needed to store in array p.

Note that the method produces the random samples 5, 16, 20 and 45, which are not mass points of the original distribution; whereas the original mass points 10, 20, 30, 40 and 50 will never be generated. However, the method preserves the mean and variance of the original distribution (as shown in Section 3.3) which are the only important quantities in many Monte Carlo simulations.

Table 3.1: The generation table of the weighted sampling method.

w	20	20	45	16	5
p	0.2	0.2	0.2	0.2	0.2

3.2.1 The Computer Codes

This subsection discusses the scalar and vector codes of the weighted sampling method implementing both techniques. The scalar codes of the weighted sampling method implementing techniques I and II are shown in Figures 3.1 and 3.3, respectively. The code generates k samples form a probability table of length n, and stores them into variable rv. The RNUNF function [46] generates uniform random numbers in the interval $(0, 1)$.

The vector code of the weighted sampling method implementing technique I is depicted by Figure 3.2 (a), while its vector compiler report is shown in Figure 3.2 (b). In this code, The SURAND(seed1,k,rn1) subroutine [43] uses seed1 as a seed to generate k random numbers which are stored in array rn1.

For the weighted sampling method using technique II, the vector code and its compiler report are presented in Figures 3.4 (a) and (b), respectively.

```
      do 10 i = 1,k
         r1  = RNUNF()
         j   = int(r1 * n) + 1
         rv(i)=x(j)*p(j)*n
10    continue
```

Figure 3.1: Scalar code of the weighted sampling method using technique I.

```
      call SURAND(seed1,k,rn1)
      do 10 i = 1,k
         j    = int(rn1(i) * n) + 1
         rv(i)=x(j)*p(j)*n
10    continue
```

(a) The vector code.

```
                      call SURAND(seed1,k,rn1)
VECT +--------        do 10 i = 1,k
     |                   j    = int(rn1(i) * n) + 1
     |_______            rv(i) = x(j)*p(j)*n
```

(b) The vector compiler report.

Figure 3.2: Vector code of the weighted sampling method using technique I.

```
      do 10 i = 1,k
         r1  = RNUNF()
         j   = int(r1 * n) + 1
         rv(i)=w(j)
10    continue
```

Figure 3.3: Scalar code of the weighted sampling method using technique II.

```
      call SURAND(seed1,k,rn1)
      do 10 i = 1,k
         j    = int(rn1(i) * n) + 1
         rv(i) = w(j)
10    continue
```

(a) The vector code.

```
                    call SURAND(seed1,k,rn1)
VECT +-------       do 10 i = 1,k
     |                 j    = int(rn1(i) * n) + 1
     |_______          rv(i) = w(j)
```

(b) The vector compiler report.

Figure 3.4: Vector code of the weighted sampling method using technique II.

The vectorization of the weighted sampling method implementing both techniques is straightforward since they contain no logical `if`-statements, and no data dependency which inhibits vectorization. The vector reports, depicted in Figures 3.2 and 3.4, demonstrate that the loop is vectorized since there are no data dependencies in the fetch operations.

3.2.2 Complexities of Scalar Codes

In this subsection, the complexities of different scalar codes are analyzed. The complexities of the vector codes are not examined, since they are similar to those of the scalar codes.

Table 3.2 summarizes the complexities of scalar codes for different sampling methods. Notations for the scalar codes are:

1. INVER: code of the inverse method (Figure 2.1),
2. EQUIP: code of the equiprobable method (Figure 2.4),
3. ALIAS: code of the alias method (Figure 2.6),
4. DISCS: code of Brown's method (Figure 2.8),
5. WGHTS1: code of the weighted sampling implementing technique I (Figure 3.1), and
6. WGHTS2: code of the weighted sampling implementing technique II. (Figure 3.3).

Table 3.2: Complexities of scalar sampling codes.

Operations	INVER	EQUIP	ALIAS	DISCS	WGHTS1	WGHTS2
Fetch	$n/2$	1	3	2	2	1
Addition	0	1	1	2	1	1
Subtraction	0	0	0	1	0	0
Multiplication	0	1	1	1	3	1
Comparison	$n/2$	0	1	0	0	0
Random number	1	1	2	1	1	1

3.3 Statistical Analysis

This section proves that the weighted sampling method provides unbiased estimators of the distribution mean and variance. Unbiased estimators for higher order moments can also be developed in a similar fashion. The subscript w is used for the weighted sampling estimators in order to distinguish them from the corresponding estimators which employ the original distribution. The estimators of mean and variance sampled directly from the original distribution are $\overline{X}$ and S^2, respectively; while, the weighted sampling method provides $\overline{X}_w$andS_w^2. The notations for mass points and the associated probabilities respectively are x_j, and p_j, for $j = 1, 2, \cdots, n$, where $\sum_{j=1}^{n} p_j = 1$.

THEOREM 1 *The estimator* $\overline{X}_w = \frac{1}{k}\sum_{i=1}^{k} p_i n x_i$ *is an unbiased estimator of the distribution mean* μ *whenever the latter exists, where* $p_i \in \{p_j, j = 1, 2, \cdots, n\}$, $x_i \in \{x_j, j = 1, 2, \cdots, n\}$ *and* k *is the sample size.*

Proof.

The expected value of $\overline{X}_w$ can be expressed as

$$\begin{aligned} E[\overline{X}_w] &= E\left[\frac{\sum_{i=1}^{k} p_i n x_i}{k}\right] \\ &= \frac{\sum_{i=1}^{k} E[p_i n x_i]}{k}. \end{aligned} \tag{3.1}$$

Since the samples are generated uniformly and $x_i, i = 1, 2, \cdots, k$, are independent and identically distributed then for every i, the expected value of $p_i n x_i$ can be expressed as

$$\begin{aligned} E[p_i n x_i] &= \sum_{j=1}^{n} \frac{1}{n}(p_j n x_j) \\ &= \mu. \end{aligned} \tag{3.2}$$

The expected value of $\overline{X}_w$ then is as follows.

$$\begin{aligned} E[\overline{X}_w] &= \frac{k\mu}{k} \\ &= \mu. \end{aligned} \tag{3.3}$$

Thus, the expected value of $\overline{X}_w$ is equal to the distribution mean, and $\overline{X}_w$ is therefore an unbiased estimator of μ. □

THEOREM 2 *The estimator* $S_w^2 = \frac{1}{k+1}\sum_{i=1}^{k} p_i n(x_i - \overline{X})^2$ *is an unbiased estimator of the distribution variance* σ^2 *whenever the latter exists, where* $p_i \in \{p_j, j = 1, 2, \cdots, n\}$, $x_i \in \{x_j, j = 1, 2, \cdots, n\}$ *and* k *is the sample size.*

Proof.

S_w^2 can be expressed as

$$\begin{aligned} S_w^2 &= \frac{1}{k+1}\sum_{i=1}^{k} p_i n(x_i - \overline{X})^2 \\ &= \frac{1}{k+1}\sum_{i=1}^{k} p_i n(x_i^2 - 2\overline{X}x_i + \overline{X}^2) \\ &= \frac{1}{k+1}\left(\sum_{i=1}^{k} p_i n x_i^2 - 2k\overline{X}\frac{\sum_{i=1}^{k} p_i n x_i}{k} + \sum_{i=1}^{k} p_i n \overline{X}^2\right) \\ &= \frac{1}{k+1}\left(\sum_{i=1}^{k} p_i n x_i^2 - 2k\overline{X}\overline{X}_w + \sum_{i=1}^{k} p_i n \overline{X}^2\right). \end{aligned} \tag{3.4}$$

The expected value of S_w^2 is as follows

$$\begin{aligned} E[S_w^2] &= E\left[\frac{1}{k+1}\left(\sum_{i=1}^{k} p_i n x_i^2 - 2k\overline{X}\overline{X}_w + \sum_{i=1}^{k} p_i n \overline{X}^2\right)\right] \\ &= \frac{1}{k+1}\left(\sum_{i=1}^{k} E[p_i n x_i^2] - 2kE[\overline{X}]E[\overline{X}_w] + kE[\overline{X}^2]\right). \end{aligned} \tag{3.5}$$

As the samples are generated uniformly,

$$\begin{aligned} E[p_i n(x_i - \mu)^2] &= \sum_{j=1}^{n} p_j (x_j - \mu)^2 \\ &= \sigma^2. \end{aligned} \tag{3.6}$$

By expanding the left hand side we obtain

$$E[p_i n x_i^2] - 2\mu E[p_i n x_i] + E[p_i n \mu^2] = \sigma^2. \tag{3.7}$$

Alternatively,

$$E[p_i n x_i^2] - 2\mu^2 + \mu^2 = \sigma^2. \tag{3.8}$$

Therefore,

$$E[p_i n x_i^2] = \mu^2 + \sigma^2. \tag{3.9}$$

The expected value of $\overline{X}^2$ is given by [95, page 474]

$$E[\overline{X}^2] = \mu^2 + \frac{\sigma^2}{k}. \tag{3.10}$$

Substituting Equations (3.9) and (3.10) into Equation (3.5) the expected value of S_w^2 can be expressed as

$$\begin{aligned} E[S_w^2] &= \frac{1}{k+1}\left(k(\mu^2+\sigma^2) - 2k\mu^2 + k(\mu^2 + \frac{\sigma^2}{k})\right) \\ &= \frac{1}{k+1}\left(k\sigma^2 + \sigma^2\right) \\ &= \sigma^2. \end{aligned} \tag{3.11}$$

Thus, the expected value of S_w^2 is equal to the given distribution variance. Hence, S_w^2 is an unbiased estimator of σ^2. □

3.4 Problem I

In this section, we report the results of the weighted sampling method using techniques I and II, implemented on the IBM 3090-180 Vector Facility computer. This method is used to estimate the mean and variance of a probability table shown in Table 2.6 from Chapter 2.

The codes were written using VS FORTRAN Version 2 Release 4. The programs were compiled using optimization OPTION(3), which exhibits the highest optimization option for scalar and vector processing.

Since the objective of this discussion is to compare the results produced by the weighted sampling method and the existing sampling methods, in this section therefore, some results from Chapter 2 will be shown again.

3.4.1 Estimation of Distribution Mean and Variance

This subsection exhibits the estimated values for the distribution mean obtained by the weighted sampling method. The results are then compared with those from Chapter 2. Table 2.6 was chosen since it gives poor results for the weighted sampling method; i.e. it results in large fractional standard deviations (FSDs) for the weighted sampling.

Notations for the scalar codes are as follows.

1. INVER: code of the inverse method,
2. ALIAS: code of the alias method,
3. DISCS: code of Brown's method,
4. IMSL: employs the IMSL routine RNGDA, in which the alias method is implemented, and
5. WGHTS: code of the weighted sampling method.

Note that the weighted sampling method implementing both techniques produce the same statistical results; therefore, we use only one code notation to exhibit the estimated values for the distribution mean and variance. Also, we do not show the results produced by the vector codes since they are the same as those obtained by the scalar codes.

The equiprobable method is not included here since its statistical results are the same as those of the inverse method. This is due to the same operations involved in their codes.

Table 3.3 summarizes the estimates of the distribution mean together with the associated FSDs evaluated using the different codes. This table shows that all of the mean estimates computed using various methods are unbiased, i.e. they estimate a value, which is almost equal to the distribution mean.

For this problem, the weighted sampling method (WGHTS) produces the largest FSD, which means that its sample variance is the largest.

In general, the sample variance of the weighted sampling method may be smaller or larger than the other methods, depending on the distribution. The weighted sampling results in relatively large FSD when the trend of the mass point values and the associated probabilities are same; i.e. when mass point values decreases and the associated probabilities also decreases, or both of them increase. In Table 2.6 used for this sampling, the values of mass points and the associated probabilities decrease. Thus, we can predict that the weighted sampling will result in poor statistics.

Table 3.4 summarizes the estimated values (estimates) for the distribution variance, obtained using different sampling methods. As this table shows, all sample variances evaluated by the different methods are almost equal to the distribution variance, which is 555.99. The estimates of the distribution variance are therefore unbiased. It should be noted that WGHTS results in the smallest %FSD.

3.4.2 Processing Time

We use two probability tables with different length ($n = 10$ and $n = 200$) for evaluating the processing time of scalar and vector codes. Note that the distribution of table with length 200 is uniform. The sample size ranges from 20,000 to 100,000.

Notations for the scalar codes are as follows.

Table 3.3: Mean estimates and their percentage FSDs.

Sample size	INVER $\overline{X}$(%FSD)	ALIAS $\overline{X}$(%FSD)	DISCS $\overline{X}$(%FSD)	IMSL $\overline{X}$(%FSD)	WGHTS $\overline{X}_w$(%FSD)
20,000	87.44(0.19)	87.40(0.19)	87.40(0.19)	87.39(0.19)	87.76(1.44)
40,000	87.42(0.14)	87.58(0.13)	87.58(0.13)	87.57(0.13)	87.36(1.02)
60,000	87.41(0.11)	87.56(0.11)	87.56(0.11)	87.56(0.11)	87.54(0.84)
80,000	87.39(0.10)	87.57(0.10)	87.57(0.09)	87.55(0.09)	87.47(0.73)
100,000	87.43(0.09)	87.50(0.09)	87.50(0.08)	87.50(0.08)	87.49(0.65)

Table 3.4: Variance estimates and their percentage FSDs.

Sample size	INVER S^2(%FSD)	ALIAS S^2(%FSD)	DISCS S^2(%FSD)	IMSL S^2(%FSD)	WGHTS S_w^2(%FSD)
20,000	557.9(1.8)	550.8(1.8)	550.1(1.8)	549.6(1.8)	554.3(0.46)
40,000	559.8(1.3)	543.4(1.3)	539.9(1.3)	539.8(1.3)	555.5(0.32)
60,000	558.0(1.0)	547.4(1.0)	541.1(1.0)	541.0(1.0)	556.6(0.26)
80,000	558.9(0.9)	548.9(0.9)	541.1(0.9)	540.9(0.9)	556.5(0.23)
100,000	557.1(0.8)	552.4(0.8)	543.8(0.8)	543.8(0.8)	556.6(0.20)

1. INVER: code of the inverse method,
2. ALIAS: code of the alias method,
3. DISCS: code of Brown's method,
4. IMSL: employs the IMSL routine RNGDA, in which the alias method is implemented,
5. WGHTS1: code of the weighted sampling implementing technique I, and
6. WGHTS2: code of the weighted sampling implementing technique II.

Note that since the processing time of the equiprobable method is the same as WGHTS2, the equiprobable method is not included.

The following notations are for the vector codes.

1. INVR1: code of the inverse method, shown in Figure 2.2,
2. INVR2: code of the inverse method, shown in Figure 2.3,
3. ALIAS: code of the alias method,
4. DISCS: code of Brown's method,
5. WGHTS1: code of the weighted sampling implementing technique I, and
6. WGHTS2: code of the weighted sampling implementing technique II.

The processing time of the scalar codes for $n = 10$ is depicted in Table 3.5. It shows that the least processing time is for the WGHTS2 code, followed by the WGHTS1 code; then the processing time increases respectively for the IMSL, DISCS, INVER, and ALIAS codes. The processing time of WGHTS1 is slightly higher than that of WGHTS2, but it is still lower than the others.

Table 3.6 shows the processing time of the scalar codes for $n = 200$. In this table, the least processing time is still for WGHTS2 code, followed by those of WGHTS1, IMSL, DISCS, and ALIAS codes. The processing time of WGHTS1, WGHTS2, IMSL, DISCS and ALIAS codes for $n = 10$ and $n = 200$ are almost equal.

Table 3.7 describes the processing times of the vector codes for $n = 10$. The least processing time is for WGHTS2 code, followed by the WGHTS1 code; then the processing time increases respectively for DISCS, INVR2, ALIAS and INVR1 codes.

Table 3.8 demonstrates the processing times of the vector codes for $n = 200$. The least processing time is for WGHTS2 code, followed by those of WGHTS1, DISCS, ALIAS, INVR1 and INVR2 codes.

Table 3.5: Scalar processing time for $n = 10$, in milliseconds.

Samples	INVER	ALIAS	DISCS	IMSL	WGHTS1	WGHTS2
20,000	100.58	159.33	98.28	94.55	85.19	83.28
40,000	201.25	318.64	196.55	188.41	170.12	166.51
60,000	302.24	478.46	294.98	282.85	255.12	249.97
80,000	402.52	637.26	392.81	378.37	340.02	333.11
100,000	504.30	796.89	491.25	471.45	425.39	416.58

Table 3.6: Scalar processing time for $n = 200$, in milliseconds.

Samples	INVER	ALIAS	DISCS	IMSL	WGHTS1	WGHTS2
20,000	382.25	159.94	97.90	94.42	85.30	83.35
40,000	766.99	320.13	195.93	189.40	170.34	166.80
60,000	1151.52	479.60	294.13	283.49	255.46	250.45
80,000	1538.74	640.09	392.59	378.68	340.47	333.99
100,000	1920.76	798.24	490.12	473.07	425.93	416.79

Table 3.7: Vector processing time for $n = 10$, in milliseconds.

Samples	INVR1	INVR2	ALIAS	DISCS	WGHTS1	WGHTS2
20,000	34.77	18.23	20.83	10.98	8.42	7.02
40,000	69.63	36.38	41.65	21.83	16.81	13.93
60,000	104.49	54.72	62.46	32.74	25.28	20.90
80,000	139.17	72.75	83.28	43.65	33.53	27.89
100,000	174.38	90.94	104.40	54.53	42.07	34.80

Table 3.8: Vector processing time for $n = 200$, in milliseconds.

Samples	INVR1	INVR2	ALIAS	DISCS	WGHTS1	WGHTS2
20,000	190.68	283.67	21.06	11.00	8.53	7.16
40,000	381.60	566.98	42.03	21.93	17.03	14.27
60,000	573.15	850.28	62.95	32.89	25.60	21.41
80,000	763.00	1132.60	83.87	43.91	33.95	28.34
100,000	955.29	1418.00	104.99	54.77	42.58	35.41

3.4.3 Speedups

In this subsection, the minimum and maximum vectorization speedups of different vector codes are investigated. The objective is to examine the range of possible speedups, which can be achieved by different vector codes and the vector code of the weighted sampling method especially. The speedup of a vector code obtained relative to the scalar code having the least processing time is referred to as the minimum vectorization speedup, denoted by V_{min}. On the other hand, the maximum vectorization speedup (V_{max}) is obtained relative to the scalar code having the largest processing time.

Table 3.9: Minimum vectorization speedups (V_{min}) for $n = 10$.

Samples	INVR1	INVR2	ALIAS	DISCS	WGHTS1	WGHTS2
20,000	2.395	4.568	3.998	7.585	9.891	11.863
40,000	2.391	4.577	3.998	7.628	9.905	11.953
60,000	2.392	4.568	4.002	7.635	9.888	11.960
80,000	2.394	4.579	4.000	7.631	9.935	11.944
100,000	2.389	4.581	3.990	7.639	9.902	11.971

Table 3.9 shows the minimum speedups of the vector codes relative to the scalar WGHTS2 code, which requires the smallest scalar processing time. The WGHTS2 attains higher speedups than WGHTS1, as we have predicted.

Table 3.10: Maximum vectorization speedups (V_{max}) for $n = 10$.

Samples	INVR1	INVR2	ALIAS	DISCS	WGHTS1	WGHTS2
20,000	4.582	8.740	7.649	14.511	18.923	22.697
40,000	4.576	8.759	7.650	14.596	18.955	22.874
60,000	4.579	8.744	7.660	14.614	18.926	22.893
80,000	4.579	8.760	7.652	14.599	18.006	22.849
100,000	4.570	8.763	7.633	14.614	18.942	22.899

The maximum speedups of the vector codes (shown in Table 3.10) are obtained

relative to the processing time of the scalar ALIAS code, which exhibits the largest scalar execution time. Note that the maximum speedups are about twice of the corresponding minimum speedups since the scalar ALIAS's processing time is about twice of the scalar WGHTS2's processing time.

Table 3.11: Minimum vectorization speedups (V_{min}) for $n = 200$.

Samples	INVR1	INVR2	ALIAS	DISCS	WGHTS1	WGHTS2
20,000	0.437	0.294	3.958	7.577	9.771	11.641
40,000	0.437	0.294	3.969	7.606	9.794	11.689
60,000	0.437	0.295	3.979	7.615	9.783	11.698
80,000	0.438	0.295	3.982	7.606	9.838	11.785
100,000	0.436	0.294	3.970	7.610	9.788	11.770

Table 3.12: Maximum vectorization speedups (V_{max}) for $n = 200$.

Samples	INVR1	INVR2	ALIAS	DISCS	WGHTS1	WGHTS2
20,000	2.005	1.348	18.151	34.750	44.812	53.387
40,000	2.010	1.353	18.249	34.974	45.038	53.748
60,000	2.009	1.354	18.293	35.011	44.981	53.784
80,000	2.017	1.359	18.347	35.043	45.324	54.296
100,000	2.011	1.355	18.295	35.070	45.109	54.243

For $n = 200$, the minimum and maximum vectorization speedups are shown in Tables 3.11 and 3.12, respectively. The minimum speedups are obtained relative to the scalar WGHTS2's processing time, while the maximum speedups are relative to the scalar INVER's processing time.

3.4.4 Efficiency

In this subsection, we exhibit the efficiencies of different scalar codes relative to the scalar INVER code, and those of different vector codes relative to the vector INVR1 code. The efficiency calculation uses the sample variances, which can be obtained from the %FSDs shown in Table 3.3.

Table 3.13: Efficiencies of scalar codes relative to those of scalar INVER, for $n = 10$.

Samples	ALIAS	DISCS	IMSL	WGHTS1	WGHTS2
20,000	0.632	1.024	1.065	0.020	0.021
40,000	0.730	1.183	1.235	0.022	0.023
60,000	0.630	1.021	1.065	0.020	0.021
80,000	0.629	1.260	1.309	0.022	0.023
100,000	0.632	1.297	1.352	0.023	0.023

Table 3.14: Efficiencies of scalar codes relative to those of scalar INVER, for $n = 200$.

Samples	ALIAS	DISCS	IMSL	WGHTS1	WGHTS2
20,000	2.392	3.908	4.053	0.077	0.079
40,000	2.768	4.523	4.680	0.085	0.087
60,000	2.393	3.902	4.048	0.077	0.079
80,000	2.394	4.819	4.998	0.085	0.086
100,000	2.402	4.952	5.130	0.086	0.088

This subsection also reports *maximum relative efficiencies* (η_{max}) of different vector codes, which are obtained relative to the smallest efficiency of the scalar code. They demonstrate the possible maximum relative efficiencies one can achieve through vectorization and an appropriate sampling method.

Tables 3.13 and 3.14 show the efficiencies of different scalar codes relative to the scalar INVER code for $n = 10$ and $n = 200$, respectively. In this tables, the relative efficiencies of WGHTS1 and WGHTS2 are smaller than one. This means that the

Table 3.15: Efficiencies of vector codes relative to those of vector INVR1, for $n = 10$.

Samples	INVR2	ALIAS	DISCS	WGHTS1	WGHTS2
20,000	1.909	1.671	3.170	0.071	0.086
40,000	2.212	1.932	3.687	0.078	0.094
60,000	1.903	1.667	3.181	0.071	0.085
80,000	1.905	2.055	3.922	0.078	0.093
100,000	1.914	2.111	4.041	0.079	0.096

speedups attained by the weighted sampling codes cannot compensate the increase in the sample variances of their mean estimates.

Table 3.16: Efficiencies of vector codes relative to those of vector INVR1, for $n = 200$.

Samples	INVR2	ALIAS	DISCS	WGHTS1	WGHTS2
20,000	0.673	9.062	17.354	0.386	0.460
40,000	0.778	10.491	20.112	0.423	0.504
60,000	0.672	9.074	17.367	0.383	0.458
80,000	0.671	11.185	21.374	0.421	0.504
100,000	0.673	11.497	22.040	0.430	0.517

Tables 3.15 and 3.16 demonstrate the efficiencies of different vector codes relative to the vector INVR1 code for $n = 10$ and $n = 200$, respectively. Table 3.15 concludes that the speedups attained by WGHTS1 and WGHTS2 cannot compensate the increase in the sample variances of their mean estimates.

For $n = 10$, the maximum relative efficiencies of different vector codes are obtained relative to the corresponding efficiencies of scalar ALIAS code. Similarly, the maximum relative efficiencies of different vector codes are calculated relative to the corresponding efficiencies of scalar INVER code for $n = 200$.

Table 3.17: Maximum efficiencies (η_{max}) of vector codes relative to efficiencies of scalar ALIAS, for $n = 10$.

Samples	INVR1	INVR2	ALIAS	DISCS	WGHTS1	WGHTS2
20,000	2.893	5.522	4.833	9.171	0.206	0.248
40,000	2.890	6.392	5.583	10.655	0.226	0.273
60,000	2.893	5.504	4.822	11.200	0.204	0.247
80,000	2.892	5.510	5.943	11.343	0.225	0.270
100,000	2.892	5.537	6.104	11.686	0.229	0.277

Table 3.18: Maximum efficiencies (η_{max}) of vector codes relative to efficiencies of scalar INVER, for $n = 200$.

Samples	INVR1	INVR2	ALIAS	DISCS	WGHTS1	WGHTS2
20,000	2.005	1.349	18.167	34.790	0.774	0.923
40,000	2.010	1.563	21.087	40.423	0.850	1.014
60,000	2.009	1.350	18.230	34.891	0.769	0.920
80,000	2.017	1.353	22.557	43.105	0.849	1.017
100,000	2.011	1.352	23.117	44.314	0.864	1.039

3.4.5 Remarks

In this problem, the weighted sampling results in the least processing time for scalar as well as vector processing. It results in the largest %FSD for the mean estimate, but it attains the smallest %FSD for the variance estimate. Further, the weighted sampling entails the lowest efficiencies since the speedups achieved by the weighted sampling are lower than the increase in sample variances of its solutions.

3.5 Problem II

In this section, we examine the performance of different sampling methods. These methods are used to estimate the mean and variance of a probability distribution represented by Table 3.19. In Problem I, it was demonstrated that the weighted sampling resulted in large FSDs. In this problem however, the weighted sampling entails smaller %FSDs than the other methods.

Table 3.19: The probability table used in Problem II.

Mass points	7	10	11	19	31	52	86	144	240	400
Probabilities	.400	.240	.144	.086	.052	.031	.019	.011	.010	.007

$\mu = 20.26$, $\sigma^2 = 1888.082$

3.5.1 Estimation of Distribution Mean and Variance

Table 3.20 demonstrates that the mean estimates computed using various methods are unbiased, i.e. their values converge to the value of the distribution mean, which is equal to 20.06. Note that the weighted sampling results in the closest estimates and the smallest %FSDs.

In contrast with Problem I, the weighted sampling in this problem results in the smallest FSDs. The characteristics of Table 3.19 is different with Table 2.6. Table 3.19

Table 3.20: Mean estimates and their percentage FSDs.

Sample size	INVER $\bar{X}$(%FSD)	ALIAS $\bar{X}$(%FSD)	DISCS $\bar{X}$(%FSD)	IMSL $\bar{X}$(%FSD)	WGHTS $\bar{X}_w$(%FSD)
20,000	20.26(1.55)	19.67(1.52)	19.67(1.67)	19.68(1.52)	20.08(0.18)
40,000	20.23(1.09)	19.57(1.07)	19.57(1.18)	19.56(1.07)	20.06(0.13)
60,000	20.09(0.88)	19.82(0.88)	19.82(0.97)	19.81(0.88)	20.06(0.10)
80,000	20.11(0.76)	19.79(0.76)	19.79(0.84)	19.78(0.76)	20.06(0.09)
100,000	20.11(0.69)	19.78(0.68)	19.78(0.75)	19.78(0.68)	20.06(0.08)

shows that the trend of the probabilities from left to right is decreasing while the trend for the mass point values is increasing.

Table 3.21: Variance estimates and their percentage FSDs.

Sample size	INVER S^2(%FSD)	ALIAS S^2(%FSD)	DISCS S^2(%FSD)	IMSL S^2(%FSD)	WGHTS S_w^2(%FSD)
20,000	1972.41(4.81)	1786.62(5.02)	1786.62(5.02)	1782.70(5.02)	1889.84(1.15)
40,000	1947.02(3.41)	1763.48(3.54)	1763.48(3.54)	1758.33(3.54)	1886.43(0.81)
60,000	1888.18(2.81)	1829.04(2.84)	1829.04(2.84)	1823.35(2.85)	1889.60(0.66)
80,000	1893.32(2.43)	1828.79(2.47)	1828.79(2.47)	1822.72(2.47)	1892.52(0.57)
100,000	1907.98(2.18)	1806.76(2.21)	1806.76(2.21)	1801.96(2.21)	1886.47(0.51)

Table 3.21 demonstrates the estimates for the distribution variance obtained using the different sampling methods. This table shows that the weighted sampling method results in the best estimate; i.e. the estimated values are close to the value of the distribution variance, which is equal to 1888.082.

3.5.2 Processing Time

The number of comparisons required by the inverse method is affected by the probability distribution. Nevertheless, the other methods are not affected by the probability distribution. Therefore, the processing time required for sampling in Problem II is the same as those in Problem I, except the inverse method requires slightly higher time. Since the difference is of the order of microseconds which will not be seen in

milliseconds, there is no further need to show the processing time. For the same reason, the vectorization speedups are not discussed here.

3.5.3 Efficiency

This subsection demonstrates the efficiencies of different scalar codes relative to the scalar INVER code, and those of different vector codes relative to the vector INVR1 code. We report also the maximum relative efficiencies of different vector codes, obtained relative to the smallest efficiency of the scalar code.

Table 3.22: Efficiencies of scalar codes relative to those of scalar INVER, for $n = 10$.

Samples	ALIAS	DISCS	IMSL	WGHTS1	WGHTS2
20,000	0.696	0.935	1.172	89.124	91.168
40,000	0.700	0.934	1.186	84.582	86.416
60,000	0.649	0.866	1.099	92.018	93.913
80,000	0.652	0.866	1.100	84.837	86.597
100,000	0.674	0.898	1.138	88.630	90.505

Table 3.23: Efficiencies of scalar codes relative to those of scalar INVER, for $n = 200$.

Samples	ALIAS	DISCS	IMSL	WGHTS1	WGHTS2
20,000	2.637	3.568	4.462	338.274	346.188
40,000	2.657	3.569	4.495	321.936	328.768
60,000	2.467	3.311	4.178	348.751	357.120
80,000	2.482	3.313	4.200	323.885	330.169
100,000	2.561	3.429	4.321	337.144	344.537

The efficiencies of different scalar codes relative to the scalar INVER code for $n = 10$ and $n = 200$ are shown in Tables 3.22 and 3.23, respectively. These tables demonstrate that the relative efficiencies of WGHTS1 and WGHTS2 are much larger than one. This is due to the scalar speedups and the smaller sample variance resulted by the weighted sampling.

Table 3.24: Efficiencies of vector codes relative to those of vector INVR1, for $n = 10$.

Samples	INVR2	ALIAS	DISCS	WGHTS1	WGHTS2
20,000	2.104	1.526	3.490	311.718	373.884
40,000	2.122	1.524	3.541	296.159	357.389
60,000	1.962	1.415	3.282	321.041	388.322
80,000	1.975	1.413	3.296	297.452	357.603
100,000	2.041	1.461	3.403	309.888	374.626

Table 3.25: Efficiencies of vector codes relative to those of vector INVR1, for $n = 200$.

Samples	INVR2	ALIAS	DISCS	WGHTS1	WGHTS2
20,000	0.742	8.275	19.104	1687.431	2010.305
40,000	0.746	8.278	19.316	1602.101	1911.968
60,000	0.693	7.699	17.922	1738.968	2079.290
80,000	0.696	7.690	17.961	1610.606	1929.431
100,000	0.717	7.960	18.563	1677.297	2016.925

Tables 3.24 and 3.25 demonstrate the efficiencies of different vector codes relative to the vector INVR1 code for $n = 10$ and $n = 200$, respectively. The high efficiencies achieved by the WGHTS1 and WGHTS2 are due to their vector speedups and small sample variances.

Table 3.26: Maximum efficiencies (η_{max}) of vector codes relative to efficiencies of scalar ALIAS, for $n = 10$.

Samples	INVR1	INVR2	ALIAS	DISCS	WGHTS1	WGHTS2
20,000	2.893	6.087	4.413	10.095	901.715	1081.544
40,000	2.890	6.134	4.406	10.233	855.981	1032.954
60,000	2.893	5.675	4.092	9.494	928.620	1123.231
80,000	2.892	5.713	4.086	9.532	860.317	1034.293
100,000	2.892	5.902	4.226	9.842	896.184	1083.403

For $n = 10$, the maximum relative efficiencies of different vector codes are obtained relative to the efficiencies of scalar ALIAS code. The maximum relative efficiencies of WGHTS1 and WGHTS2 are at most 928.6 and 1123.2 respectively. Similarly for $n = 200$, the maximum relative efficiencies of different vector codes are calculated relative to the corresponding efficiencies of scalar INVER. Table 3.27 reports that the weighted sampling methods attain excellent relative efficiencies up to 4177.5.

Table 3.27: Maximum efficiencies (η_{max}) of vector codes relative to efficiencies of scalar INVER, for $n = 200$.

Samples	INVR1	INVR2	ALIAS	DISCS	WGHTS1	WGHTS2
20,000	2.005	1.487	16.588	38.297	3382.738	4029.993
40,000	2.010	1.500	16.639	38.823	3220.114	3842.925
60,000	2.009	1.391	15.469	36.008	3493.775	4177.517
80,000	2.017	1.403	15.508	36.222	3248.104	3891.077
100,000	2.011	1.442	16.006	37.323	3372.469	4055.344

3.5.4 Remarks

The weighted sampling results in the smallest %FSDs for both distribution mean and variance estimates. In contrast with the previous problem, the weighted sampling codes result in the highest efficiencies in Problem II, due to the vectorization speedup achieved and the decrease in sample variance of the solution.

3.6 Conclusions

In this chapter, we have proposed the weighted sampling method for generating samples from arbitrary discrete distributions, which are represented by probability tables. The proposed method has been examined for several aspects, including mean and variace estimators, processing time, speedup, efficiency.

The weighted sampling method can be implemented in two different techniques. The weighted sampling method implementing the first technique can make direct use of the original probability tables without any alterations, while that implementing the second technique requires a generation table for constructing samples. Therefore, the first approach can be incorporated easily into large simulation codes, while the second approach is faster since fewer calculations are required.

The scalar and vector codes of the proposed weighted sampling method (implemented in both ways) are simpler than those of the other methods. The weighted sampling method requires the least processing time in scalar as well as in vector processing, compared to other methods. Moreover, this sampling method is also amenable to vectorization without special effort since there is no form of data dependency, which inhibits vectorization of the code.

This method may result in larger or smaller fractional standard deviation of the estimated value for the distribution mean than those of the other methods. In Problem I the weighted sampling results in larger FSDs, while in Problem II it entails smaller FSDs of the mean estimate. However, it should be emphasized that this method entails the smallest FSDs of the estimated values for the distribution variances in both problems.

The weighted sampling method therefore satisfies the criteria for a good sampling method, as stated by Devroye [24, page 8], namely speed, length of the compiled code, portability, simplicity, and readability.

Chapter 4

Large Sparse Linear Systems

4.1 Introduction

In this chapter[1], we examine the applications of Monte Carlo methods for solving large sparse linear equations using absorbing and ergodic Markov chains. The inverse method, Brown's and the weighted sampling methods are implemented for determining the state transitions in Markov chains. It is shown that Brown's method and the weighted sampling method can reduce the time complexities of the Monte Carlo methods from $\Theta(n^3)$ to $\Theta(n^2)$ for solving a system of n linear equations. Further, the weighted sampling results in simpler code and faster execution on scalar as well as vector machines compared to the other sampling methods, as demonstrated with computational results on IBM 3090-180 machine with a vector facility.

This chapter is organized as follows. Section 4.2 describes the background and motivations for this chapter. Monte Carlo methods for solving a system of linear equations are briefly reviewed in Section 4.4. Section 4.4 discusses the applications of the the weighted sampling method for determining state transitions in these Monte Carlo methods. The time complexity analyses of Monte Carlo solutions are carried out in Section 4.5, while the vectorization of Monte Carlo solutions is elaborated in

[1] An earlier version of this chapter is published in [84].

Section 4.6. Sections 4.7 presents the sparse data structures for Monte Carlo solutions. Computational results are discussed in Sections 4.8. Finally, concluding remarks are given in Section 4.9.

4.2 Background and Motivations

Von Neumann and Ulam proposed a fundamental Monte Carlo method for inversion of a matrix (this method was published by Forsythe and Leibler [28]). Later, Wasow [101] also proposed a different method for the inversion and subsequently compared its efficiency with the former method. Obviously, these methods can be used to solve a system of linear equations. Similar methods can also be used to solve a large system of integral and/or differential equations (see [32]), which cannot be solved efficiently by the classical numerical analysis [77].

The principal advantage of the Monte Carlo method for solving a system of linear equations is that, unlike the direct or iterative numerical methods, the solution of a variable can be estimated independently of the solutions of other variables. This fact becomes very relevant when the number of variables is large and the estimates of only a small number of variables are required, and when implementations on vector and parallel computers are considered. Further, it necessitates simple data structure and nonzero elements can be taken care of easily. Consequently, the computation occupies only relatively small memory. Thus, the Monte Carlo method is attractive for sparse matrices since the direct and iterative numerical methods for such matrices require complicated data structures, which often result into inefficient algorithms for vector and/or parallel processing.

The Monte Carlo method for solving a system of linear equations possesses large intrinsic parallelism (see [6]). Moreover, the use of novel sampling methods, such as the one proposed in Chapter 3 can further reduce the computing time on vector and parallel computers. Consequently, the Monte Carlo solutions are suitable for vector

and parallel processing.

There are two Monte Carlo methods for solving a system of linear equations. The first method simulates a discrete-time absorbing Markov chain, while the second method simulates a discrete-time ergodic Markov chain [77]. In these methods, the determination of state transitions during random walks can be viewed as generation of samples from a transition probability matrix, which is a two-dimensional probability table. For this purpose, the inverse method, discussed in Chapter 2, is often used [5]. Obviously, the sampling method affects the time complexity of the Monte Carlo solutions, and the complexity turns out to be $\Theta(n^3)$ [5]. $\Theta(f(n))$ means that the time complexity can be expressed using a function $f(n)$ bounded by constant factors both above and below [89].

In this context, we incorporate the weighted sampling method, discussed in Chapter 3, into the two Monte Carlo methods for solving a sparse system of linear equations. It is demonstrated that the Monte Carlo solutions using this sampling method require simpler scalar as well as vector codes, and entail valid results. Further, we show that the time complexities of the Monte Carlo methods are reduced from $\Theta(n^3)$ to $\Theta(n^2)$ and we obtain faster execution on scalar as well as vector computers.

4.3 Monte Carlo Solutions

Let us begin by considering a nonsingular system of equations defined by[2]

$$\sum_{s=1}^{n} \mathbf{A}_{rs}\mathbf{x}_s = \mathbf{b}_r, \text{ where } r \in \{1, 2, \cdots, n\}, \tag{4.1}$$

which can also be written in the matrix equation $\mathbf{Ax}=\mathbf{b}$. This equation has a unique solution given by $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$. By introducing $\mathbf{H} = \mathbf{I} - \mathbf{A}$, where $\mathbf{I}$ is an identity matrix, Equation (4.1) can be rewritten as

$$\mathbf{x} = \mathbf{Hx} + \mathbf{b}. \tag{4.2}$$

[2]The mathematical exposition in this section is based on Halton [32].

When the spectral radius of $\mathbf{H}$ is less than one ($\rho(\mathbf{H}) < 1$), the Neumann series expansion

$$\begin{aligned} \mathbf{x} &= (\mathbf{I}-\mathbf{H})^{-1}\mathbf{b} \\ &= (\mathbf{I}+\mathbf{H}+\mathbf{H}^2+\cdots+\mathbf{H}^m+\cdots)\mathbf{b} \\ &= \sum_{m=0}^{\infty}\mathbf{H}^m\mathbf{b} \end{aligned} \tag{4.3}$$

is absolutely convergent.

4.3.1 Solution Using An Absorbing Markov Chain

The Monte Carlo method, suggested by Forsythe and Leibler [28], employs an absorbing Markov chain to solve a system of linear equations. In this method, the procedure for obtaining the estimate of an unknown is as follows. A set of random walks is defined on the augmented index set, $S = \{0, 1, ..., n\}$, where index 0 represents the termination index for the random walk. Let $\mathbf{P}$ denote a $(n+1) \times (n+1)$ transition probability matrix, and $\mathbf{P}_{ij}$ denote the probability of transition from index i to index j, with $\mathbf{P}_{ij} \neq 0$ unless $\mathbf{H}_{ij} = 0$. Then, corresponding to random walk $\Gamma = \{r, s_1, \cdots, s_m, 0\}$, with $r, s_1, \cdots, s_m \neq 0$, the primary estimator for an unknown $\mathbf{x}_r$ is given as

$$X_r(\Gamma) = \mathbf{Z}_{r,s_1}\mathbf{Z}_{s_1,s_2}\cdots\mathbf{Z}_{s_{m-1},s_m}\mathbf{Z}_{s_m,0}, \tag{4.4}$$

where for $j \neq 0$, $\mathbf{Z}_{ij} = \mathbf{H}_{ij}/\mathbf{P}_{ij}$ and for $j = 0$, $\mathbf{Z}_{i,0} = \mathbf{b}_i/\mathbf{P}_{i0}$. Note that the Monte Carlo estimate of each of the unknowns in Equation (4.3) can be obtained independently of the remaining unknowns.

The determination of state transitions during random walks can be viewed as sampling from the transition probability matrix. This probability matrix can be considered as a set of one-dimensional probability tables; that is, each row of the matrix is a one-dimensional probability table. The inverse method [56] is usually used for this sampling process. An algorithm using this method is given in Figure 4.1. In

this algorithm, s_1 denotes a current state, s_2 the next state, and $\mathbf{C}$ denotes a matrix for the cumulative distribution of the transition probability matrix.

```
PROCEDURE Statei(s1, s2)
BEGIN
    Generate a random number ξ uniform on (0,1)
    s2 = 0
    WHILE ( ξ > C_{s1,s2}) DO
        s2 = s2 + 1
    END DO
END
```

Figure 4.1: An algorithm for the state determination using the inverse method.

The secondary estimator for $\mathbf{x}_r$ is then expressed by the arithmetic mean of k primary estimates as follows

$$Y_r = \frac{1}{k} \sum_{i=1}^{k} X_r(\Gamma_i). \tag{4.5}$$

The expected value of Y_r is $\mathbf{x}_r$ as $k \to \infty$.

A sequential algorithm for the Monte Carlo solution employing an absorbing Markov chain is given in Figure 4.2. This algorithm calls procedure Statei, described in Figure 4.1, to determine the state transitions. Note that for matrix $\mathbf{H}$ of size $n \times n$, the matrices $\mathbf{P}$ and $\mathbf{C}$ are of size $(n+1) \times (n+1)$ since column 0 and row 0 are added for the absorbing state.

```
BEGIN
    FOR i = 1 TO n STEP 1 DO
        Y_i = 0
        FOR j = 1 TO k STEP 1 DO
            Z = 1; s_1 = i
            WHILE s_1 ≠ 0 DO
                CALL Statei(s_1, s_2)
                IF (s_2 ≠ 0) THEN Z = Z * H_{s_1,s_2} / P_{s_1,s_2}
                IF (s_2 = 0) THEN Z = Z * b_{s_1} / P_{s_1,0}
                s_1 = s_2
            END DO
            Y_i = Y_i + Z
        END DO
        Y_i = Y_i / k
    END DO
END
```

Figure 4.2: A sequential Monte Carlo algorithm employing an absorbing Markov chain.

4.3.2 Solution Using An Ergodic Markov Chain

An application of an ergodic Markov chain for solving a system of linear equations was proposed by Wasow [101]. Wasow introduced a different estimator and compared its variance to that of the previous method.

```
BEGIN
    FOR i = 1 TO n STEP 1 DO
        Y_i = 0
        FOR j = 1 TO k STEP 1 DO
            Z = 1; s_1 = i; T = b_i
            FOR k = 1 TO m STEP 1 DO
                CALL Statei(s_1, s_2)
                Z = Z * H_{s_1,s_2} / P_{s_1,s_2}
                T = T + Z * b_{s_1}
                s_1 = s_2
            END DO
            Y_i = Y_i + T
        END DO
        Y_i = Y_i / k
    END DO
END
```

Figure 4.3: A sequential Monte Carlo algorithm employing an ergodic Markov chain.

In this method, scoring for the primary estimator of an unknown is carried out in every random walk step. For a random walk from an ergodic Markov chain $\Lambda = \{r, s_1, \cdots, s_{m-1}, s_m\}$, the primary estimator of an unknown $\mathbf{x}_r$ is given by

$$X_r(\Lambda) = \mathbf{b}_r + \mathbf{b}_{s_1}\mathbf{Z}_{r,s_1} + \mathbf{b}_{s_2}\mathbf{Z}_{r,s_1}\mathbf{Z}_{s_1,s_2} + \cdots + \mathbf{b}_{s_m}\mathbf{Z}_{r,s_1}\mathbf{Z}_{s_1,s_2}\cdots\mathbf{Z}_{s_{m-1},s_m}, \qquad (4.6)$$

where $\mathbf{Z}_{ij} = \mathbf{H}_{ij}/\mathbf{P}_{ij}$. Wasow confirmed that the variance of this estimator is smaller than that of the estimator given in Equations (4.4) when the absorption probabilities in the transition probability matrix of an absorbing Markov chain ($\mathbf{P}_{i0}$ for $i =$

$1, 2, \cdots, n$) are relatively small. Further comparison of their variances is discussed by Edmundson [27].

The sequential algorithm for the Monte Carlo method employing an ergodic Markov chain is shown in Figure 4.3. This algorithm calls procedure Statei, described in Figure 4.1, to determine the state transitions. For this method, matrices $\mathbf{P}$ and $\mathbf{C}$ have the same size as matrix $\mathbf{H}$, which is $(n \times n)$.

4.4 The Weighted Sampling Method For State Transitions

The transition probability matrix for a Markov chain is a set of one-dimensional probability tables. In Chapter 3, the weighted sampling method has been used to sample from a one-dimensional probability table. Here, we extend the idea and apply the weighted sampling method to sample from a transition probability matrix for determining state transitions in the Monte Carlo methods for solving a system of linear equations.

Assume that s_1 is a current state, and it is located in the i-th row of the transition probability matrix. The probabilities are $\{\mathbf{P}_{ij}, j = 1, 2, \cdots, \mathrm{l}_i\}$, where l_i is the number of nonzero probabilities in the i-th row. Then, we use the weighted sampling method to select the next state, s_2, according to a discrete uniform distribution in the interval $[1, \mathrm{l}_i]$. Similar to the case of a one-dimensional probability table (discussed in Chapter 3), the primary estimator has to be subsequently multiplied by the adjustment factor $\mathbf{P}_{ij} \times \mathrm{l}_i$, where j is the selected state number.

For the Monte Carlo method employing an absorbing Markov chain, we modify the primary estimator in Equation (4.4) to the one in the following theorem.

THEOREM 3 *In the Monte Carlo method employing an absorbing Markov chain, when the weighted sampling is used to carry out random walk* $\tilde{\Gamma} = \{r, s_1, \cdots, s_m, 0\}$,

with $r, s_1, \cdots, s_m \neq 0$, *the primary estimator for an unknown* $\mathbf{x}_r$ *is given as*

$$\tilde{X}_r(\tilde{\Gamma}) = \tilde{\mathbf{Z}}_{r,s_1}\tilde{\mathbf{Z}}_{s_1,s_2}\cdots\tilde{\mathbf{Z}}_{s_{m-1},s_m}\tilde{\mathbf{Z}}_{s_m,0}, \tag{4.7}$$

for $j \neq 0, \tilde{\mathbf{Z}}_{ij} = \mathbf{Z}_{ij}[\mathbf{P}_{ij} \times \mathbf{l}_i]$ *and for* $j = 0, \tilde{\mathbf{Z}}_{i,0} = \mathbf{Z}_{i0}[\mathbf{P}_{i0} \times \mathbf{l}_i]$, *where the quantities in the square brackets represent the adjustment factors.*

Proof.

According to Equation (4.4), $\mathbf{Z}_{ij} = \mathbf{H}_{ij}/\mathbf{P}_{ij}$ for $j \neq 0$, and $\mathbf{Z}_{i,0} = \mathbf{b}_i/\mathbf{P}_{i0}$ for $j = 0$. Therefore, $\tilde{\mathbf{Z}}_{ij} = \mathbf{H}_{ij}\mathbf{l}_i$ for $j \neq 0$, and $\tilde{\mathbf{Z}}_{i,0} = \mathbf{b}_i\mathbf{l}_i$ for $j = 0$. This means that the random walks are carried out based on a uniform transition probability matrix, and the probability for each row is $1/\mathbf{l}_i$ for $i = 1, 2, \cdots, n$. Thus, the estimator is unbiased. □

Similarly, for the Monte Carlo method employing an ergodic Markov chain the primary estimator is modified as follows.

THEOREM 4 *In the Monte Carlo method employing an ergodic Markov chain, when the weighted sampling is used to carry out random walk* $\tilde{\Lambda} = \{r, s_1, \cdots, s_{m-1}, s_m\}$, *the primary estimator for an unknown* $\mathbf{x}_r$ *is given as*

$$\tilde{X}_r(\tilde{\Lambda}) = \mathbf{b}_r + \mathbf{b}_{s_1}\tilde{\mathbf{Z}}_{r,s_1} + \mathbf{b}_{s_2}\tilde{\mathbf{Z}}_{r,s_1}\tilde{\mathbf{Z}}_{s_1,s_2} + \cdots + \mathbf{b}_{s_m}\tilde{\mathbf{Z}}_{r,s_1}\tilde{\mathbf{Z}}_{s_1,s_2}\cdots\tilde{\mathbf{Z}}_{s_{m-1},s_m}, \tag{4.8}$$

where $\tilde{\mathbf{Z}}_{ij} = \mathbf{Z}_{ij}[\mathbf{P}_{ij} \times \mathbf{l}_i]$.

Proof.

Equation (4.6) shows that $\mathbf{Z}_{ij} = \mathbf{H}_{ij}/\mathbf{P}_{ij}$. Therefore, $\tilde{\mathbf{Z}}_{ij} = \mathbf{H}_{ij}\mathbf{l}_i$. It means that the random walks are carried out based on a uniform transition probability matrix, and the probability for each row is $1/\mathbf{l}_i$ for $i = 1, 2, \cdots, n$. Thus, the estimator is unbiased. □

```
PROCEDURE Statew(s1, s2)
BEGIN
    Generate a random number ξ uniform on (0,1)
    s2 = ⌊ξ × (l_s1 + 1)⌋
END
```

Figure 4.4: An algorithm for the state determination using the weighted sampling method.

The modified estimators described by Equations (4.7) and (4.8) can be easily incorporated into the Monte Carlo algorithms shown in Figures 4.2 and 4.3, respectively. Figure 4.4 gives the algorithm for determining the next state (s_2) based on the weighted sampling method.

4.5 Time Complexity Analysis

This section analyzes the time complexities of the Monte Carlo algorithms for solving a system of linear equations[3]. The complexities here are expressed in terms of the number of comparisons and the number of state transitions for obtaining a primary estimator. The analysis is carried out in two steps. First, we obtain the number of comparisons required to determine a next state for each state transition. Second, we estimate the number of state transitions in a random walk required for obtaining a primary estimator (in the next discussion, this is referred to as random walk length). Then, the complexity of the Monte Carlo algorithm can be calculated by multiplying the two results.

[3]The discussion in this section is based on that given in Bhavsar [5, pp. 98–113].

4.5.1 The State Determination

The Inverse Method

When the inverse method is used to determine the state transitions (see Figure 4.1), the algorithm complexity is as follows.

Let V_{s_1} denote a random variable representing the number of comparisons required to determine the next state s_2 from a current state s_1. We assume that the number of nonzero probabilities for each row of matrix $\mathbf{P}$ is n; i.e. $\mathbf{l}_i = n$, for $i = 1, 2, \cdots, n$. With this assumption, the results are overestimated since $\mathbf{l}_i \leq n$ is the case for a sparse system of linear equations. The expectation and the variance of the number of comparisons can be calculated based on the following probability theory. The expected number of comparisons ($E[V_{s_1}]$) is given by

$$E[V_{s_1}] = \sum_{i=1}^{n+1} i\mathbf{P}_{s_1,(i-1)}, \tag{4.9}$$

while the variance is given by

$$Var[V_{s_1}] = \sum_{i=1}^{n+1} \mathbf{P}_{s_1,(i-1)} \left(i - E[V_{s_1}]\right)^2. \tag{4.10}$$

Equation (4.10) can be simplified as

$$Var[V_{s_1}] = \sum_{i=1}^{n+1} i^2\mathbf{P}_{s_1,(i-1)} - E[V_{s_1}]^2. \tag{4.11}$$

Thus, the inverse method requires $\Theta(n)$ number of comparisons to determine a next state. The dependence of the above equations on n is not clear. The following example illustrates the dependence of the expected value and its variance on n.

Example: Consider that $P_{s_1,j}, j = 0, 1, \cdots, n,$ are uniformly distributed; i.e. $P_{s_1,j} = \frac{1}{n+1}$ for $j = 0, 1, \cdots, n$. Then the expected number of comparisons is given by

$$\begin{aligned} E[V_{s_1}] &= \frac{1}{n+1} \sum_{i=1}^{n+1} i, \\ &= \frac{n+2}{2}, \end{aligned} \tag{4.12}$$

(4.13)

and the associated variance is

$$Var[V_{s_1}] = \frac{1}{n+1}\sum_{i=1}^{n+1} i^2 - \left(\frac{n+2}{2}\right)^2. \tag{4.14}$$

Since $\sum_{i=1}^{n+1} i^2 = \frac{n+1}{3}(n+\frac{3}{2})(n+2)$ (see [51, p. 55]), the Equation (4.14) can be simplified as

$$\begin{aligned} Var[V_{s_1}] &= \frac{1}{n+1}\left(\frac{n+1}{3}(n+\frac{3}{2})(n+2)\right) - \left(\frac{n+2}{2}\right)^2, \\ &= \frac{n(n+12)}{12}. \quad \diamond \end{aligned} \tag{4.15}$$

Thus, it is seen that the expected number of comparisons and the associated variance depend on n.

The Weighted Sampling Method

When the weighted sampling method is used for the state determination (see Figure 4.4), it does not require any comparisons and therefore it requires only a constant time. Thus, in this case the time complexity is *independent* of the matrix size.

4.5.2 The Monte Carlo Algorithms

This subsection discusses the time complexities of the Monte Carlo algorithms for solving linear equations by employing ergodic and absorbing Markov chains.

Algorithm Using An Ergodic Markov Chain

In the algorithm using an ergodic Markov chain, the random walk length for each sample is predetermined. The random walks for all samples are given to be the same. For solving n number of linear equations using k number of samples for each of the unknowns, if we determine that the random walk length is equal to n, the total number of state transitions equals kn^2.

When the inverse method is used for determining state transitions, each state transition requires $\Theta(n)$ number of comparisons. Thus, the time complexity of the Monte Carlo algorithm is $\Theta(n^3)$.

However, when the weighted sampling method is used to determine state transitions, no comparisons are required. Consequently, the time complexity of the Monte Carlo algorithm reduces to $\Theta(n^2)$.

Algorithm Using An Absorbing Markov Chain

In this case, the random walk length (number of state transitions) for each sample is a random variable. The expectation of this random variable is found out as follows.

The substochastic matrix $\mathbf{Q}$, corresponding to the transition probability matrix $\mathbf{P}$, is given as [51]

$$\mathbf{Q} = \mathbf{P}_{ij}, \quad i, j \neq 0, \tag{4.16}$$

and the fundamental matrix $\mathbf{F}$ is given by

$$\mathbf{F} = (\mathbf{I} - \mathbf{Q})^{-1}, \tag{4.17}$$

where $\mathbf{I}$ is the identity matrix.

In estimating an unknown x_i, for each sample the length of random walks from state i to absorption is a discrete random variable with the mean is given as [51]

$$\mu_i = \sum_{j=1}^{n} \mathbf{F}_{ij}. \tag{4.18}$$

Note that an element $\mathbf{F}_{ij}$ of the fundamental matrix $\mathbf{F}$ describes the total number of walks visiting state j, if the chain started from state i, until it reaches an absorbing state.

In order to estimate n unknowns, the expectation of the total number of state transitions is expressed by

$$E[W] = k \sum_{i=1}^{n} \mu_i, \tag{4.19}$$

where k is the number of samples for each unknown. This equation is $\Theta(n^2)$. Since the dependence of Equation (4.19) on n is not clear, we use the following example to clarify it.

Example: Consider the transition probability matrix $\mathbf{P}$ with size $(n+1)\times(n+1)$ as follows.

$$\mathbf{P} = \begin{pmatrix} 1 & 0 & 0 & . & . & . & 0 \\ a & a & a & . & . & . & a \\ a & a & a & . & . & . & a \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ a & a & a & . & . & . & a \end{pmatrix} \tag{4.20}$$

where $a = \frac{1}{n+1}$.

The substochastic matrix $\mathbf{Q}$ of size $(n \times n)$ is given by the lower partition of matrix $\mathbf{P}$. According to Equation (4.17), the fundamental matrix $\mathbf{F}$ will be

$$\mathbf{F} = \mathbf{I} + \frac{a}{1-na}\mathbf{E}, \tag{4.21}$$

where $\mathbf{E}$ is a $(n \times n)$ matrix with all the elements equal to 1.

When the walk is started from state i, the expected number of state transitions, calculated using Equation (4.18), is given by

$$\begin{aligned} \mu_i &= 1 + n\frac{\frac{1}{n+1}}{1-\frac{n}{n+1}} \\ &= n+1. \end{aligned} \tag{4.22}$$

According to Equation (4.19), the expected number of state transitions for estimating n unknowns will be

$$E[W] = kn(n+1). \quad \diamond \tag{4.23}$$

It is seen that the expected number of state transitions depends on n.

When the inverse method is used for determining state transitions, each state transition requires $\Theta(n)$ number of comparisons. Thus, the time complexity of the Monte Carlo algorithm is $\Theta(n^3)$.

In contrast with the inverse method, the weighted sampling method does not require any comparisons for determining state transitions. Thus, the time complexity of the Monte Carlo algorithm is only $\Theta(n^2)$.

In conclusion, we have shown that by incorporating the weighted sampling method for determining the state transitions, the time complexity of the Monte Carlo algorithms can be reduced from $\Theta(n^3)$ to $\Theta(n^2)$. Since Brown's method does not need any comparisons and requires a constant time for determining state transitions, the time complexity of the Monte Carlo algorithms incorporating Brown's method is the same as that of Monte Carlo algorithms implementing the weighted sampling.

Note that Gauss elimination method (a direct method) has the time complexity of $\Theta(n^3)$ [55], while that of the Gauss-Seidel iteration method is $\Theta(n^2)$ [98].

4.6 Vectorization of The Monte Carlo Algorithms

4.6.1 Solution Using An Absorbing Markov Chain

In the scalar Monte Carlo algorithm as shown in Figure 4.2, a random walk is carried out at a time. The chain is tracked from birth to absorption. In the vectorized Monte Carlo algorithm, however, a set of random walks are carried out concurrently. For this purpose, a stack (a set of vectors) is utilized to hold the attribute values of random walks [80]. Since the random walk length is a random variable, the stack may contain absorbed and nonabsorbed walks. When the stack contains relatively many absorbed walks, then the utilization of vector processing hardware decreases. Therefore, a gather process is required to remove absorbed walks and gather the nonabsorbed

walks. The gather process should be done in such a way that the tradeoff between the vector processing efficiency and the overhead time is optimal.

The description of Figure 4.5 is as follows. Line 1 is the loop for estimating n unknowns. Initially, $\nu = k$. Line 6 shows the loop for random walks, and limits their length to at most m. For each random walk, lines 8–13 show that ν number of samples are carried out concurrently in vector processing. Line 9 shows that the weighted sampling is used for determining state transitions. The `if`-statements are used in lines 10–11 to select particular walks for the primary estimate calculation. The secondary estimate calculation of the absorbed walks is done in lines 14–16. Then, the gather process is carried out in lines 18–23. Line 24 shows that the value of ν is updated; i.e. the stack contains only nonabsorbed walks. This algorithm carries out the gather process in every transition of random walk since it results in minimum processing time. Finally, line 26 describes that the average value of the secondary estimate is calculated according to the number of absorbed walks $(k - \nu)$; i.e. all random walks of k samples do not necessarily reach the absorbing state due to the limitation of random walk length.

When the inverse method is used in the Monte Carlo code, the innermost loop will be the loop for implementing the inverse method; this loop replaces the weighted sampling given in line 9. With the existence of this loop, the larger loop (lines 8–13) is not vectorizable due to the `go to`-statement in the innermost loop.

Brown's method requires three lines of codes (see Figure 2.9 (a)) to replace line 9. These codes do not contain a loop. Therefore, using the weighted sampling method or Brown's method the vectorization will be achieved for the larger segment/loop of the program. Since k is usually much larger than n, the vectorization of the larger loop results in less processing time than that of the innermost loop. Thus, the weighted sampling method enhances the vectorizability of the code, and reduces the processing time.

```
BEGIN
1.    FOR i = 1 TO n STEP 1 DO /* Estimate n unknowns */
2.        Y_i = 0; ν = k
3.        FOR j = 1 TO ν STEP 1 DO
4.            Z_j = 1; s_{1_j} = i /* Initialization */
5.        END DO
6.        FOR τ = 1 TO m STEP 1 DO /* Random walk loop */
7.            Generate ν random numbers ξ_j, j = 1, ··· , ν
8.            FOR j = 1 TO ν STEP 1 DO /* Carry out ν samples */
9.                s_{2_j} = ⌊ξ_j × l_{s_{1_j}}⌋ + 1 /* The weighted sampling */
10.               IF (s_{2_j} ≠ 0) THEN Z_j = Z_j * H_{s_{1_j},s_{2_j}} * l_{s_{1_j}} /* Scoring */
11.               IF (s_{2_j} = 0) THEN Z_j = Z_j * b_{s_{1_j}} * l_{s_{1_j}} /* Scoring */
12.               s_{1_j} = s_{2_j}
13.           END DO
14.           FOR j = 1 TO ν STEP 1 DO /* Calculate the secondary estimate */
15.               IF (s_{2_j} = 0) THEN Y_i = Y_i + Z_j
16.           END DO
17.           λ=0
18.           FOR j = 1 TO ν STEP 1 DO
19.               IF (s_{2_j} ≠ 0) THEN /* Gather nonabsorbed walks */
20.                   λ = λ + 1
21.                   Z_λ = Z_{s_{2_j}}
22.               END IF
23.           END DO
24.           ν = λ
25.       END DO
26.       Y_i = Y_i/(k − ν)
27.   END DO
END
```

Figure 4.5: A vectorized Monte Carlo algorithm employing an absorbing Markov chain and the weighted sampling.

4.6.2 Solution Using An Ergodic Markov Chain

Similar to the vectorized algorithm of the solution using an absorbing Markov chain, the vectorized algorithm here also utilizes the stack processing. In this algorithm, the random walk length is predetermined; i.e. all random walks of k samples have the same length. This fact eliminates the use of `if`-statements in the primary and secondary estimate calculation, see lines 8–13 and and lines 14–16, respectively. Since the termination of random walks is the same for all samples, the gather process is not required. Note that the average value of the secondary estimate is calculated according to k samples (see line 18), since all random walks reach termination.

Thus, the vectorization of this algorithm is more efficient than that of the Monte Carlo algorithm using an absorbing Markov chain. The vectorized Monte Carlo algorithm using an ergodic Markov chain and incorporating the weighted sampling is shown in Figure 4.6.

4.7 Sparse Data Structures

In this section, we describe the data structures for handling the sparse matrices in the Monte Carlo solutions of linear equations. These data structures basically utilize indirect addressing of arrays, which are similar to those described in [73]. They are suitable for handling sparse matrices with regular or irregular pattern of nonzero locations, and do not inhibit the vectorization of the Monte Carlo codes.

A sparse vector may be stored in a full length vector. It is often used because of its simplicity even though it results into wastage of storage space. In order to economize the space, we can use a real valued array to hold only the nonzero elements of a sparse vector, and an integer array to store the indices of these elements.

```
BEGIN
1.    FOR i = 1 TO n STEP 1 DO /* Estimate n unknowns */
2.        Y_i = 0
3.        FOR j = 1 TO k STEP 1 DO
4.            Z_j = 1; s_{1_j} = i; T_j = b_i /* Initialization */
5.        END DO
6.        FOR τ = 1 TO m STEP 1 DO /* Random walk loop */
7.            Generate k random numbers ξ_j, j = 1, ..., k
8.            FOR j = 1 TO k STEP 1 DO /* Carry out k samples */
9.                s_{2_j} = ⌊ξ_j × l_{s_{1_j}}⌋ + 1 /* The weighted sampling */
10.               Z_j = Z_j * H_{s_{1_j}, s_{2_j}} * l_{s_{1_j}} /* Scoring */
11.               T_j = T_j + Z_j * b_{s_{2_j}} /* Scoring */
12.               s_{1_j} = s_{2_j}
13.           END DO
14.           FOR j = 1 TO k STEP 1 DO /* Calculate the secondary estimate */
15.               Y_i = Y_i + T_j
16.           END DO
17.       END DO
18.       Y_i = Y_i/k
19.   END DO
END
```

Figure 4.6: A vectorized Monte Carlo algorithm employing an ergodic Markov chain and the weighted sampling.

Further, a sparse matrix can be regarded as a collection of sparse vectors wherein each row of the matrix is considered as a sparse vector. For compressing this sparse matrix, we use a dense matrix to hold the nonzero elements, and an integer matrix to store the corresponding indices of locations. Let us consider the following example to illustrate this notion.

Example: Consider a system of linear equations $\mathbf{x} = \mathbf{H}\mathbf{x} + \mathbf{b}$, with the coefficient matrix $\mathbf{H}$ as a 5×5 sparse matrix given by

$$\mathbf{H} = \begin{pmatrix} 0.11 & 0 & 0 & 0.14 & 0.15 \\ 0 & 0.22 & 0.23 & 0.24 & 0 \\ 0 & 0 & 0.33 & 0 & 0 \\ 0 & 0 & 0 & 0.44 & 0.45 \\ 0.51 & 0 & 0.53 & 0 & 0 \end{pmatrix} \tag{4.24}$$

Since the number of nonzero elements in each row does not vary significantly, we can use the following dense matrix ($\mathbf{D}$) to store the nonzero elements

$$\mathbf{D} = \begin{pmatrix} 0.11 & 0.14 & 0.15 \\ 0.22 & 0.23 & 0.24 \\ 0.33 & 0 & 0 \\ 0.44 & 0.45 & 0 \\ 0.51 & 0.53 & 0 \end{pmatrix} \tag{4.25}$$

together with the following index matrix (**N**)

$$\mathbf{N} = \begin{pmatrix} 1 & 4 & 5 \\ 2 & 3 & 4 \\ 3 & 0 & 0 \\ 4 & 5 & 0 \\ 1 & 3 & 0 \end{pmatrix}, \tag{4.26}$$

which stores the column indices of nonzero elements in the original sparse matrix. Note that the zero entries in the index matrix represent the zero elements in **D**. ◇

The transition probability matrix of an absorbing Markov chain or an ergodic Markov chain is determined according to the dense matrix. Then, we can use **D** and **N** together to carry out the calculation of the primary estimates.

The procedure for determining state transitions based on matrix **N** is as follows. Suppose the current state is 2 (represented by the second row) and the next state selected is 3 (represented by the third column). Then, the real next state is 4, which is the content of element $\mathbf{N}_{23}$. Corresponding to this state transition, similar procedure is used to carry out the calculation of the primary estimate based on matrix **D**.

When the number of nonzero elements in each row varies significantly, the above approach cannot economize the storage space. Consider the following two cases. First, if matrix **H** is sparse with only few rows without any zero elements. Second, if matrix **H** is sparse and there are many rows containing only zero elements.

In the first case, the full rows should be handled separately, while the sparse rows can be handled by a dense matrix. This technique can be extended to handle such a case wherein the number of nonzero elements in each row varies significantly.

The occurrence of the second case in a large sparse matrix is rare. If such a problem arises we can use an index matrix and an index array to identify the locations of nonzero elements. The index matrix will be used to store the column indices of nonzero elements, while the index array will be used to store the corresponding row

indices. Let us illustrate with the following example.

Example: Consider a 5×5 coefficient matrix $\mathbf{H}$ given by

$$\mathbf{H} = \begin{pmatrix} 0.11 & 0 & 0 & 0 & 0.15 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0.51 & 0 & 0.53 & 0 & 0 \end{pmatrix}. \tag{4.27}$$

Then, the dense matrix ($\mathbf{D}$) is given as

$$\mathbf{D} = \begin{pmatrix} 0.11 & 0.15 \\ 0.51 & 0.53 \end{pmatrix}, \tag{4.28}$$

and the column indices are stored as

$$\mathbf{N} = \begin{pmatrix} 1 & 5 \\ 1 & 3 \end{pmatrix}, \tag{4.29}$$

while the corresponding row indices are stored by index array

$$M = (1 \quad 5). \quad \diamond \tag{4.30}$$

4.8 Results and Discussion

In this section, we consider a Laplace equation, which is one of the most important partial differential equations in mathematical physics and engineering [20, p. 443]. The objective is to illustrate the application of the Monte Carlo solutions and compare the effects of different sampling methods on the solutions. We chose the inverse method as a comparison for the weighted sampling, since this method is usually used. Also, the inverse method is expected to perform well due to the fact that the number of mass points in this problem is at most only four. Several aspects are examined, including the solution and its error, the processing time, the vectorization speedup, and the efficiency.

For this purpose, the Laplace equation is applied on a boundary value problem. This problem was chosen since it can result in a sparse matrix with regular nonzero structure (a band matrix). Therefore, it is expected that the Monte Carlo methods perform well for this problem. Moreover, as a comparison of the Monte Carlo solutions, we use the iterative method since it is good for a sparse matrix with regular nonzero structure [25].

Using a cartesian coordinate system, the problem can be defined as

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0. \tag{4.31}$$

The boundary conditions for the region $0 \leq x \leq 10$, and $0 \leq y \leq 10$ are chosen as $u(x,0) = u(x,10) = u(10,y) = -10$ and $u(0,y) = 10$.

Further, finite difference method using five point formula [20, p. 444] is used to approximate the original problem. The $x-$ and $y-$axes are discretized into 33 intervals. This discretization results in 1024 internal points. Each point represents an unknown in the linear equation. Thus, the linear system contains 1024 linear equations. The linear equations can then be solved using the Gauss-Seidel iterative method [20, p. 450] or the Monte Carlo methods.

In order to verify the numerical solutions, we use the analytical solution of the Laplace equation obtained using some results from reference [4, p. 312].

In the discussion, we use the same notations for the scalar and vector codes of Monte Carlo algorithms. The notations are given as

1. SEIDEL: code of the Gauss-Seidel iterative method,
2. ERGIN: code of the ergodic algorithm using the inverse sampling method,
3. ERGDS: code of the ergodic algorithm using Brown's sampling method,
4. ERGWS: code of the ergodic algorithm using the weighted sampling method,
5. ABSIN: code of the absorbing algorithm using the inverse sampling method,
6. ABSDS: code of the absorbing algorithm using Brown's method, and
7. ABSWS: code of the absorbing algorithm using the weighted sampling method.

4.8.1 Solutions for All Unknowns

This subsection reports the solutions obtained using different codes. The solutions of all unknowns are first analyzed. Then, the convergence of the solutions for a particular unknown is examined. In comparing the solutions, we chose the solution of the iterative method as a reference.

The solutions of the Laplace equation for all unknowns are shown in Figures 4.7–4.12. Note that the Monte Carlo solutions are obtained by employing 1000 samples for each unknown.

Figure 4.7 shows the solution obtained using the iterative method (SEIDEL). The number of iterations for the iterative method which results in a convergent solution is found to be equal to 80.

Figures 4.8 and 4.9 depict the solutions obtained by ERGWS code utilizing random walk lengths 50 and 125, respectively.

It should be noted that ERGIN and ERGDS codes result in the same solutions as shown in Figures 4.8 and 4.9, since the transition probabilities are uniformly distributed.

Figure 4.10 demonstrates the solution obtained using ABSIN, while Figures 4.11 and 4.12 depict the solution obtained by ABSDS and ABSWS, respectively. It is found that Figures 4.10 to 4.12 are similar to Figure 4.7. This means that ABSIN, ABSDS and ABSWS codes result in approximately correct solutions.

Table 4.1: The average values of solutions for all unknowns and their errors.

Average	Iterative solution	Monte Carlo Solutions					
		Ergodic			Absorbing		
		ERGIN	ERGDS	ERGWS	ABSIN	ABSDS	ABSWS
Solution	-2.8664	-2.3394	-2.3394	-2.3394	-2.7563	-2.9611	-2.7583
Error	0.0000	1.3967	1.3967	1.3967	0.5505	0.5855	0.5587

The error for all unknowns of the Monte Carlo solution is calculated relative to the solution of the iterative method. We define the error as

$$\text{Error} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}, \tag{4.32}$$

where $\hat{y}_i$ is a solution for an unknown obtained by the iterative method, while y_i is the solution for the same unknown obtained using the Monte Carlo code. The number of unknowns is denoted by n.

The average values of the solution for all unknowns and the corresponding errors are presented in Table 4.1. As shown in this table, the Monte Carlo codes employing an absorbing Markov chain result in smaller errors. For each sample, the random walk length in the ergodic Markov chain is chosen to be equal to 125, which also represents the maximum random walk length in the absorbing Markov chain.

As shown in Table 4.1, the ergodic algorithms result in larger errors than the absorbing ones. This is caused by the fixed lengths of the random walks for estimating the solutions of all unknowns. When we chose a random walk of length 50, good

solutions of the boundary points are obtained, while poor solutions result for the central points, as shown in Figure 4.8. However, if the random walk of length is 125, the solutions of the central points improve, but the solutions of the boundary points become inacceptable, as shown in Figure 4.9. In conclusion, the absorbing codes are more suitable for estimating all unknowns.

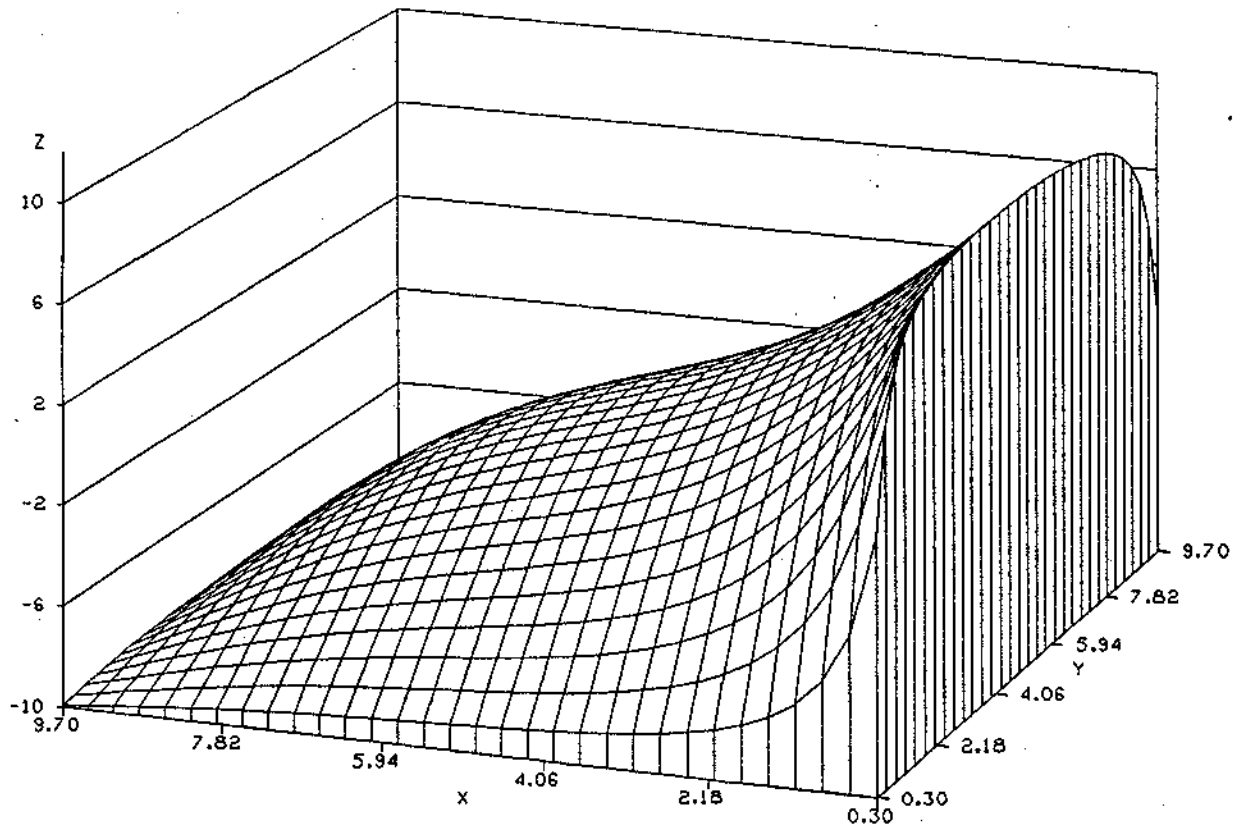

Figure 4.7: Solution obtained using the iterative method (SEIDEL).

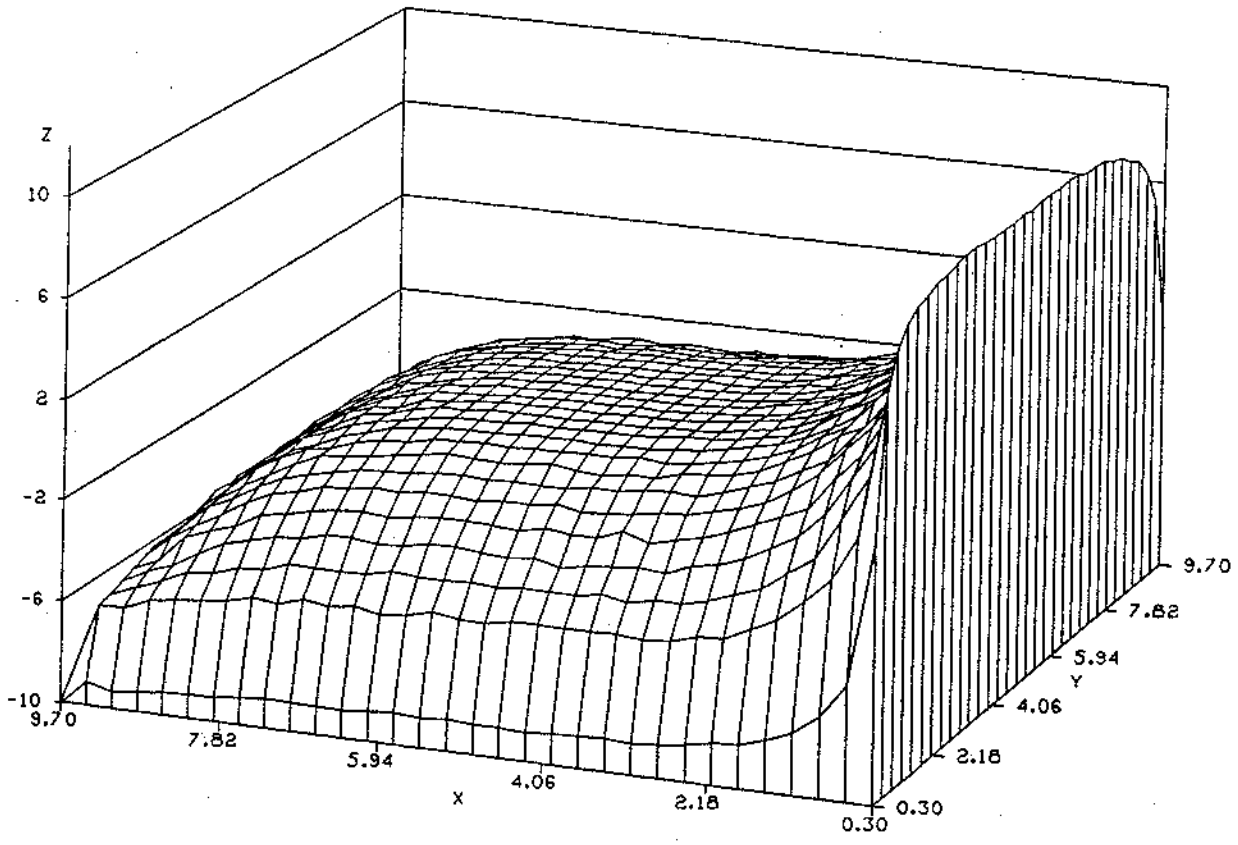

Figure 4.8: Solution obtained by ERGWS, sample size=1000, random walk length=50.

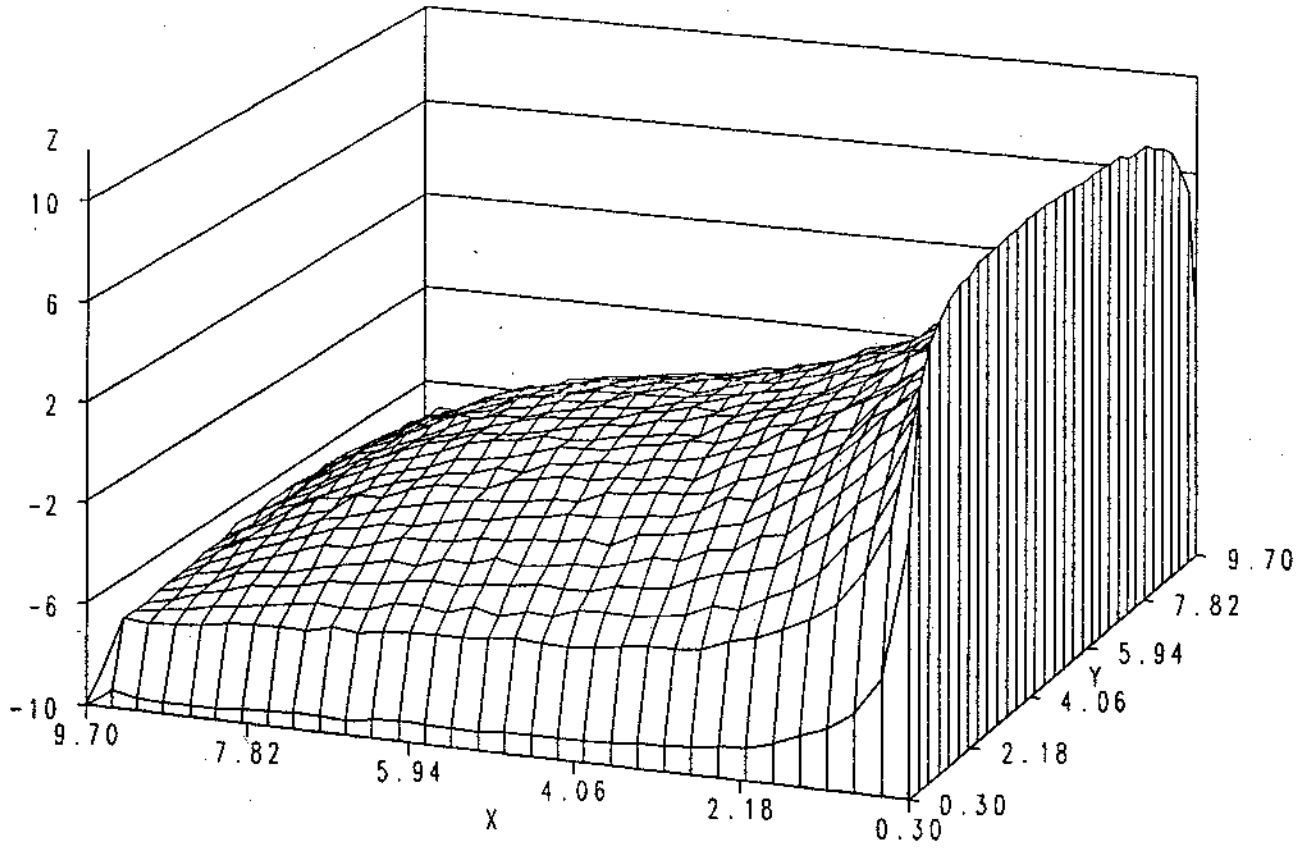

Figure 4.9: Solution obtained by ERGWS, sample size=1000, random walk length=125.

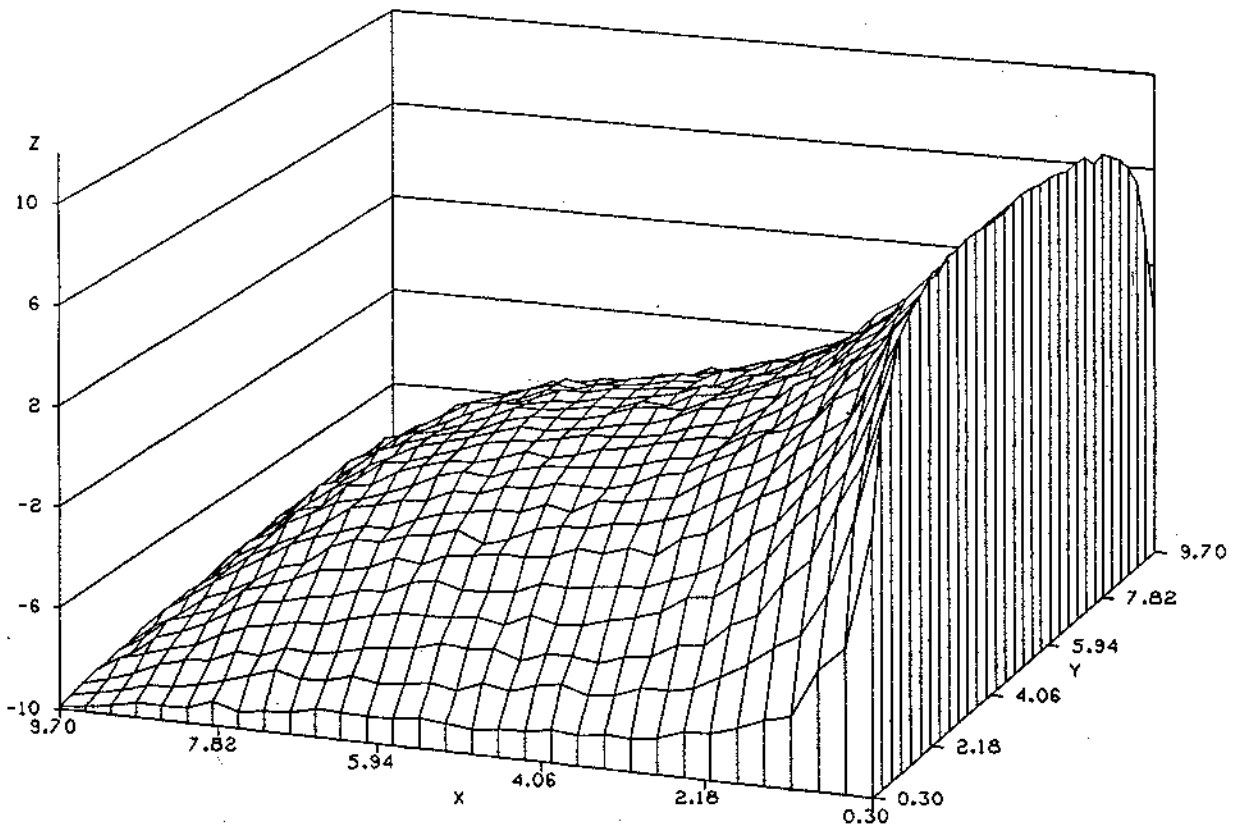

Figure 4.10: Solution obtained using ABSIN, sample size=1000, maximum random walk length=100.

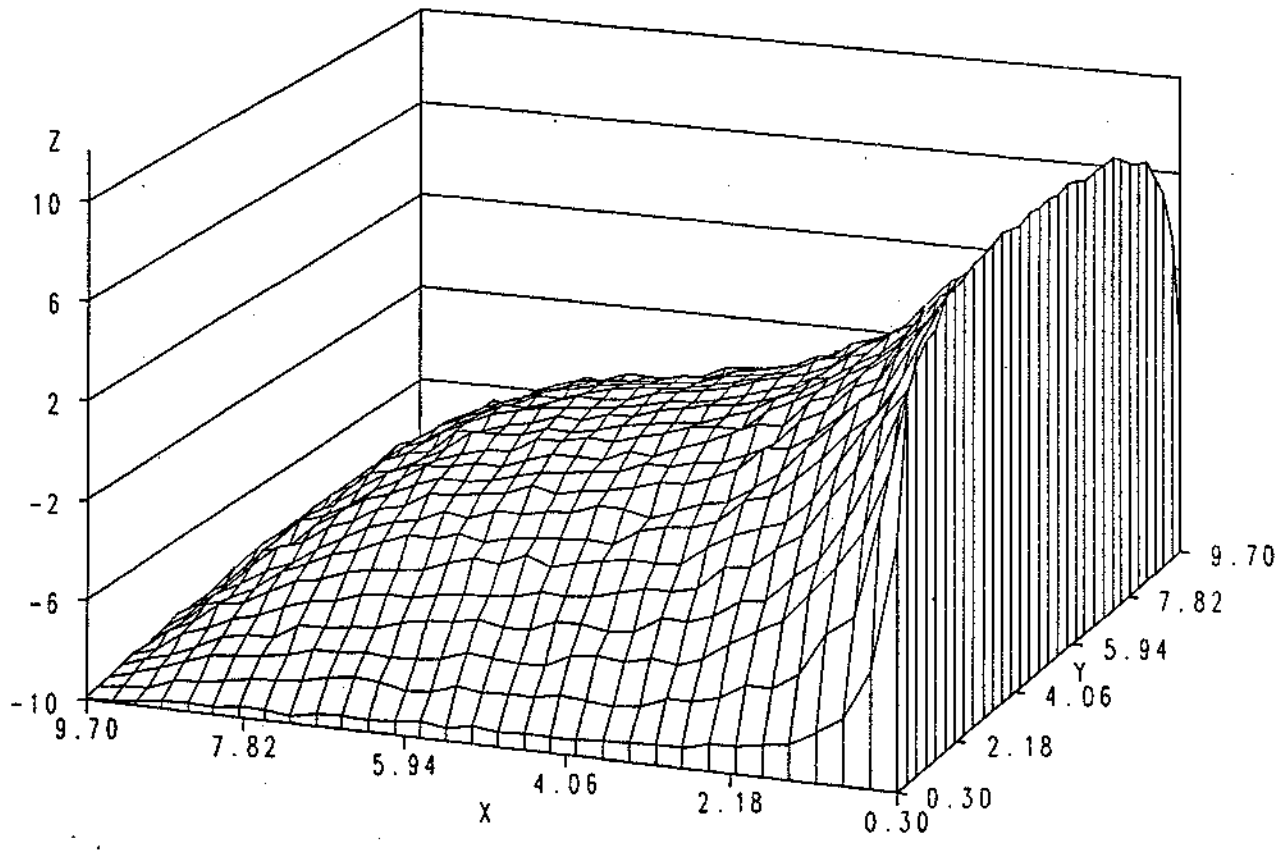

Figure 4.11: Solution obtained using ABSDS, sample size=1000, maximum random walk length=100.

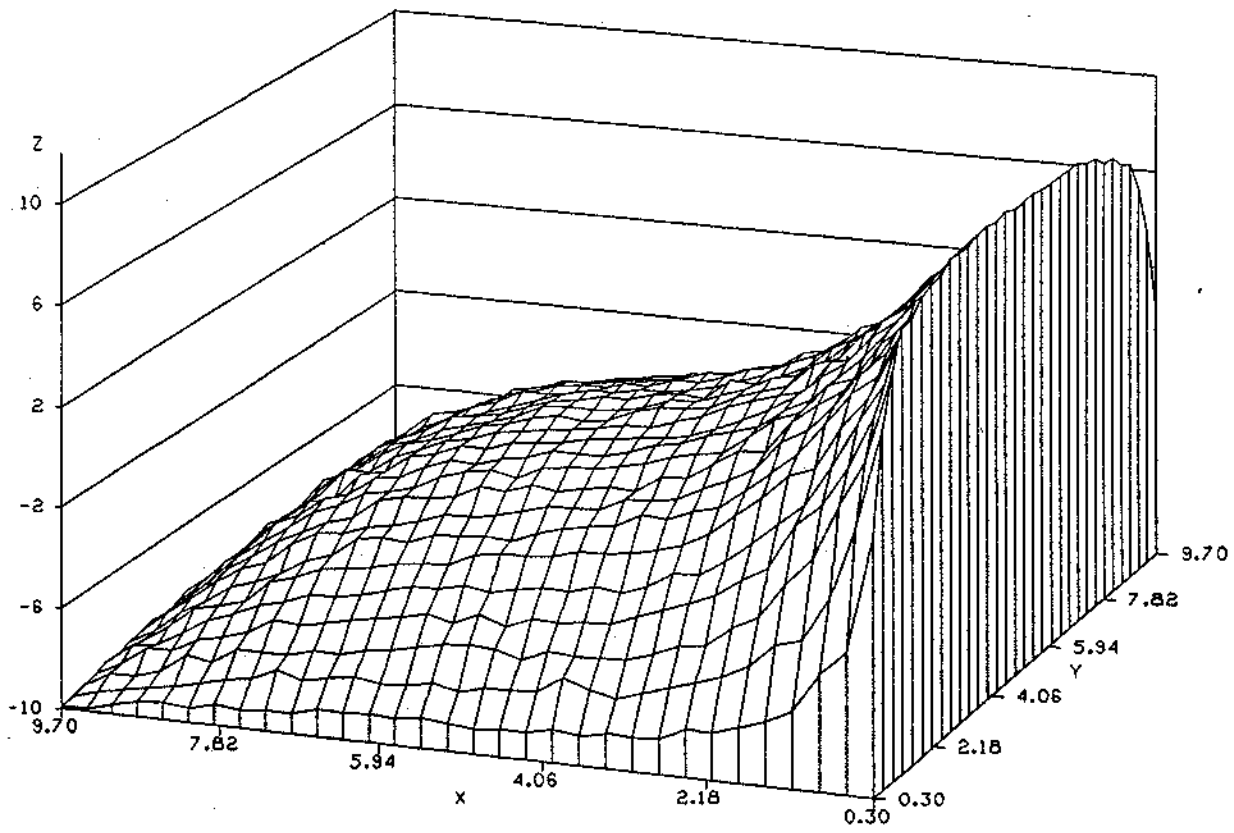

Figure 4.12: Solution obtained using ABSWS, sample size=1000, maximum random walk length=100.

4.8.2 Solutions for Particular Unknowns

In this subsection, we examine the solutions for three different unknowns located at the *boundary, intermediate and centre* points of the region. The boundary point is located on the lower left corner. The centre point is the cross section between the two diagonals, while the intermediate point is in between the boundary and centre points. Their cartesian coordinates are (0.3030, 0.3030), (2.4242, 2.4242) and (4.8485, 4.8485), respectively.

Figure 4.13 gives the solutions of the Laplace equation at point (0.3030, 0,3030) obtained by different codes. The random walk lengths for the ergodic codes are chosen to be equal to 10 state transitions. The maximum random walk lengths for the absorbing codes are restricted to 125 state transitions. This figure demonstrates that the convergence of the ergodic's solutions (ERGIN, ERGDS and ERGWS) to the SEIDEL's solutions is better than those of the ABSIN, ABSDS and ABSWS. Note that the ergodic codes using the inverse method (ERGIN), Brown's method (ERGDS) and the weighted sampling (ERGWS) result in the same solutions, since each row of the transition probability matrix of the ergodic Markov chain is uniformly distributed.

In order to demonstrate the variation of the solution we use *standard deviation of a solution* (SD[solution]), which is defined as [56, p. 145]

$$SD[\overline{X}] = \sqrt{\frac{S^2}{k}}, \tag{4.33}$$

where k is a sample size; $\overline{X}$ and S^2 denote the estimators of the distribution mean and variance, respectively. Note that this equation is the numerator of the equation for the fractional standard deviation, given in Equation (2.4). We do not use FSD here, since some of the solutions are zero.

The standard deviations of Monte Carlo solutions, given in Figure 4.13, are demonstrated in Figure 4.14. It shows that the ergodic codes attain the lowest SDs, followed by those of ABSDS, ABSWS and ABSIN, respectively. The ABSIN code results in

slightly higher SDs than ABSWS.

Figure 4.15 demonstrates the solutions at point (2.4242, 2.4242), which is the intermediate point. We chose 80 as the random walk lengths for the ergodic codes, and 125 as the maximum random walk lengths for the absorbing codes. This figure shows that all solutions converge to the SEIDEL's solutions.

Figure 4.16 shows the standard deviations of Monte Carlo solutions given in Figure 4.15. It is seen that all SDs are relatively equal for the corresponding sample size, except those of ABSDS.

Figure 4.17 depicts the solutions at point (4.8485, 4.8485) resulted from different codes. The maximum random walk lengths for absorbing codes are restricted to 125 state transitions. The convergence of all solutions are good except that of ABSIN.

Figure 4.18 demonstrates the SDs of the Monte Carlo solutions shown in Figure 4.17. This figure shows that the ergodic codes attain the lowest SDs, then followed by ABSWS, ABSDS and ABSIN, respectively. The ABSWS results in slightly higher SDs than those of ergodic codes.

Based on these results, we can conclude that the ergodic codes attain the closest solutions relative to the Gauss-Seidel's solutions. Also, they result in the smallest standard deviations of the Monte Carlo solutions. Thus, the ergodic codes are more suitable for estimating an unknown at a particular point.

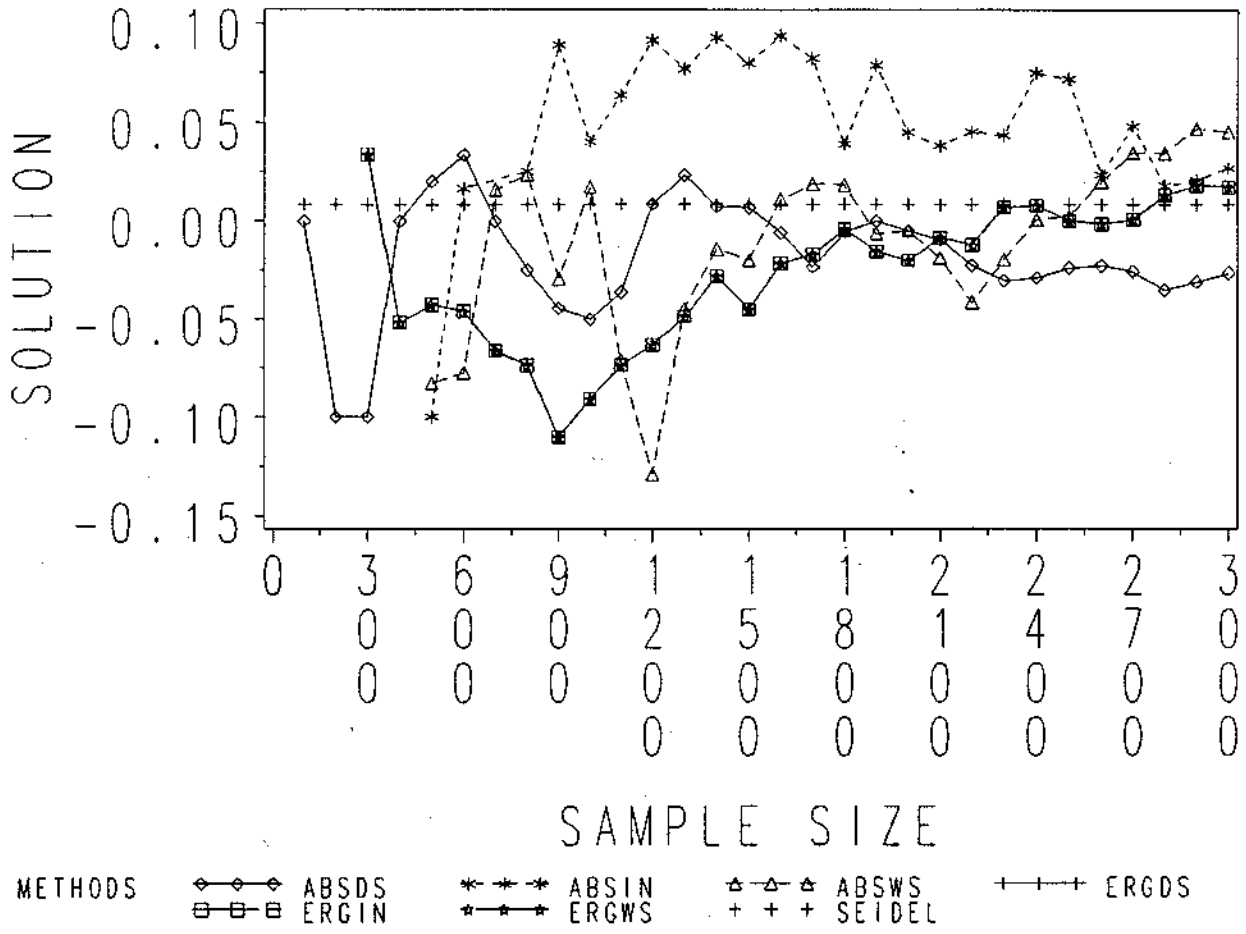

Figure 4.13: Solution of the Laplace equation at the boundary point, ERGIN, ERGDS and ERGWS use random walk lengths=10, ABSIN, ABSDS and ABSWS utilize maximum random walk lengths=125.

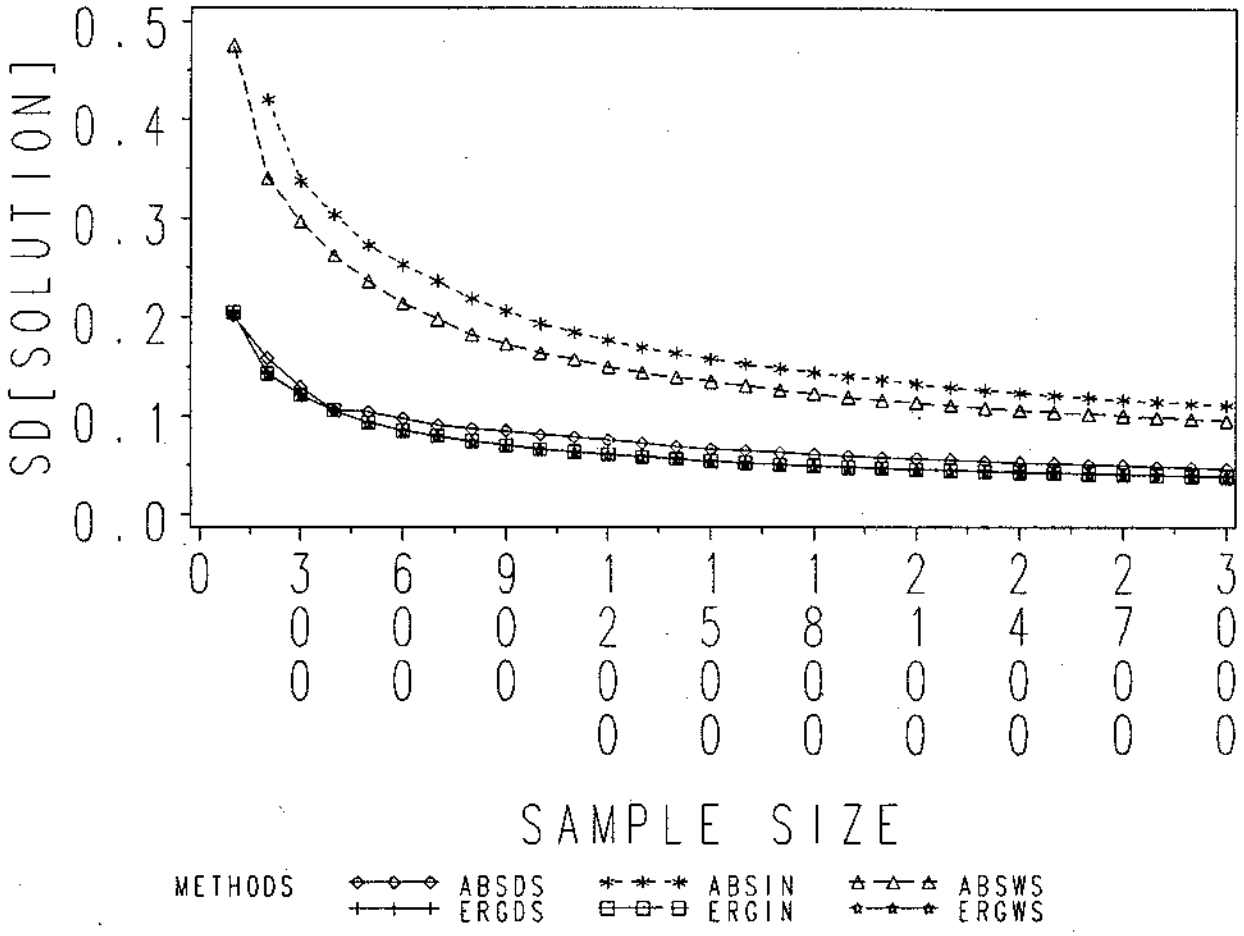

Figure 4.14: Standard deviations of Monte Carlo solutions given in Figure 4.13.

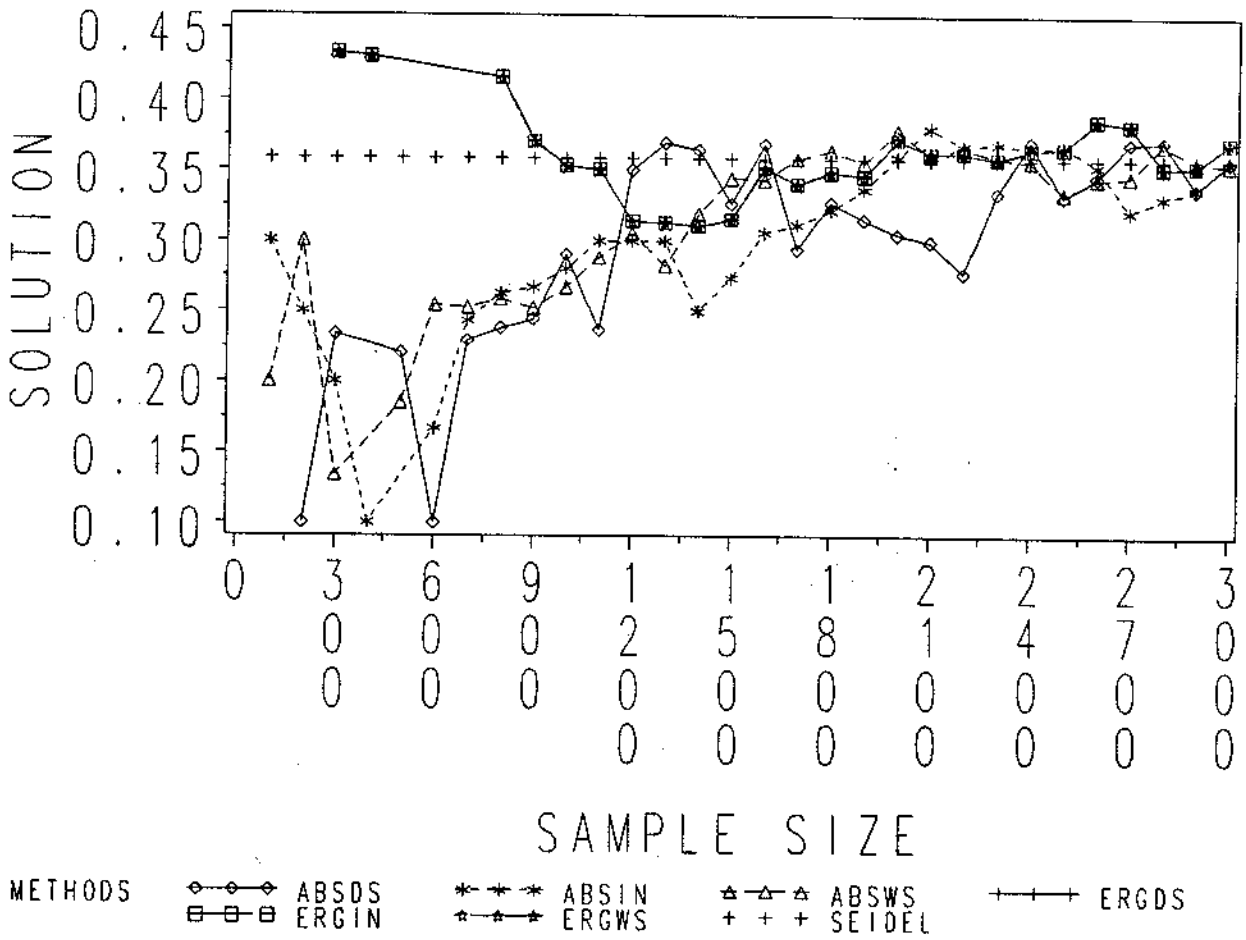

Figure 4.15: Solution of the Laplace equation at the intermediate point, ERGIN, ERGDS and ERGWS use random walk lengths=80, ABSIN, ABSDS and ABSWS utilize maximum random walk lengths=125.

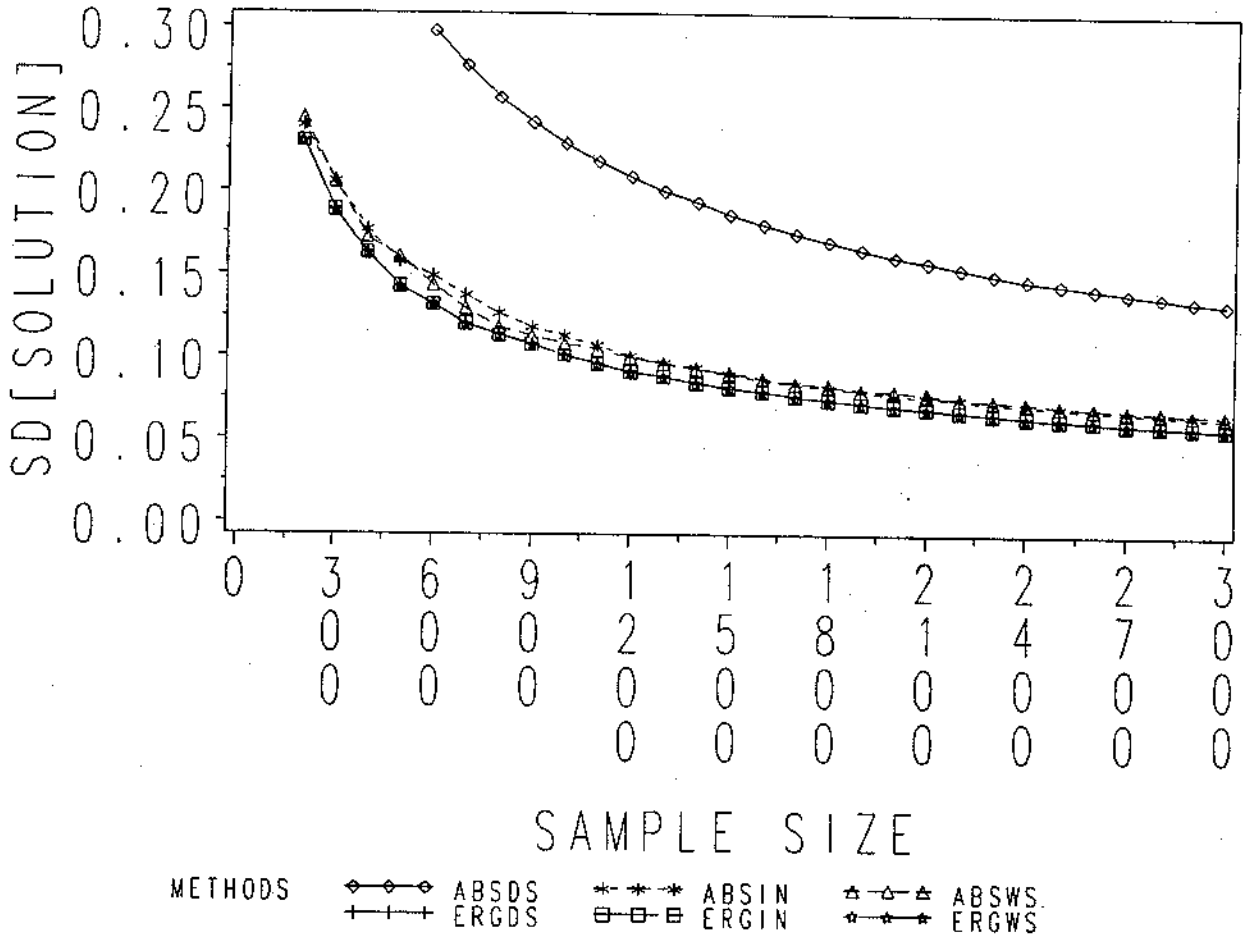

Figure 4.16: Standard deviations of Monte Carlo solutions shown in Figure 4.15.

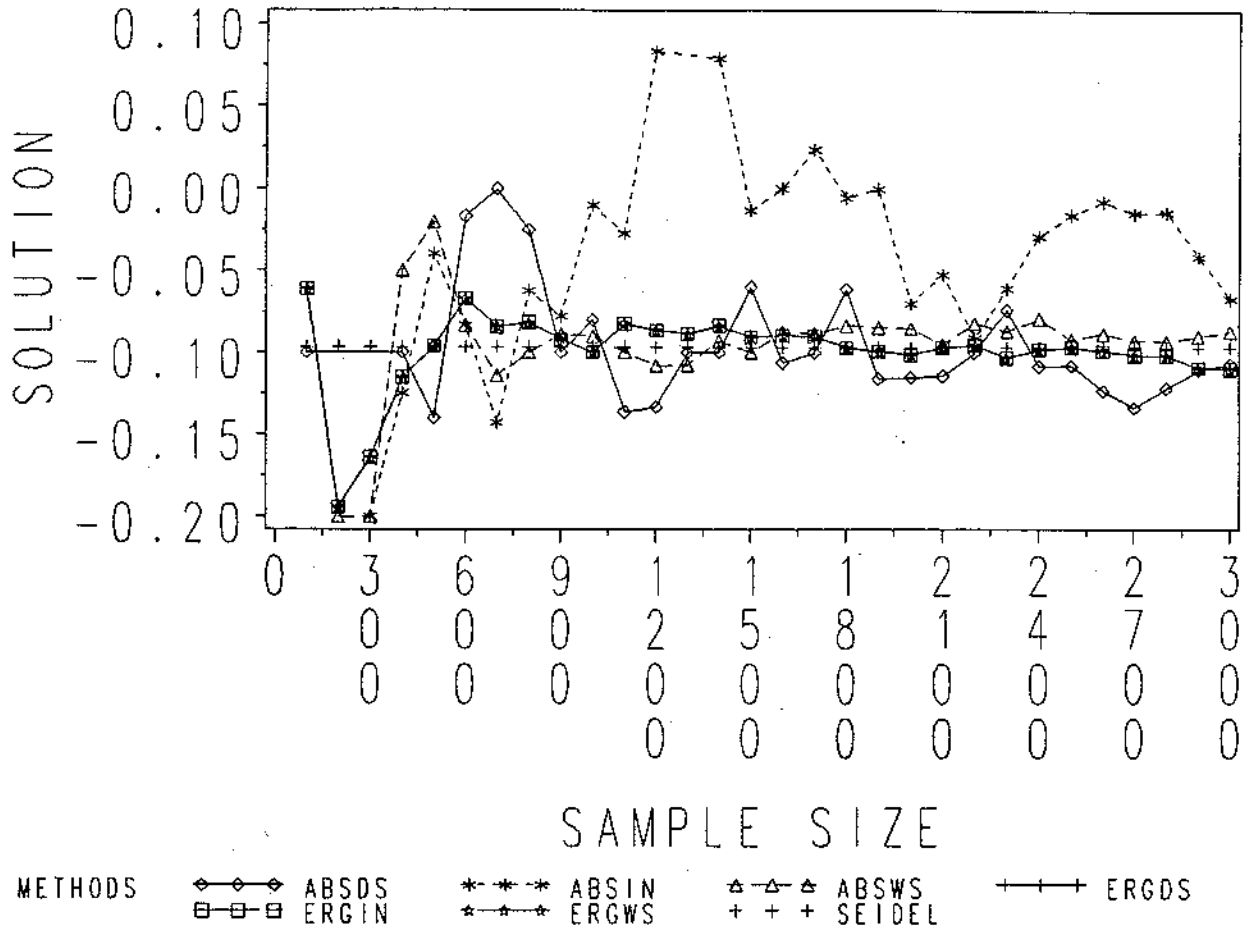

Figure 4.17: Solution of the Laplace equation at the centre point, ERGIN, ERGDS and ERGWS use random walk lengths=120, ABSIN, ABSDS and ABSWS utilize maximum random walk lengths=125.

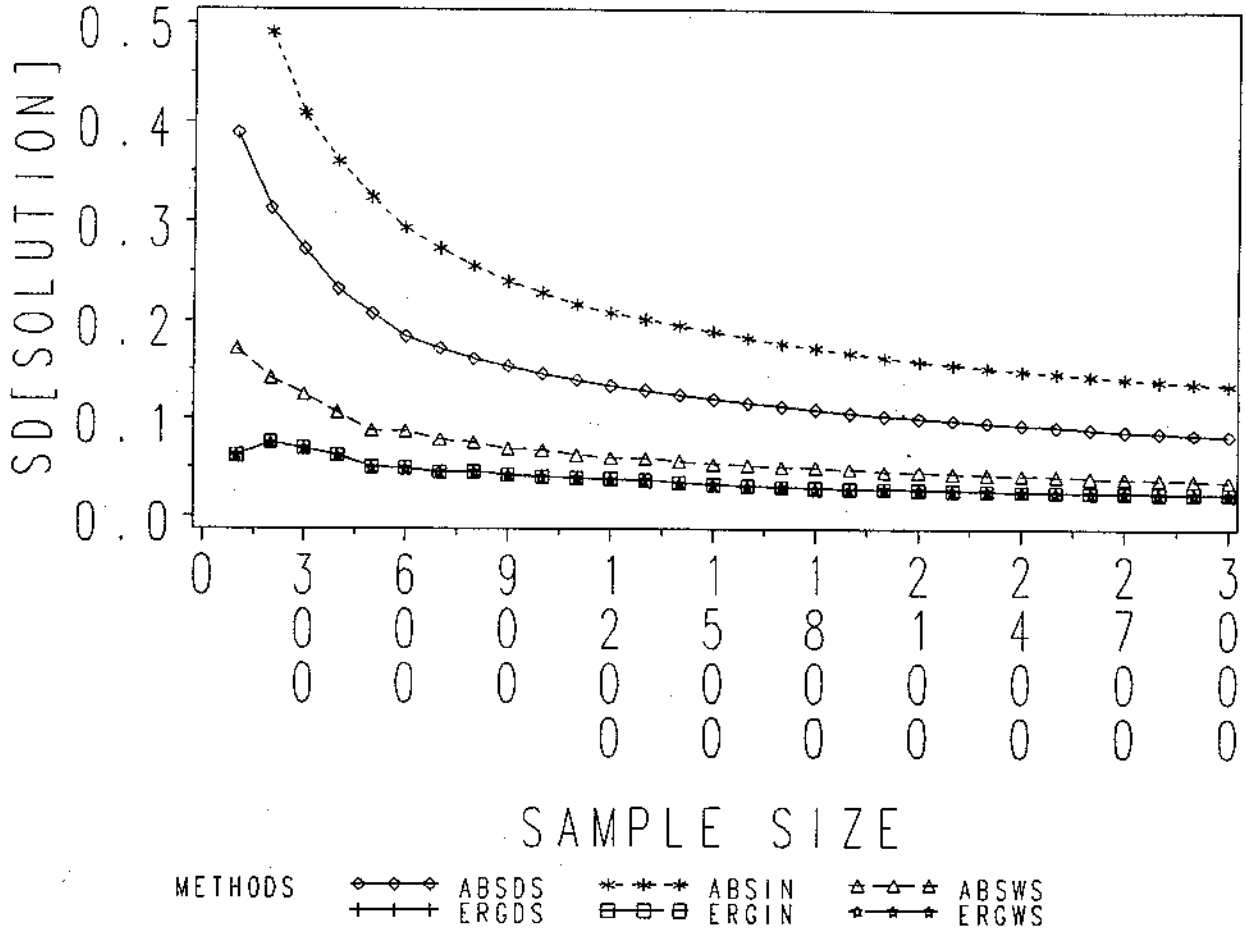

Figure 4.18: Standard deviations of Monte Carlo solutions depicted in Figure 4.17.

4.8.3 Processing Time

In this subsection, we report the processing time of different codes. The scalar and vector processing time of the solutions for all unknowns are first discussed. Then, the scalar and vector processing of the solutions for particular unknowns are presented. Finally, this subsection demonstrates the growth of processing time with respect to the growth of the number of unknowns.

Table 4.2: Processing time for estimating all unknowns.

Sample size	Processing Mode	Processing time in seconds					
		Ergodic			Absorbing		
		ERGIN	ERGDS	ERGWS	ABSIN	ABSDS	ABSWS
100	Scalar	30.80	31.09	24.36	32.30	39.28	31.99
	Vector	15.62	7.10	4.46	11.56	12.98	10.62
500	Scalar	154.15	155.54	121.65	161.95	193.87	150.48
	Vector	79.75	34.05	20.32	43.97	48.62	39.08
1000	Scalar	308.45	310.58	243.64	324.69	387.97	300.89
	Vector	164.01	69.18	41.23	81.89	92.19	73.46
2000	Scalar	618.42	621.02	485.23	644.35	778.78	601.56
	Vector	329.25	141.43	86.03	163.40	178.48	147.21

The scalar and vector processing time of solutions for all unknowns are depicted in Table 4.2. In the scalar processing, the ergodic and absorbing codes incorporating the weighted sampling method (ERGWS and ABSWS) require about 20 % less computing time than those of the codes implementing the inverse method (ERGIN and ABSIN). The ergodic and absorbing codes implementing Brown's method (ERGDS and ABSDS) require larger processing time than those incorporating the inverse method. The performance of the inverse method is comparatively good since the maximum number of comparisons is only four.

In the vector processing, Table 4.2 shows that ERGIN requires 3.5 times larger computing time than that of ERGWS. This means that the weighted sampling can enhance the vectorization of the ergodic algorithm; however, it is less successful in

the absorbing algorithm.

We further compare the processing time for all unknowns required by the iterative method (SEIDEL) and the Monte Carlo method, which requires the least processing time. For this purpose, we chose sample size of 2000, since the solutions obtained by SEIDEL and ERGWS are relatively equal. Table 4.2 demonstrates that ERGWS requires the smallest processing time in scalar and vector processor, which are 485.23 and 86.03 seconds, respectively. The scalar and vector processing of SEIDEL require 616 milliseconds and 456 milliseconds, respectively.

Therefore, the scalar ERGWS requires about 788 times longer processing time than SEIDEL; whereas, the vector ERGWS requires about 188 times longer processing time than the vector SEIDEL. Assuming that the errors are relatively equal, the scalar ERGWS is beneficial if it computes less than 0.13 % of the total unknowns (1.30 unknowns). Moreover, the vector ERGWS still gives benefit when it estimates less than 0.53 % of the total unknowns (5.43 unknowns).

Thus, for the estimation of a single unknown on the IBM 3090 scalar processor, the speedup of the ERGWS over the iterative method is about 1.30; whereas, in the vector processor, the speedup is about 5.43.

Table 4.3: Processing time for estimating particular unknowns.

Points	Process-ing Mode	Processing time in milliseconds					
		Ergodic			Absorbing		
		ERGIN	ERGDS	ERGWS	ABSIN	ABSDS	ABSWS
Boundary	Scalar	122.59	158.40	99.42	69.39	97.00	81.24
	Vector	67.36	49.13	17.08	12.49	20.69	16.98
Interme-diate	Scalar	970.55	1048.48	767.99	934.86	1151.72	873.69
	Vector	545.25	314.74	137.39	195.56	236.25	176.22
Centre	Scalar	1457.69	1562.29	1156.06	1206.35	1499.02	1119.24
	Vector	823.55	497.98	212.90	475.82	576.68	424.91

Table 4.3 shows the scalar and vector processing time of different unknowns at three different points. The number of samples is 2000. The objective here is to

examine the range of processing time required by different unknowns at different points (boundary, intermediate and centre). For the absorbing codes (ABSIN and ABSWS), the centre point requires the largest processing time since it needs the longest random walk length. Obviously, the boundary point require less processing time than the centre point.

Table 4.4: Processing time for different number of unknowns.

Number of unknowns	Mode	Processing time in seconds						Iterative (milli-seconds)
		Ergodic			Absorbing			
		ERGIN	ERGDS	ERGWS	ABSIN	ABSDS	ABSWS	
64	S	9.97	10.32	7.93	7.06	7.81	6.96	7.83
	V	5.49	2.04	1.35	1.77	2.24	1.61	5.72
256	S	77.02	80.63	60.54	69.05	79.86	64.81	61.49
	V	42.21	16.20	10.50	15.51	18.27	13.95	44.57
1024	S	618.42	621.02	485.23	644.35	778.78	601.56	483.56
	V	329.25	141.43	86.03	163.40	178.48	147.21	363.34

S: scalar, and V: vector.

For the ergodic codes (shown in Table 4.3), the processing time for estimating an unknown at the intermediate point is about 7.8 times larger than that of the boundary point, since the random walk length for the intermediate point was chosen to be 80 while that of the boundary point is 10. Similarly, the processing time for the centre point is 1.5 times larger than that of the intermediate point since the random walk length for the centre point was chosen to be 120.

The average random walk lengths of ABSIN for the boundary, intermediate and centre points are 6, 93 and 121, respectively; whereas, those of ABSWS are 7, 93 and 120, respectively. The maximum random walk length for ABSIN, ABSDS and ABSWS is 125.

Table 4.4 demonstrates that the scalar and vector processing time of Monte Carlo solutions increase by a factor of about 8, which is caused by the increase in problem size by a factor of 4 and the increase in random walk length by a factor of 2. The number of samples for each unknown in Monte Carlo codes is 2000. The scalar and

vector processing time of iterative method increase by a factor of about 8, which is caused by the increase in problem size by a factor of 4 and the increase in maximum iteration by a factor of 2.

4.8.4 Speedup

In this subsection, we present the vectorization speedup for each method. This speedup is obtained as the scalar processing time divided by the corresponding vector processing time. First, the speedups of the Monte Carlo codes for estimating all unknowns are shown. Then, we demonstrate the speedups of the Monte Carlo codes for estimating particular unknowns.

Table 4.5: Vectorization speedup for each method estimating all unknowns.

Sample size	Monte Carlo codes					
	Ergodic			Absorbing		
	ERGIN	ERGDS	ERGWS	ABSIN	ABSDS	ABSWS
100	1.97	4.38	5.46	2.79	3.03	3.01
500	1.93	4.57	5.99	3.68	3.99	3.85
1000	1.88	4.49	5.91	3.96	4.21	4.10
2000	1.88	4.39	5.64	3.94	4.36	4.09

Table 4.5 describes the vectorization speedup for each method, when it estimates all unknowns. It is obtained from Table 4.2. The maximum speedup (about 6) is achieved by the vector ERGWS over the scalar one. The ERGWS can achieve the best speedup, since the weighted sampling contains no loop and the random walk length is predetermined.

It is seen in Table 4.5 that the increase in sample size from 100 to 1000 affects the speedups of the absorbing codes (ABSIN, ABSDS and ABSWS). In the ergodic codes however, the speedups are relatively constant with the increase in number of samples.

Table 4.6: Vectorization speedup for particular unknowns.

Points	Speedups					
	Ergodic			Absorbing		
	ERGIN	ERGDS	ERGWS	ABSIN	ABSDS	ABSWS
Boundary	1.82	3.22	5.82	5.56	4.69	4.78
Intermediate	1.78	3.33	5.59	4.78	4.88	4.96
Centre	1.77	3.14	5.43	2.54	2.60	2.63

Table 4.6 describes the effect of locations of unknowns to the vectorization speedup. For the ergodic codes, the speedups for different locations are relatively equal. For the absorbing codes, the centre point has the least speedup since this point requires the longest random walk length wherein the vectorization of state transition process is relatively slow due to the existence of `if`-statement.

Table 4.7 shows the vectorization speedups for different number of unknowns. These speedups are obtained according to Table 4.4. The speedups for the iterative method and the ergodic codes are relatively constant. The absorbing codes attain the highest speedups when the number of unknowns is equal to 256. It is due to the optimality of vector processing for this size.

Table 4.7: Vectorization speedups for different number of unknowns.

Number of unknowns	Speedups						
	Ergodic			Absorbing			Iterative method
	ERGIN	ERGDS	ERGWS	ABSIN	ABSDS	ABSWS	
64	1.82	5.06	5.87	3.99	3.49	4.32	1.37
256	1.82	4.98	5.77	4.45	4.37	4.65	1.38
1024	1.88	4.39	5.64	3.94	4.36	4.09	1.33

4.8.5 The Efficiency

We discuss here the efficiencies of the Monte Carlo solutions at the boundary, intermediate and centre points. They are obtained based on the scalar and vector processing time presented in Table 4.3. The sample variances of the solutions can be calculated from Figures 4.14 to 4.18, for a sample size of 2000.

Table 4.8: Efficiencies of different scalar codes relative to those of scalar ABSIN.

Points	Relative Efficiencies				
	ERGIN	ERGDS	ERGWS	ABSDS	ABSWS
Boundary	4.920	3.808	6.067	4.142	1.183
Intermediate	1.161	1.075	1.467	0.187	1.013
Centre	27.680	25.827	34.902	2.014	13.632

Tables 4.8 and 4.9 demonstrate the efficiencies of different codes relative to those of ABSIN using the scalar and vector processing time, respectively. It is seen that ERGWS attains the largest relative efficiencies for the solutions of the three points, using scalar as well as vector processing time.

Table 4.9: Efficiencies of different vector codes relative to those of vector ABSIN.

Points	Relative Efficiencies				
	ERGIN	ERGDS	ERGWS	ABSDS	ABSWS
Boundary	1.612	2.210	6.356	3.496	1.019
Intermediate	0.432	0.749	1.715	0.191	1.051
Centre	19.325	31.959	74.752	2.065	14.163

For the solution at the boundary point, ABSIN requires the smallest scalar processing time. The other codes get relative efficiencies larger than one, since the sample variance of ABSIN is comparatively larger than those of the other codes. A similar case occurs for the relative efficiencies obtained using the vector processing time.

ERGWS requires the smallest scalar and vector processing time for the solution at the intermediate point. The relative efficiencies for this point, given in Tables 4.8 and 4.9, also represent the scalar and vector speedups since the sample variances are relatively equal.

Table 4.10: Maximum relative efficiencies (η_{max}) of different vector codes relative to those of scalar ABSIN.

Points	Relative Efficiencies					
	Ergodic			Absorbing		
	ERGIN	ERGDS	ERGWS	ABSIN	ABSDS	ABSWS
Boundary	8.954	12.277	35.314	5.556	19.420	5.661
Intermediate	2.066	3.580	8.201	4.780	0.912	5.023
Centre	48.994	81.025	189.520	2.535	5.235	35.907

The comparison between the efficiencies of different vector codes relative to those of the scalar ABSIN results in maximum relative efficiencies, presented in Table 4.10. These maximum relative efficiencies provide indication of the possible efficiencies achieved by utilizing vectorization and different Monte Carlo methods. The ERGWS achieves the highest efficiency since its sample variance is the smallest.

4.9 Conclusions

In this chapter, we have discussed the Monte Carlo methods for solving a sparse linear system. The inverse method is commonly used for sampling in the Monte Carlo solutions. It was found that this method is time consuming, especially for a long probability table.

The vectorized Monte Carlo codes employing ergodic Markov chains (ERGIN, ERGDS and ERGWS) require simpler codes than those employing absorbing Markov chains (ABSIN, ABSDS and ABSWS), since their random walk lengths for all samples are predetermined. Vectorizing the absorbing Monte Carlo codes requires a stack

processing scheme to handle the random nature of the random walk lengths.

The weighted sampling method enhances the vectorizability of the Monte Carlo codes, since it does not contain any loops and there is no form of data dependency in the computational process which can inhibit vectorization.

In the Monte Carlo codes, the data structures for handling sparse matrices with irregular patterns of nonzero locations basically utilize the indirect addressing techniques.

It was found that ERGWS results in the least scalar and vector processing time, and the largest speedups. For estimating an unknown, the speedup of scalar ERGWS relative to scalar SEIDEL is about 1.3, while the speedup of vector ERGWS relative to vector SEIDEL is about 5.4.

For the solutions at particular points, ERGWS attains the smallest sample variance and achieves the largest relative efficiency as well. In general, the Monte Carlo codes employing ergodic Markov chains entails smaller sample variances than those of Monte Carlo codes utilizing absorbing Markov chains. This is due to the fact that the nonzero elements of matrix $\mathbf{H}$ are the same, and the transition probability matrices are uniformly distributed. If the elements of such a matrix are close to each others, a uniform transition probability matrix can be utilized. Consequently, sampling can be carried out directly as the case of the equiprobable method.

Thus, we have shown that the Monte Carlo codes incorporating the weighted sampling method result in simpler codes, valid solutions, faster execution on scalar as well as vector processing modes, and higher efficiencies.

Chapter 5

Neutron Transport Problems

5.1 Introduction

This chapter considers Monte Carlo methods for solving neutron transport problems. In a neutron transport problem, several quantities of interest are calculated. These quantities represent physically interpretable data, such as the number of collisions in a space element, detector responses and leakage probabilities. These problems usually involve complex geometries and a seven-dimensional space in time, energy, position and angle. The solution of many such problems cannot be obtained using analytical methods, and classical numerical methods can solve only problems involving simple geometries. However, Monte Carlo methods are applicable to solving these problems. These methods often require extensive computing time; therefore, speeding up of the computational process is desirable. This chapter examines whether the use of the weighted sampling method enhances the performance of Monte Carlo calculations in particle transport problems.

In order to investigate the effects of incorporating the weighted sampling method on the Monte Carlo solutions, the inverse method and Brown's method are used as a reference for comparison. The inverse method is chosen since it is used by the Monte Carlo code employed in this work, which is MORSE (Multigroup Oak Ridge

Stochastic Experiment) code. Brown's method is selected since it is best suited for vector processing (as demonstrated in Chapter 2).

This chapter is organized as follows. Some definitions and the theory of neutron transport are briefly reviewed in the next section. Section 5.3 reviews the use of the Monte Carlo method for solving neutron transport problems. The MORSE code is introduced in Section 5.4. In Section 5.5, the methods used for speeding up Monte Carlo simulations are discussed. Section 5.6 briefly reviews different sampling methods and illustrates that the sampling process is the workhorse of the Monte Carlo method. Sections 5.7 to 5.9 examine three neutron transport problems with different physical characteristics, using the three selected sampling methods. The Monte Carlo codes are implemented on the IBM 3090-180VF, and their performance is discussed. Finally, concluding remarks are given in the last section.

5.2 Neutron Transport

Neutron transport is described by a Boltzmann transport equation, which catalogues all possible occurances during the transport process [85]. Some fundamental assumptions of this equation are: (a) neutrons flow without changing direction or speed until they either interact with atomic nuclei in the considered domain or escape from the domain, (b) neutrons do not appreciably alter the interacted medium within the time interval considered, and (c) external fields, such as gravity and electromagnetic fields, are not accounted for [3].

A seven-dimensional space is needed for describing neutron transport. These dimensions are three for spatial coordinates, two for describing direction of particle, and the others for magnitude of velocity (or equivalently energy) and time.

5.2.1 Definitions

Before describing the Boltzmann transport equation, some definitions and terminology mainly based on reference [85] are briefly introduced in this subsection.

The *flux* of neutrons is defined as the number of neutrons per unit area per unit time. Another useful measure is called *fluence,* which is defined as the number of neutrons per unit area. It is equivalent to regarding fluence as a time-integrated flux over some specified time interval.

Neutrons interact with nuclei in several ways. Interactions or collisions can lead to absorption, scattering or multiplicative effects. Absorption occurs when a neutron is absorbed and no neutron is emitted. In scattering, the neutron continues flying after a collision but its energy and direction may be altered. Multiplicative effects occur when more than one neutron are emitted after a neutron is absorbed by the nucleus. Scattering is further classified into several scattering processes, such as elastic scattering and inelastic scattering. Elastic scattering conserves both kinetic energy and momentum of the colliding particles (neutron and target nucleus). In inelastic scattering, a portion of the incident-neutron energy appears as an excitation of the target nucleus which is subsequently released by photon emission. The probabilities of these interactions depend on the neutron's energy and the nature of the target nucleus. The interaction probabilities are described by cross-sections, which are defined below.

The *microscopic cross section* (σ) is defined as the probability of interaction of a neutron with a single nucleus per unit area. The unit of area is usually expressed in *barns,* where a barn is equal to $10^{-28}\ m^2$. The probability of interaction with the aggregate of nuclei that compose a medium is described by the *macroscopic cross section*, which is denoted by Σ . If N is the volumetric density (nuclei/m^3) of the target medium, then $\Sigma = N\sigma$. For a mixture of target nuclide, the macroscopic cross section is given as

$$\Sigma = \sum_i N_i \sigma_i,$$

where N_i and σ_i are, respectively, the volumetric density and the microscopic cross section of the i-th nuclide [85]. The macroscopic cross section has units of reciprocal length and is usually expressed in m^{-1}. The reciprocal of the total cross section, $1/\Sigma$, represents the *mean-free-path* of a neutron in a medium; that is the mean distance a neutron travels before encountering an interaction.

The cross section is energy and material dependent. For a given target nucleus and interaction type, the cross sections are usually tabulated as a function of the incident radiation energy. The particle's energy is divided into G energy groups; each energy group has a set of particle cross sections. The first energy group corresponds to the highest energy, while the $G-$th group corresponds to the lowest energy.

The subscripts in the cross sections designate the types of interactions. These are typically, σ_a for the absorption cross section, $\nu\sigma_f$ for the average number of neutrons released per fission times the fission cross section, and σ_t for the total (all possible reactions) cross section. The scattering cross section is described by a set of energy transfer cross sections, $\sigma_{g,g'}$, which denotes a scattering cross section from energy group g to energy group g'. The scattering cross section is usually represented in a matrix form, which is called the scattering matrix.

Table 5.1 shows a typical example of a neutron scattering matrix. This scattering matrix is an upper right triangular matrix, in which the elements of each row are located starting from the diagonal. In the presentation, however, the elements of each row are written starting from the first column in order to compress the space. This upper right triangular scattering matrix means that a particle cannot gain energy after a collision; that is no up-scattering is allowed.

Table 5.1: A neutron cross section matrix.

Energy Group	Energy (MeV)	σ_a	$\nu\sigma_f$	σ_t	$\sigma_{g,g'}$ Scattering Matrix						
1	3–∞	0.02	0.00	1.81	1.10	0.16	0.23	.	.	0.12	0.07
2	1.4–3	0.00	0.00	2.02	1.70	0.10	0.11	.	.	0.01	
.	.	.	.	.	.	.	.	.	.	.	
.	.	.	.	.	.	.	.	.	.	.	
.	.	.	.	.	.	.	.	.	.	.	
g	0.003–0.017	0.00	0.00	1.46	$\sigma_{g,g}$	$\sigma_{g,g+1}$	$\sigma_{g,g+2}$	.	$\sigma_{g,G}$		
.	.	.	.	.	.	.	.	.			
.	.	.	.	.	.	.	.				
.	.	.	.	.	.	.					
G	$< 10^{-8}$	0.23	0.00	1.57	1.57						

5.2.2 The Boltzmann Transport Equation

In physical terms, the Boltzmann equation accounts for additions to and subtractions from transporting neutrons in a given increment of space, energy, direction and time [85]. The Boltzmann transport equation can be written as follows [3,66].

$$\frac{1}{v}\frac{\partial}{\partial t}\phi(\overline{r},E,\overline{\Omega},t) = -\overline{\Omega}.\nabla\phi(\overline{r},E,\overline{\Omega},t) - \Sigma_t(\overline{r},E)\phi(\overline{r},E,\overline{\Omega},t) + \int\int dE'd\overline{\Omega}'\Sigma_s(\overline{r},E'\rightarrow E,\overline{\Omega}'\rightarrow\overline{\Omega})\phi(\overline{r},E',\overline{\Omega}',t) + Q(\overline{r},E,\overline{\Omega},t), \quad (5.1)$$

where

$\overline{r}$ = position vector,

E = particle's kinetic energy,

v = particle's speed corresponding to its kinetic energy E,

$\overline{\Omega}$ = a unit vector which describes the particle's direction of motion,

t = time variable,

$\phi(\overline{r},E,\overline{\Omega},t)$ = time-dependent angular flux,

$\phi(\overline{r},E,\overline{\Omega},t)dEd\overline{\Omega}$ = number of particles per unit volume and time at the space point $\overline{r}$ and time t with energies in dE about E and with directions in $d\overline{\Omega}$ about $\overline{\Omega}$,

$\frac{1}{v}\frac{\partial}{\partial t}\phi(\overline{r},E,\overline{\Omega},t)dEd\overline{\Omega}$ = net accumulation (gains minus losses) per unit volume and time at the space point $\overline{r}$ and time t of particles with energies in dE about E and with directions in $d\overline{\Omega}$ about $\overline{\Omega}$,

$-\overline{\Omega}.\nabla\phi(\overline{r},E,\overline{\Omega},t)dEd\overline{\Omega}$ = net convective loss (due to particle spreading) per unit volume and time at the space point $\overline{r}$ and time t of particles with energies in dE about E and directions in $d\overline{\Omega}$ about $\overline{\Omega}$,

$\Sigma_t(\overline{r}, E)$ = total cross sections at the space point $\overline{r}$ for particles of energy E,

$-\Sigma_t(\overline{r}, E)\phi(\overline{r}, E, \overline{\Omega}, t)dEd\overline{\Omega}$ = particle loss (by absorption and scattering) per unit volume and time at the space point $\overline{r}$ and time t of particles with energies in dE about E and directions in $d\overline{\Omega}$ about $\overline{\Omega}$,

$\Sigma_s(\overline{r}, E' \rightarrow E, \overline{\Omega}' \rightarrow \overline{\Omega})dEd\overline{\Omega}$ = the differential scattering cross section which describes the probability per unit path that a particle with an initial energy E and initial direction $\overline{\Omega}$, undergoes a scattering collision at $\overline{r}$ which places it into a direction within $d\overline{\Omega}$ about $\overline{\Omega}$ with a new energy in dE about E,

$\{\int\int \Sigma_s(\overline{r}, E' \rightarrow E, \overline{\Omega}' \rightarrow \overline{\Omega})\phi(\overline{r}, E', \overline{\Omega}', t)dE'd\overline{\Omega}'\}dEd\overline{\Omega}$ = number of particles resulting from collisions which enter a unit volume and time at the space point $\overline{r}$ and time t of particles with energies in dE about E and directions within $d\overline{\Omega}$ about $\overline{\Omega}$, and

$Q(\overline{r}, E, \overline{\Omega}, t)dEd\overline{\Omega}$ = number of source particles emitted per unit volume and time at the space point $\overline{r}$ and time t with energies in dE about E and directions within $d\overline{\Omega}$ about $\overline{\Omega}$.

The energy dependence of Equation (5.1) can be represented in terms of energy groups which are defined as

ΔE_g = energy width of the g-th group,

$g = 1$ corresponds to the highest energy group,

$g = G$ corresponds to the lowest energy group,

with the energy constraint given by

$$\sum_{g=1}^{G} \Delta E_g = \int_0^{E_0} dE = E_0, \tag{5.2}$$

which is the maximum particle energy.

The Boltzmann equation can further be integrated with respect to energy over the energy interval ΔE_g. The Boltzmann equation for an energy group g is as follows [66].

$$\frac{1}{v_g}\frac{\partial}{\partial t}\phi_g(\overline{r},\overline{\Omega},t) = -\overline{\Omega}.\nabla\phi_g(\overline{r},\overline{\Omega},t) - \Sigma_t^g(\overline{r})\phi_g(\overline{r},\overline{\Omega},t) + \sum_{g'=g}^{1}\int_{4\pi} d\overline{\Omega}'\Sigma_s^{g'\to g}(\overline{r},\overline{\Omega}'\to\overline{\Omega})\phi_{g'}(\overline{r},\overline{\Omega}',t) + Q_g(\overline{r},\overline{\Omega},t), \tag{5.3}$$

where

$\phi_g(\overline{r},E,\overline{\Omega},t)$ = time-dependent group angular flux,

$\Sigma_t^g(\overline{r},E)$ = energy-averaged total cross section for the g-th group, and

$\Sigma_s^{g'\to g}(\overline{r},\overline{\Omega}'\to\overline{\Omega})$ = group g' to group g scattering cross section.

Analytical solutions of the Boltzmann equation can be obtained only in a few very simple and highly idealized cases [3]. In most practical situations, approximate solutions are the options, which may be obtained using classical numerical methods and Monte Carlo methods [85]. The classical numerical methods generally involve the derivation of a discritized approximation to the Boltzmann equation. The resulting system of equations may then be solved iteratively. The classical numerical solutions are however limited to geometries compatible with cartesian, cylindrical or spherical coordinate systems. Thus, for complex geometry problems, one has to rely on the Monte Carlo method. The Monte Carlo method for solving a Boltzmann equation is discussed in the next section.

5.3 The Monte Carlo Method

In the Monte Carlo method, histories of the travel of individual particles through the geometry are constructed and then analyzed to obtain relevant information, such as fluence and fluence related quantities [59]. A particle history includes the birth of a

particle at the source, its random walk through the transporting medium, where it undergoes various scattering interactions, and its death, which terminates its history. The termination of a particle history can occur when the particle becomes absorbed, leaves the geometric region of interest, or reaches energy level below some preassigned energy cut-off. In the following subsections, the basic requirements and and the random walk procedure of the Monte Carlo method are described.

5.3.1 Basic Requirements

In order to solve the Boltzmann transport equation using the Monte Carlo method, the following items should be provided for computing each particle history (random walk) [66].

Source of Particles

A radiation source is specified by its position, geometry, directional and energy distributions. This provides the $Q(\overline{r}, E, \overline{\Omega}, t)$ term in Equation (5.1). The source variables are produced by constructing random samples from given probability distributions. A source particle is usually assigned a statistical weight of unity. The weight is further modified during random walks.

Geometry

In order to track particles throughout the system and relate the positions of particles to the materials encountered, the geometry can be specified analytically by defining the surfaces of different geometrical objects. This approach however may not be practical for a complex object. An alternative approach is that the geometry may be specified by a set of elementary bodies. The elementary bodies can be combined together using logical operators to form the zone of a particular object. This approach is called the combinatorial geometry method [66].

Scoring

The quantities of interest are estimated at the end of the random walks of particles. The following estimators can be used to score fluence or fluence-like quantities at a point or a region [59]:

- *Body crossing estimator* scores the fluence crossing a surface. This estimator accumulates the weight of particles crossing the surface divided by the absolute value of the cosine of the angle between the normal to the surface and the direction of the incident particle.
- *Track length estimator* evaluates fluence. The track length of particles crossing a given zone is summed, then divided by the volume of the zone.
- *Collision density estimator* evaluates fluence, by accumulating the weight of particles colliding within a zone, divided by the product of the total cross section of the material and the volume of the zone.

In these estimators, scoring is done when a particle visits the region or surface of interest. An expected-value estimator, such as the *next event estimator* [17], is used when the probabilities of particles reaching the region of interest are low. This estimator evaluates the probability of the next collision being at the detector site; however, the position of the particle being tracked is not altered and the particles need not necessarily enter the detector site. The estimator of the detector response at energy E is then given by [39]

$$S(E) = \frac{1}{K} \sum_{i=1}^{M} \frac{w \exp(-\beta)\, p(E' \rightarrow E; \theta)}{2\pi R^2} \tag{5.4}$$

where w is the statistical weight of a particle leaving a collision, β is the number of mean-free-paths from the collision point to the detector site, $p(E' \rightarrow E; \theta)$ is the probability of scattering from energy E to energy E' and reaching the detector at angle θ between the incident and scattered beams, R is the distance between the

collision point and the detector site, while M is the number of collisions encountered, and K is the number of particle histories.

5.3.2 Random Walk Procedure

The Monte Carlo method solves the Boltzmann integro-differential transport equation by randomly tracking sufficient number of random walks through the problem geometry. Estimates of quantities of interest are then obtained by accumulating the contribution of individual particles.

The density of particles leaving a source or emerging from a real collision with phase space coordinates (group $g, \overline{r}, \overline{\Omega}, t$) is denoted by variable $\chi_g(\overline{r}, \overline{\Omega}, t)$, which is defined as

$$\chi_g(\overline{r}, \overline{\Omega}, t) = \sum_{g'=g}^{1} \int_{4\pi} d\overline{\Omega}' \Sigma_s^{g' \to g}(\overline{r}, \overline{\Omega}' \to \overline{\Omega}) \phi_{g'}(\overline{r}, \overline{\Omega}', t) + Q_g(\overline{r}, \overline{\Omega}, t). \qquad (5.5)$$

One of the forms of the Boltzmann transport equation is the *integral emergent particle density equation*, which can be derived from Equation (5.5). This equation is given as

$$\chi_g(\overline{r}, \overline{\Omega}, t) = C_{g' \to g}(\overline{r}, \overline{\Omega}' \to \overline{\Omega}) T_{g'}(\overline{r}' \to \overline{r}, \overline{\Omega}') \chi_{g'}(\overline{r}', \overline{\Omega}', t') + Q_g(\overline{r}, \overline{\Omega}, t), \qquad (5.6)$$

where

$C_{g' \to g}(\overline{r}, \overline{\Omega}' \to \overline{\Omega})$ = a collision integral operator, and

$T_{g'}(\overline{r}' \to \overline{r}, \overline{\Omega}')$ = a transport integral operator.

A detailed derivation of Equation (5.6) and definitions of the integral operators can be found in reference [37]. The collision and transport integral operators are then approximated by the following summation equation

$$\chi_g(\overline{r}, \overline{\Omega}, t) = \sum_{n=0}^{\infty} \chi_g^n(\overline{r}, \overline{\Omega}, t), \qquad (5.7)$$

where

$\chi_g^0(\overline{r},\overline{\Omega},t) = Q_g(\overline{r},\overline{\Omega},t)$, and

$\chi_g^n(\overline{r},\overline{\Omega},t)d\overline{\Omega} = C_{g'\to g}(\overline{r},\overline{\Omega}' \to \overline{\Omega})T_{g'}(\overline{r}' \to \overline{r},\overline{\Omega}')\chi_g^{n-1}(\overline{r}',\overline{\Omega}',t')$, which denotes the emergent particle density of particles emerging from n-th collision and having phase space coordinates (group $g, \overline{r}, d\overline{\Omega}$ about $\overline{\Omega}$, time t).

The collision and the transport integral operators control random walks. The flight distance is sampled from a distribution described by the transport integral operator, while the particle status after a collision is determined based on a distribution defined by the collision integral operator.

The random walk process is initiated by sampling a source particle. The source coordinates (group $g_0, \overline{r}_0, \overline{\Omega}_0$, time t_0) are selected from $Q_g(\overline{r},\overline{\Omega},t)$, and each source particle is given a weight equal to unity. Then, a flight distance R is sampled from $\Sigma_t^{g_0}(r)e^{-g(\overline{r},R,\overline{\Omega}_0)}$ to determine the site for the first collision $\overline{r}_1$, and the particle's age $t_1 = t_0 + R/v_{g_0}$. All particles are forced to scatter with the probability of scattering defined as $\Sigma_s^{g_0}(\overline{r}_1)/\Sigma_t^{g_0}(\overline{r}_1)$. The weight is further modified by the nonabsorption probability, which is defined as $1 - [\Sigma_a^{g_0}(\overline{r}_1)/\Sigma_t^{g_0}(\overline{r}_1)]$.

A new energy group g_1 is sampled according to the distribution

$$\frac{\int_{4\pi} d\overline{\Omega}\Sigma_s^{g_0\to g}(\overline{r}_1,\overline{\Omega}_0 \to \overline{\Omega})}{\Sigma_s^{g_0}(\overline{r}_1)}, \tag{5.8}$$

and a new direction $\overline{\Omega}$ is determined from

$$\frac{\Sigma_s^{g_0\to g_1}(\overline{r}_1,\overline{\Omega}_0 \to \overline{\Omega})}{\Sigma_s^{g_0\to g_1}(\overline{r}_1)}. \tag{5.9}$$

The random walk process is repeated until the particle history is terminated. The particle history can be terminated by energy cut-off, age cut-off and weight cut-off. During the random walk process, the particle contributions to the quantity of interest are scored according to the particle's weight and the estimator used.

5.4 The MORSE Code

A number of general purpose Monte Carlo codes for particle transport calculation are available. MCNP, MORSE and TRIPOLI are perhaps the three most widely used codes [37]. The MORSE (Multigroup Oak Ridge Stochastic Experiment) code is a multipurpose neutron and gamma-ray transport Monte Carlo code. This code consists of about 8000 lines of FORTRAN statements, with very few comments. The capabilities of MORSE include the ability to simulate either a neutron or a gamma-ray problem or a coupled neutron and secondary gamma-ray problem. The capabilities also include the handling of multigroup cross sections, the ability to solve forward or adjoint problems, modular input–output structure, debugging routines, a three-dimensional combinatorial geometry package, and several options of importance sampling techniques [66]. With the three-dimensional combinatorial geometry package provided, MORSE can form complicated geometries by combining several basic geometrical bodies.

The MORSE code is chosen for use in this work, since it is a multigroup code and has been widely used at the University of New Brunswick. In the multigroup Monte Carlo method, the energy range of interest is divided into several groups and the average cross sections are obtained for each group. Therefore, MORSE involves an extensive table lookup searching, which is a time consuming process both in scalar and vector processing. The table lookup process here is really the construction of samples from arbitrary discrete distributions.

5.4.1 Main Modules

The MORSE code modules are shown in Figure 5.1. The *source module* generates source particles. Then, the *random walk routines* generate particle histories and do the major bookkeeping. The *combinatorial geometry routine* keeps track of particle positions with the geometry domain, and identifies the corresponding material. The

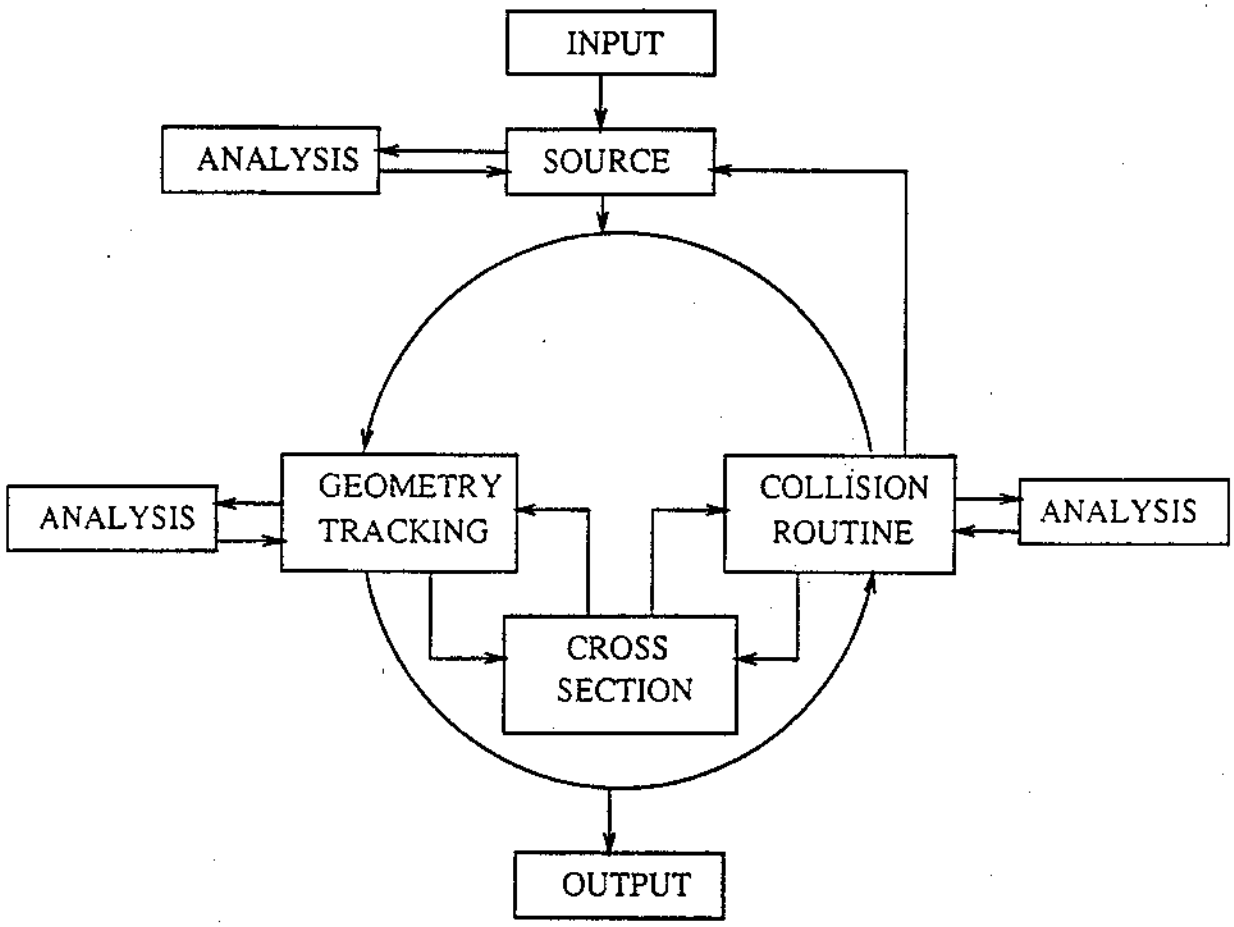

Figure 5.1: MORSE code modules.

collision module determines the energy loss and change of direction of a particle after a collision. The probability tables for collision and tracking processes are provided by the *cross section module*. The *analysis module* is used to estimate the contribution of each random walk to the parameters of interest.

5.4.2 Probability Tables

This subsection describes briefly the construction of the scattering probability matrix from the group-to-group energy transfer cross sections discussed in Subsection 5.2.1. The angular scattering probability tables, used in the MORSE code for sampling the outcome of a collision, are also discussed.

Scattering Probability Tables

The scattering cross section is the summation of group-to-group energy transfer cross section. That is

$$\sigma_s^g = \sum_{g'=1}^{G} \sigma_{g,g'}, \text{ for } g' = 1, 2, \cdots, G. \tag{5.10}$$

The probability of scattering from energy group g to g' is simply $\sigma_{g,g'}/\sigma_s^g$. The scattering probability table for energy group g is formed by the transfer probabilities from energy group g to all other possible energy groups. The scattering probability matrix is a set of the scattering probability tables from the first to the last energy group. If up-scattering (energy gain) does not occur, then the scattering probabilty matrix is an upper right triangular matrix. The effect of a collided nucleus on the length of a probability table in an elastic scattering is discussed below.

Following an elastic scattering, the outgoing neutron energy (E') lies in the interval defined by [59]

$$\alpha^2 E \leq E' \leq E, \tag{5.11}$$

where E is the incoming energy, and $\alpha = (A-1)/(A+1)$, with A is the ratio of the target mass to the neutron mass. A can be approximated by the mass number

of the target nucleus [59].

According to Equation (5.11), if A is relatively large, then the minimum energy of an outgoing neutron is close to its maximum energy; i.e. the energy interval corresponding to a single collision is narrow. This means that an outgoing neutron does not lose much of its energy after a collision. In this case, only a short probability table is required since the outgoing energy group will not be far from the incoming energy group. On the contrary, when A is relatively small the energy interval of an outgoing neutron is wide. A neutron can lose most (all if $A = 1$) of its energy in one collision. Therefore, a long probability table is needed since the outgoing energy group can be very far from the incoming energy group (see Section 5.7 for further discussion). Thus, the length of a scattering probability table is affected by the mass number of the target nucleus.

Angular Scattering Probability Tables

In isotropic scattering, an outgoing neutron scatters with equal probability into any direction. For anisotropic scattering however, probability tables are required to sample scattering angles. The angular scattering probability table is derived as shown below.

Practical applications require the probability of scattering to be given in terms of the scattering angle. The probability of elastic scattering from an energy E to an energy E' with an angle $\cos^{-1}\mu$ can be expressed as [40]

$$\begin{aligned} p(E \to E';\mu) &= \frac{\sigma_s(E \to E';\mu)}{\sigma_s(E')} \\ &= \frac{1}{4\pi\sigma_s(E')} \sum_{l=0}^{L} S_l(E \to E') P_l(\mu), \end{aligned} \tag{5.12}$$

where S_l refers to the l-th Legendre coefficient and P_l to ordinary Legendre polynomial of order l. In a multigroup cross section structure, where energy E and energy E' lie, respectively, within group g and g', then $\sigma_s(E \to E')$ corresponds to $\sigma_{g,g'}$, while $\sigma_s(E')$ becomes $\sigma_s^{g'}$. Note that the use of Legendre polynomial of order $(2n-1)$

results in the construction of an angular probability table for n discrete angles [39]. Since these polynomials are related to the distribution moments and subsequently the Legendre coefficients, only n angles can be determined for $(2n-1)$ coefficients.

5.5 Speeding Up of MORSE Computations

A Monte Carlo code, such as the MORSE code, generally requires a large amount of computer time in order to obtain results with small statistical variability. There are, basically, two approaches to speeding up the simulation time of such a code. The first approach is to modify the statistical methods, such as the scoring method (the estimators) and the sampling method. We can also bias the problem using importance sampling techniques such that particles are directed to important zones, directions or energies. The second approach is to reduce the processing time by employing vector and/or parallel processing. As mentioned earlier in the previous chapters, we consider the use of the weighted sampling method and vector processing in order to speed up the computations. This section discusses the vector processing for Monte Carlo particle transport codes, while the sampling methods are investigated in the following section.

With the advent of vector and parallel computers, many efforts are directed towards employing such computers to reduce the processing time of particle transport simulations. Monte Carlo particle transport codes have been implemented on vector and/or parallel computers at the University of New Brunswick [7,80,81,90,91,92,104], and elsewhere [10,13,14,47,61,62,63].

The vectorizability of Monte Carlo particle transport code can be enhanced using two approaches. The first approach globally restructures the entire code; whereas, the second approach tries to eliminate the vectorization inhibitors in the code. Vector computers provide compilers for vectorization. However, such compilers cannot exploit vectorization of many Monte Carlo particle transport codes, since most parts

of the codes contain implicit loops and random walk computation [10]. Consequently, the codes have to be *rewritten* with global transformations to make them vectorizable. The vectorization inhibitors have to be removed as well in order to achieve high vectorizability.

The notion of restructuring particle transport codes is as follows. A vector code uses the event-based algorithm (discussed in Section 4.6) in which a set of particles are simulated simultaneously from one event to another. Events could be the particle generation, collision and boundary crossing procedures in a particle transport algorithm. Note that this algorithm is different with the history based algorithm, which simulates only one particle at a time from birth to death.

The vectorized code is designed to follow many particles through their random walks simultaneously. This is done by forming particle vectors and then applying the same operations to all the particles in the vector using vector instructions to speed up the computation rates. Thus, vector processing can exploit the advantage of executing identical operations on contiguous data elements (vectors).

In the Monte Carlo particle transport method, the geometry algorithm must be able to track particles throughout the system and relate the positions of particles to the materials encountered. We found that this process can be time consuming, and contains vectorization inhibitors. Therefore, an alternative geometry algorithm should be used to speed up the computation.

In the previous chapter, it was demonstrated that the weighted sampling method could enhance the vectorizability of the event-based Monte Carlo codes. As a first step, we investigate the statistical as well as the processing time effects of incorporating the weighted sampling method into the scalar MORSE code. We further examine the effect of vectorizing only the parts in which sampling is performed, since the compiler cannot exploit the vectorization of the MORSE code in its present state. Thus, the objective here is to investigate local speedups, which can be introduced to the MORSE code.

5.6 Table Lookup

MORSE requires an extensive table lookup searching, particularly to sample an outgoing neutron energy and direction at every collision. The table lookup process can be viewed as constructing samples based on discrete distributions, whose probability tables are stored in arrays. The inverse method [56] is usually used to construct samples from a probability table. As discussed in Chapter 2, this method is not efficient for table lookup computation since it involves comparisons and requires a preprocessed table. Also, this method is not suitable for vector processing as it involves linear (step-by-step) searching.

Since the table lookup computation can significantly affect to the simulation time, efforts have been made to develop efficient algorithms for the table lookup process in Monte Carlo particle transport codes by utilizing vector computers [14,92].

Brown [14] proposed a vectorizable algorithm for Monte Carlo codes. Brown's method requires only one comparison to construct a sample, regardless of the length of the probability table. This method, however, requires a complex procedure to set up tables for sampling. For each energy group the table for sampling also requires an array of size $3n$, where n is the array size of the probability table (see Section 2.6 for the details). Consequently, a large memory size and a significant amount of time are required to prepare tables for sampling from large scattering probability matrices, which are often used in neutron transport codes; such as the MORSE code.

Tassou et al. [92] implemented a vectorized Monte Carlo neutron transport, COM, on the Cyber-205 supercomputer. The initial scalar and the vector version of the COM programs utilized functional fits for neutron cross sections and elastic isotropic scattering kinematics in the center-of-mass system to calculate the energy of collided particles. In order to investigate the effect of the table lookup process, a 36-group neutron cross section table was introduced in the scalar as well as the vector COM codes. The incorporation of this table lookup computation was found to slow down

Table 5.2: An illustration of a scattering cumulative probability matrix.

		New Energy Group								
		1	2	3	4	5	.	.	.	G
	1	0.30	0.40	0.70	0.87	0.93	.	.	.	1.0
	2		0.20	0.40	0.55	0.78	.	.	.	1.0
Old	3			0.30	0.50	0.76	.	.	.	1.0
	4				0.60	0.78	.	.	.	1.0
Energy	5					0.72	.	.	.	1.0
	.						.	.	.	1.0
Group	.							.	.	1.0
	.								0.87	1.0
	G									1.0

the scalar computation by about 12 %, while the vector processing was slowed down by about 87 %. The vectorization speedup therefore decreases from about 16 (without table lookup) to only 9.6 (with table lookup).

A table lookup process implemented using the inverse method exists in MORSE. The inverse method requires cumulative probabilities, which are prepared before initiating the random walk process. An illustration of a scattering cumulative probability matrix is shown in Table 5.2. The matrix is upper right triangular since only down-scattering is allowed.

There is a need for an alternative sampling method for table lookup, which can enhance the vectorizability of Monte Carlo codes and can utilize directly the scattering and the angular scattering probability tables. For this purpose, the weighted sampling method is incorporated into the MORSE code. As a first step, the performance of the sequential MORSE code is examined for three problems with different physical characteristics. Brown's method is also implemented. The statistical results and the processing time required for each method are then compared.

It should be noted here that zero probabilities in the scattering and the angular scattering probabilities must be excluded in constructing Brown's tables for sampling. In the existing MORSE codes, the specified lengths of probability tables include

zero probabilities. The MORSE codes, therefore, must be modified to exclude zero probabilities.

In the MORSE code, sampling for an outgoing neutron energy group and direction is done in a subroutine called COLISN. The incorporation of the code for the weighted sampling method in subroutine COLISN is simple. The next energy group and the scattering direction are chosen uniformly and the particle's weight is then adjusted. After the outgoing group is sampled, the particle weight is adjusted by multiplying its value by the product of the scattering probability of the selected group and the length of the corresponding probability table excluding zero probabilities. A similar adjustment is also done after sampling for an outgoing scattering angle.

In the following sections, we examine the applications of the MORSE code for solving three problems with different physical characteristics. The objective is to analyze the effect of the inverse method, Brown's and the weighted sampling methods on the solutions and processing time. Each problem is solved using three different versions of MORSE codes, incorporating the three different sampling methods, viz. the inverse method, Brown's and the weighted sampling methods.

5.7 Problem I

This problem is one of the sample problems distributed with the MORSE code [66]. In this problem, MORSE calculates the fast-neutron fluence at several radial distances from a point isotropic fission source in an infinite medium of air (see Figure 5.2). The transport medium air is assumed to consist of only oxygen and nitrogen with a total density of 1.29 grams/liters.

In this problem we examine the fluence at three shells, which are the innermost shell, a middle shell, and the outermost shell; their radii are 30, 200 and 450 meters, respectively. The objective here is to analyze the performance of diffferent sampling methods on the estimated values of neutron fluence of several distance.

MORSE uses the combinatorial geometry package to describe thirteen concentric spherical shells of air surrounding the point source. The neutron fluence crossing the spherical shells is to be estimated. For this purpose, boundary crossing estimators are used. In this estimator, the value of the weight of a particle crossing a shell divided by the absolute value of the cosine of angle between the normal to the surface and the direction of an incident particle is accumulated. Then, the average of these values is used to estimate the neutron fluence.

A cross section library consisting of 22 neutron energy groups is employed in the MORSE simulation. The simulation uses five Legendre coefficients, which results in three scattering angles for each angular probability table. This sample problem analyzed only the top 13 neutron groups; i.e. neutrons were followed until their energy dropped below the value corresponds to group 13.

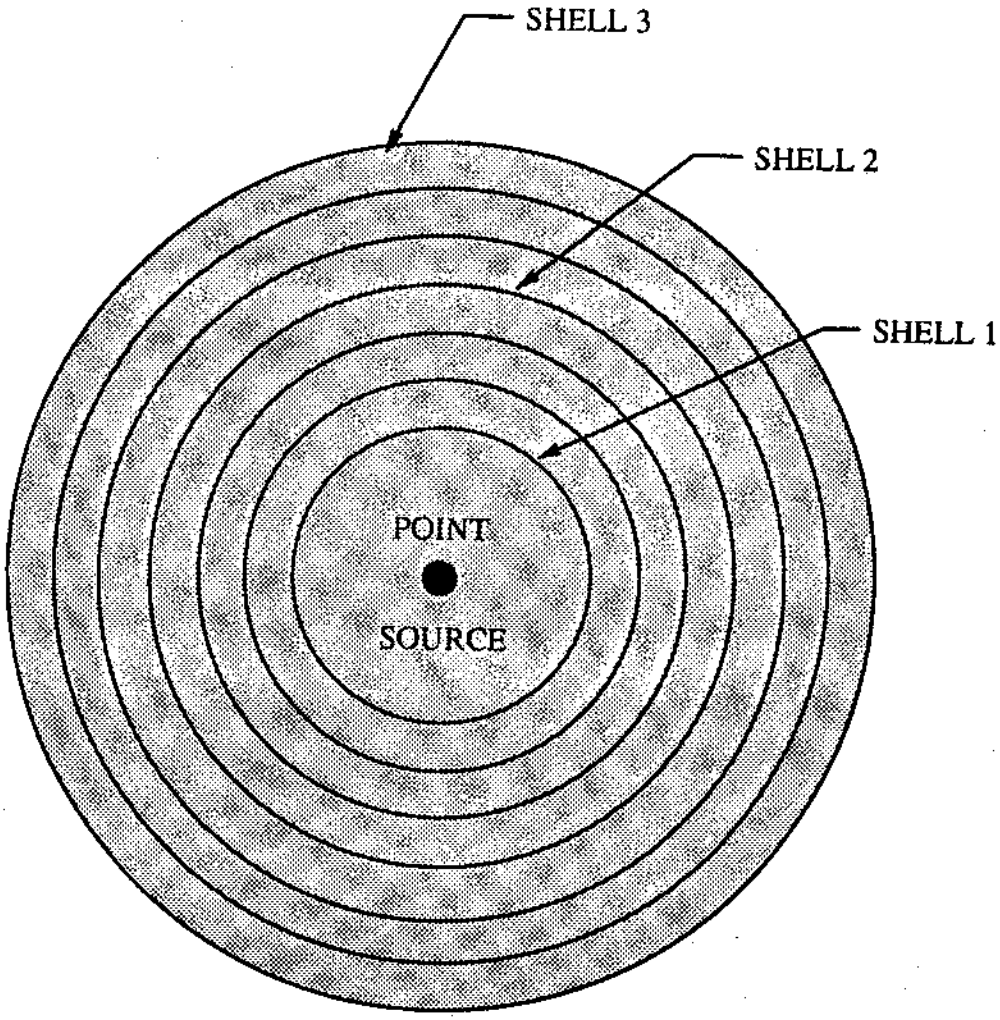

Figure 5.2: An illustration of MORSE Problem I: A point source is surrounded by concentric spherical shells. The radii of Shell 1, Shell 2 and Shell 3 are 30, 200 and 450 meters, respectively.

Table 5.3: The scattering probability matrix of air for Problem I.

Incoming energy group	Outgoing energy group												
	1	2	3	4	5	6	7	8	9	10	11	12	13
1	.3791	.2414	.0535	.0503	.0593	.0445	.0560	.0292	.0061	.0264	.0311	.0168	.0065
2		.3928	.2664	.0607	.0429	.0407	.0613	.0330	.0070	.0305	.0366	.0202	.0080
3			.4106	.3386	.0339	.0276	.0420	.0304	.0063	.0468	.0437	.0150	.0052
4				.4969	.4007	.0146	.0202	.0117	.0025	.0113	.0198	.0154	.0068
5					.5150	.4087	.0528	.0042	.0012	.0053	.0069	.0041	.0017
6						.5146	.4754	.0000	.0001	.0015	.0044	.0029	.0013
7							.5757	.3669	.0430	.0109	.0004	.0016	.0014
8								.4243	.1600	.4142	.0002	.0002	.0011
9									.1507	.8218	.0276	.0000	.0000
10										.5076	.4924	.0000	.0000
11											.7315	.2685	.0000
12												.8099	.1901
13													1.0000

Table 5.4: Some scattering angles and their corresponding cumulative probabilities of air for Problem I.

Group-to-Group	CP	μ	CP	μ	CP	μ
1→1	0.7865	0.9397	0.9668	0.5575	1.0000	-0.0979
1→2	0.5487	0.0062	0.8108	0.7800	1.0000	-0.7469
1→3	0.5245	-0.8781	0.8266	-0.1151	1.0000	0.7531
2→2	0.7623	0.9372	0.9565	0.5366	1.0000	-0.0867
2→3	0.5572	-0.0266	0.7878	0.7717	1.0000	-0.7598
2→4	0.5086	-0.8696	0.8209	-0.1124	1.0000	0.7538
3→3	0.7502	0.9322	0.9443	0.5205	1.0000	-0.1146
3→4	0.5518	-0.0794	0.8193	-0.7910	1.0000	0.7613
3→5	0.4166	-0.0266	0.7401	-0.8354	1.0000	0.7687

The mass numbers of oxygen and nitrogen are 16 and 14, respectively. With these mass numbers, Equation (5.11) results in a relatively small interval of possible outgoing neutron energy. This means that the outgoing neutron loses only a small portion of its energy after each collision. Therefore, the probability table for each group has high probabilities for the first few subsequent groups and low probabilities for the rest of the groups. Table 5.3 shows the scattering probability matrix of air, which was produced by the MORSE code. The matrix is upper right triangular since only down-scattering occurs during energy transitions.

Table 5.4 presents a part of the angular scattering tables used by MORSE in Problem I. It shows the cosines of scattering angles (μ) and the corresponding cumulative probabilities (CP) for energy groups 1, 2 and 3. For each group-to-group of energy change, there are three scattering angles preceded by their cumulative probabilities, which are accumulated from the probability of the first angle to the probability of the third angle.

5.7.1 Fluence Estimates

This subsection reports results of Problem I, obtained by employing different sampling methods. The notations for the computer codes corresponding to each method are:

1. INVER: original MORSE code which uses the inverse sampling method,
2. DISCS: MORSE code with Brown's method, and
3. WGHTS: MORSE code with the weighted sampling implemented such that random samples are constructed directly from the original probability tables, and subsequently multiplied by the adjustment factors, (this is Technique I in Chapter 3).

In the discussion, the results obtained by INVER code are used as a reference, since this method is the original method employed in MORSE. The results are analyzed based on convergence to an unbiased solution, statistical variability, processing time, and efficiency. The estimated fluences at three spherical shells, obtained using the three sampling methods, are given in Tables 5.5 to 5.8. The estimated fluence is given together with the fractional standard deviation and the variability (statistical error) associated with 99 % confidence level in the results. The corresponding confidence interval is defined by

$$\overline{X} \pm 2.576\sqrt{\frac{S^2}{k}} \tag{5.13}$$

where k is the sample size; $\overline{X}$ and S^2 denote the sample mean and sample variance, respectively. A high confidence level was chosen here since these intervals are used to determine whether the estimated results are biased or not, in comparison with the corresponding estimates obtained using INVER.

The neutron fluence consists of two components, an uncollided component and a collided component. The uncollided fluence results from neutrons emanating from the source and reaching the shell directly without encountering any collisions with the intervening medium. The collided fluence, on the other hand, results from neutrons having experienced collisions before crossing the shell.

Table 5.5: Uncollided fluence of Problem I.

Sample size	INVER	DISCS	WGHTS
	Estimate(FSD)	Estimate(FSD)	Estimate(FSD)
20,000	7.1658E-1(0.00707)	7.1569E-1(0.00707)	7.1568E-1(0.00707)
	±0.131E-1*	±0.130E-1	±0.130E-1
40,000	7.1592E-1(0.00500)	7.1546E-1(0.00500)	7.1516E-1(0.00500)
	±0.092E-1	±0.092E-1	±0.092E-1
60,000	7.1588E-1(0.00408)	7.1519E-1(0.00408)	7.1501E-1(0.00408)
	±0.075E-1	±0.075E-1	±0.075E-1
80,000	7.1571E-1(0.00354)	7.1502E-1(0.00354)	7.1498E-1(0.00354)
	±0.065E-1	±0.065E-1	±0.065E-1
100,000	7.1381E-1(0.00316)	7.1322E-1(0.00316)	7.1315E-1(0.00316)
	±0.058E-1	±0.058E-1	±0.058E-1

the analytical solution of uncollided fluence= 7.1762E-01.
* statistical variability corresponds to a 99 % confidence interval.

The total fluence is the summation of the uncollided and collided components. The estimated values of uncollided fluence crossing the innermost concentric spherical shell are shown in Table 5.5. These values can be verified against analytical solutions, which is an exponential relationship given by the following equation

$$\text{Uncollided fluence} = \sum_{g=1}^{G} f_g e^{-\Sigma_t^g r}, \tag{5.14}$$

where

f_g = fraction of source neutrons emitted in energy group g,

r = radius of spherical shell,

G = last energy group,

Σ_t^g = total cross section at energy group g.

The above equation results in a value of 7.1762E-01.

In Table 5.5, an increasing number of source particles (sample size) is utilized, in order to examine the convergence of the solutions. One would expect that the estimated fluence remains relatively unchanged as the sample size increased. One can notice, however, that for all sampling methods, the estimated uncollided fluence remains almost constant for sample sizes from 20,000 to 80,000 particles. As the sample size increases to 100,000 particles, the estimated fluence in the three sampling methods decreases slightly.

In order to explain the above discrepancy, let us examine the variability associated with each estimate, as defined by the confidence interval. One can see that, for all sampling methods, the FSDs decrease as the sample size increases. This indicates convergence to the solution. With the increase in sample size from 80,000 to 100,000, the estimated values in the three sampling methods slightly decrease, at most by 0.019E-1. This difference however is less than the statistical variability, which is equal to 0.058E-1; hence the fluctuation in the estimated values for uncollided fluence is still within the confidence intervals. Thus, the estimated values converge to the expected solution.

It is seen in Table 5.5 that the magnitude of the analytical solution is inside the confidence intervals estimated by different sampling methods. Thus, the estimated uncollided fluence obtained by employing the three sampling sampling methods are unbiased.

It is important to notice in Table 5.5 that the three sampling methods result in values that are very close in magnitude to each other. The trend of convergence

is similar. This is, however, expected since the uncollided fluence depends only on source particles, and is not directly affected by the particular sampling method used. This is because the sampling method affects only particle collisions and in turn the collided component of the fluence. The only influence of the sampling method is in changing the sequence of random numbers at which source particles are selected. However, this is smoothed out as the sample size increased to a few thousands. A similar trend is observed for the other two shells.

Table 5.6: Total fluence crossing Shell 1 (30 m in radius) of Problem I.

Sample size	INVER	DISCS	WGHTS
	Estimate(FSD)	Estimate(FSD)	Estimate(FSD)
20,000	1.3774(0.01751)	1.3406(0.01489)	1.3435(0.02274)
	±0.062*	±0.051	±0.079
40,000	1.3752(0.01089)	1.3276(0.01028)	1.3253(0.01474)
	±0.039	±0.035	±0.050
60,000	1.3787(0.01198)	1.3153(0.00812)	1.3447(0.01734)
	±0.043	±0.028	±0.060
80,000	1.3649(0.01135)	1.3131(0.00788)	1.3296(0.01443)
	±0.040	±0.027	±0.049
100,000	1.3596(0.00970)	1.3117(0.00702)	1.3317(0.01320)
	±0.034	±0.024	±0.045

* statistical variability corresponds to a 99 % confidence interval.

Tables 5.6 to 5.8 show the estimates obtained for the total fluence crossing three different cocentric spherical shells, respectively. By comparing the estimated values with those of the uncollided fluence reported in Table 5.5, one can directly conclude that the uncollided fluence constitutes only a small portion of the total fluence. The collided component therefore dominates the total fluence. Hence one would expect the sampling method to directly affect the estimated values of total fluence.

One can notice in Table 5.6 that the estimated fluence converges, as indicated by the decrease in FSDs, as the sample size increases. The estimated values for the three sampling methods converge to about the same value. This is indicated by the fact that the intervals obtained by DISCS and WGHTS overlap those of INVER. This

means that DISCS and WGHTS also result in unbiased estimates, since the INVER's solutions are used as a reference.

Table 5.7: Total fluence crossing Shell 2 (200 m in radius) of Problem I.

Sample size	INVER Estimate(FSD)	DISCS Estimate(FSD)	WGHTS Estimate(FSD)
20,000	2.1078(0.02494)	1.9685(0.02012)	1.9839(0.11728)
	±0.135*	±0.102	±0.599
40,000	2.0480(0.01779)	2.0049(0.01638)	1.8611(0.06882)
	±0.094	±0.085	±0.330
60,000	2.0496(0.01480)	2.0185(0.01610)	1.8640(0.05616)
	±0.078	±0.084	±0.270
80,000	2.0671(0.01427)	2.0348(0.01502)	1.8510(0.04713)
	±0.076	±0.079	±0.225
100,000	2.0776(0.01448)	2.0460(0.01380)	1.8137(0.03985)
	±0.078	±0.073	±0.186

* statistical variability corresponds to a 99 % confidence interval.

Table 5.8: Total fluence crossing Shell 3 (450 m in radius) of Problem I.

Sample size	INVER Estimate(FSD)	DISCS Estimate(FSD)	WGHTS Estimate(FSD)
20,000	1.1906(0.03296)	1.1940(0.03408)	1.0298(0.20122)
	±0.1011*	±0.1048	±0.5338
40,000	1.1855(0.02305)	1.1845(0.02457)	0.9170(0.14068)
	±0.0704	±0.0750	±0.3323
60,000	1.1712(0.01868)	1.1607(0.01925)	0.9319(0.11716)
	±0.0564	±0.0576	±0.2813
80,000	1.1741(0.01810)	1.1682(0.01793)	0.8849(0.09502)
	±0.0547	±0.0540	±0.2166
100,000	1.1751(0.01612)	1.1623(0.01575)	0.9123(0.07903)
	±0.0488	±0.0472	±0.1857

* statistical variability corresponds to a 99 % confidence interval.

From Table 5.6, for Shell 1, the FSDs of DISCS's results are slightly smaller than those of INVER, while the FSDs of estimates obtained using the weighted sampling are about 1.3 times larger than those of INVER. This large variance can be explained

by the fact that the adjustment factors employed by the weighted sampling method vary significantly. Subsequently, the weights of collided neutrons are affected, and their values have a larger variability. The weighted sampling results (WGHTS) are still, however, unbiased since they lie within the bound of the 99 % confidence intervals.

Table 5.7 shows the results for Shell 2. Here one can notice that the solutions of INVER and DISCS converge to about the same value, whereas the WGHTS's solutions converge to a smaller value. However, the 99 % confidence intervals of WGHTS still overlap those of INVER. One can conclude therefore that the weighted sampling results are still unbiased estimates of the neutron fluence.

The total fluence of neutrons crossing the outermost shell with radius of 450 meters is given in Table 5.8. Here one can notice that the values of the estimated fluence obtained using INVER and DISCS are close to each other for all sample sizes. Their solutions converge to almost the same value. The WGHTS code results in a slightly lower value than those of the other two methods. However, the confidence intervals of WGHTS overlap those of INVER for sample size 20,000, 40,000 and 60,000, as shown in Table 5.8. For sample sizes of 80,000 and 100,000 the confidence intervals of WGHTS do not overlap those of INVER. The difference is at most 0.028, which is about 2.4 % relative to the INVER's estimate.

The smaller estimated values obtained by WGHTS can be explained by the following reason. The weighted sampling utilizes uniform distributions to sample the outgoing energy groups and the scattering angles, while the other codes use the original scattering probability matrix and the original angular scattering tables. Therefore, the random walks in WGHTS have higher probabilities of reaching lower energy groups than those in the other codes. This leads to earlier termination of the random walks in WGHTS, due to the premature reaching of the energy cut-off. Therefore, the number of neutrons reaching energy cut-off in the weighted sampling is more than in the other two sampling methods. This is indicated by the fact that the weighted

sampling produces fewer collisions, as shown in Table 5.10. Consequently, a shell with a large radius receives less accumulated fluence, hence the average values (estimated fluence) are smaller.

Comparing the values of total fluence of the three shells (see Tables 5.6 and 5.8) one can notice that the fluence for Shell 2 is higher than that of Shells 1 and 3. This is not what one generally expects, that is the farther away the spherical shell from the source the lower is the fluence. This behaviour can be explained by the fact that Shell 2 receives particles scattered backward towards the source, which results in an increased collided fluence.

The FSDs of estimates obtained using the three sampling methods increase as the shell radius increases. This is also due to the fact that the variability of the statistical weights of a neutron increases as the distance from the source increases, since the energy of a neutron varies significantly during the travel for a long distance.

5.7.2 Processing Time and Speedups

Here, we examine the execution time (cpu time) required for the three sampling methods. Table 5.9 shows the processing time required by the different scalar MORSE codes, namely INVER, DISCS and WGHTS, executed in scalar mode. Five different sample sizes are reported in this table since the solutions shown in the previous tables are obtained with these sample sizes. Different number of samples is also useful for recognizing the pattern of processing time. The processing time of DISCS is about 12 % less than that of INVER; whereas WGHTS entails the least processing time, which is about 32 % of the processing time required by INVER.

The number of collisions resulting in each sampling method directly affect its processing time. This is due to the fact that each collision requires computation for the tracking of collided particles, and therefore a larger number of collisions requires more processing time. Further, it is found that for each sampling method the scalar processing time is linearly proportional to the number of collisions. For each collision,

the processing time required by the various sampling methods is close to each other.

The number of collisions occured during Monte Carlo simulations is reported since they are affected by the sampling methods used. The number of collisions produced by different sampling methods is shown in Table 5.10. It can be seen that for each sampling method, an increase in sample size by a factor of two produces an increase in the number of collisions by a factor about two. That is, there is a linear relationship between the number of collisions and the sample size.

As shown in Table 5.10, the DISCS produces about 87 % of INVER's collisions, while WGHTS produces only about 26 %. As discussed before, the random walks in WGHTS have higher probabilities of reaching lower energy groups than those in the other codes, due to the utilization of uniform distributions. Consequently, more random walks were terminated earlier as they reach the energy cut-off. The WGHTS code therefore produces fewer collisions.

The scalar speedups of the MORSE codes incorporating Brown's method (DISCS) and the weighted sampling method (WGHTS) relative to the MORSE code implementing the inverse method (INVER) are given in Table 5.11. One can notice that the speedups for each sampling method is constant as the sample size increases. This is due to the linear relationship between the sample size and the processing time required by each sampling method. On average, DISCS is about 13 % faster than the existing INVER, while the WGHTS is about three times faster. This emphasizes

Table 5.9: Scalar processing time of MORSE simulation for Problem I.

Sample size	Processing time in seconds		
	INVER	DISCS	WGHTS
20,000	79.278	68.291	25.117
40,000	155.973	135.996	49.674
60,000	228.802	206.976	75.793
80,000	305.631	273.029	101.569
100,000	378.767	340.185	125.696

Table 5.10: Number of collisions in MORSE simulation for Problem I.

Sample size	Number of collisions		
	INVER	DISCS	WGHTS
20,000	384,994	330,833	99,118
40,000	761,302	663,336	197,157
60,000	1142,531	995,898	295,382
80,000	1521,797	1329,106	392,965
100,000	1903,822	1662,812	490,279

Table 5.11: Speedups of different scalar MORSE codes relative to scalar INVER for Problem I.

Sample size	Scalar speedups	
	DISCS	WGHTS
20,000	1.161	3.156
40,000	1.147	3.140
60,000	1.105	3.019
80,000	1.119	3.009
100,000	1.113	3.013

the attractiveness of using the weighted sampling method to speed up Monte Carlo calculations.

5.7.3 Local Speedups

In the previous subsection, the processing time of the entire MORSE code incorporating different sampling methods was examined. This section concentrates on examining the processing time of different sampling methods in the subroutine used to handle collision process (subroutine COLISN). Thus, the objective here is to obtain the local speedups of different sampling methods as they are incorporated into subroutine COLISN. This local speedup provides a direct measure of the speeding of the sampling method, without the additional overhead of other routines in the MORSE code.

Table 5.12: Scalar processing time of sampling methods in COLISN routine for Problem I.

Sample size	Processing time in milliseconds		
	INVER	DISCS	WGHTS
20,000	1263.860	1332.077	1251.416
40,000	2530.249	2660.251	2509.246
60,000	3791.720	3994.551	3756.862
80,000	5054.139	5334.382	5006.014
100,000	6318.385	6656.959	6258.126

The scalar processing time required by the three sampling methods is reported in Table 5.12. The processing time of these sampling methods has the same trend. That is, the processing time of each sampling method is linearly proportional to the sample size.

For all sample sizes, the least scalar processing time is required by WGHTS, while the largest one is needed by DISCS. The processing time of INVER is close to that of WGHTS. This indicates that the inverse method performs well here, since it requires only relatively few comparisons for each sampling to locate an outgoing energy group. This is due to the large probabilities for the first few energy groups and low probabilities for the subsequent energy groups in the scattering probability

tables used by the inverse method (see Table 5.3) for this problem.

On average, the processing time of scalar DISC and WGHTS relative to that of INVER are about 105 % and 99 %, respectively. Therefore, the time required by different sampling methods to construct a sample is about the same. This fact supports the argument, discussed in Subsection 5.7.2, that the processing time for each collision in different sampling methods is about the same. Thus, the global speedups achieved by the MORSE codes incorporating Brown's and the inverse sampling methods, shown in Table 5.11, are mostly affected by the reduction in the number of collisions.

5.7.4 Local Vectorization Speedups

The vectorization speedups of MORSE codes incorporating different sampling methods cannot be obtained, since the entire MORSE code in its present state is not fully vectorizable. One can examine however the effect of vectorization on local speedups, since subroutine COLISN is vectorizable. This local speedup should provide some indications of the global vectorization speedup one could attain, if the entire MORSE code is vectorized after implementing an event-based Monte Carlo algorithm, discussed in Section 4.6.

Table 5.13: Vector processing time of sampling methods in COLISN routine for Problem I.

Sample size	Processing time in milliseconds		
	INVER	DISCS	WGHTS
20,000	364.371	150.948	136.364
40,000	729.298	301.443	273.995
60,000	1095.980	452.000	412.343
80,000	1460.923	604.338	545.436
100,000	1824.385	753.166	684.846

Table 5.13 reports the processing time of vectorized INVER, DISCS and WGHTS. One can notice that for all sample sizes the least processing time is for WGHTS,

Table 5.14: Speedups of the vectorized sampling methods relative to the scalar inverse method for Problem I.

Sample size	Vector speedups		
	INVER	DISCS	WGHTS
20,000	3.469	8.373	9.177
40,000	3.469	8.394	9.158
60,000	3.460	8.389	9.111
80,000	3.460	8.363	9.178
100,000	3.463	8.389	9.138

followed by DISCS and INVER. Note that the vector INVER requires the largest processing time, which is not the case of the scalar processing. The processing time of vector DISCS and WGHTS is only about 41 % and 38 % of that of vector INVER. These results are in contrast with those of the scalar processing time. This indicates that the vectorizability of the weighted sampling method is the best.

The processing time of the vector code relative to that of the scalar code is usually calculated in order to provide speedup factors. As shown in Table 5.14, the speedups of different vector codes (INVER, DISCS and WGHTS) are reported relative to scalar INVER code. This table shows that for different sample sizes the speedups of the three sampling methods are about constant. The WGHTS code provides the highest speedup, about 9.1, while INVER attains the lowest speedup of about 3.5. The DISCS entails speedup about 8.4.

5.7.5 Efficiency

In the preceding two subsections, the statistical and the computational performance have been separately examined. These two performace criteria can be combined by considering the efficiency of the Monte Carlo solution. As discussed in Section 2.7, the efficiency of a Monte Carlo solution is inversely proportional to the product of the sample variance and the processing time. Therefore, the efficiency of an estimate obtained by WGHTS relative to that obtained using INVER is the multiplication of the sample variance and the processing time of INVER divided by the multiplication of the sample variance and the processing time of WGHTS (see Equation 2.4).

Table 5.15: Efficiencies of different MORSE codes relative to those of INVER, for Shell 1 of Problem I.

Sample size	Relative Efficiency	
	DISCS	WGHTS
20,000	1.695	1.967
40,000	1.381	1.845
60,000	2.644	1.515
80,000	2.509	1.962
100,000	2.284	1.696

Tables 5.15 to 5.17 show the efficiencies of DISCS and WGHTS relative to those of INVER for Shells 1, 2 and 3, respectively. Table 5.15 shows that the relative efficiencies of DISCS and WGHTS, for different sample sizes, are not constant. For all sample sizes, the relative efficiencies of DISCS are higher than its scalar speedups over INVER, given in Table 5.11, while those of WGHTS are lower than its scalar speedups. The reason is that DISCS results in lower sample variances; whereas, the increase in sample variances of WGHTS is higher than its scalar speedups.

For a sample size of 60,000, the relative efficiency of DISCS increases due to the increase in the sample variance of INVER and the decrease in DISCS's sample variance. However, WGHTS has the lowest relative efficiency for this sample size,

since the increase in its sample variance is higher. On average, the efficiency of DISCS relative to that of the inverse method is about 2.1, while the relative efficiency of the weighted sampling is about 1.8. The relative efficiency of the weighted sampling is lower than its scalar speedup, since its sample variance is larger than that of INVER.

Table 5.16 shows that, on average, the relative efficiencies of DISCS and WGHTS are about 1.35 and 0.3, respectively. The relative efficiency of DISCS is higher than its scalar speedup, since its sample variance is slightly smaller than that of INVER. On the other hand, the relative efficiency of WGHTS is much smaller than its scalar speedup, since its sample variance is much larger than that of INVER. However, one can notice that the relative efficiency of WGHTS increases as the sample size increases, since its sample variance converges faster than that of INVER.

The relative efficiencies of solutions for Shell 3 are reported in Table 5.17. The relative efficiencies of DISCS and WGHTS, on average, are about 1.1 and 0.16, respectively.

In summary, the relative efficiencies of DISCS and WGHTS decrease with respect to the increase in shell's radii. The relative efficiencies of WGHTS drop significantly since the increase in its sample variances is much larger than those of INVER. The relative efficiencies of WGHTS for Shells 2 and 3, reported in Tables 5.16 and 5.17, are less than one. This indicates that the scalar speedup achieved by the weighted

Table 5.16: Efficiencies of different MORSE codes relative to those of INVER, for Shell 2 of Problem I.

Sample size	Relative Efficiency	
	DISCS	WGHTS
20,000	2.045	0.161
40,000	1.412	0.254
60,000	0.963	0.253
80,000	1.043	0.344
100,000	1.264	0.522

Table 5.17: Efficiencies of different MORSE codes relative to those of INVER, for Shell 3 of Problem I.

Sample size	Relative Efficiency	
	DISCS	WGHTS
20,000	1.080	0.113
40,000	1.011	0.141
60,000	1.060	0.121
80,000	1.152	0.192
100,000	1.192	0.208

Table 5.18: Efficiencies of different MORSE codes relative to those of INVER, for about 5 % FSD of Problem I.

Shell	Relative Efficiency	
	DISCS	WGHTS
Shell 1	0.671	0.452
Shell 2	5.744	0.071
Shell 3	1.080	0.037

sampling method did not compensate for the increase in its sample variances. However, if an event-based vectorization is implemented for the entire MORSE code, one may expect that weighted sampling will result in higher efficiency than the inverse method, since the vectorization speedup of the weighted sampling code (shown in Table 5.14) is about 9.1.

The relative efficiencies shown in Tables 5.15 to 5.17 are calculated based on a given sample size. In some applications, one may desire an answer with a given acceptable value of FSD. For FSD of about 5 %, the efficiencies of solutions obtained by DISCS and WGHTS relative to those of INVER are summarized in Table 5.18. For Shells 1, 2 and 3, the relative efficiencies of DISCS are not constant. This indicates that the processing time as well as the sample variances of both INVER and DISCS do not have the same pattern. The relative efficiency of WGHTS however decreases as the shell's radius increases. The relative efficiencies for a given sample size also

decrease with increasing radius in WGHTS.

5.7.6 Remarks

This section examined the performance of MORSE codes incorporating different sampling methods for solving a simple problem of a point source in air. The statistical results, processing time, speedups and efficiencies of three sampling methods were discussed. It was found that weighted sampling results in unbiased solutions with the largest fractional standard deviations, and requires the least processing time. The MORSE code incorporating weighted sampling is three times faster than that incorporating the inverse method.

Two types of efficiencies have been discussed. The first type is obtained according to a given sample size, while the second is based on a given percentage of fractional standard deviation. Most of the relative efficiencies results from the weighted sampling codes are less than one, which indicate that their scalar speedups did not compensate for the increase in sample variances of their solutions.

In Problem I, a boundary crossing estimator has been used, and the intervening medium is air. The problem discussed in the next section has different physical characteristics, and a different type of estimator is employed.

5.8 Problem II

The geometry for this problem is depicted in Figure 5.3. A fast neutron beam is directed towards the test section, which is water in the form of a cylindrical body. The water density is 0.25 g/cm^3. The test section is surrounded by air and artificial external void, respectively. The air density is 1.29 g/l. The transport of neutrons is terminated when the neutrons reach external void. Two point detectors are located perpendicular to both the neutron beam and the axis of the cylindrical test section; they are in the air medium. It should be noted that the water is not contained in a

pipe, since the objective is to analyze the effect of water on the sampling methods. Water contains hydrogen which has some peculiar probability table structure, as explained in Section 5.4.2.

A neutron cross section library consisting of 33 energy groups with five scattering angles were used in the MORSE simulations. The scattering probability matrices for air and water media are shown in Tables 5.19 and 5.20, respectively. The size of these matrices is 33 by 33, and they are upper right triangular matrices in which the elements in each row are introduced starting from the main diagonal. The presentation of these matrices however is not in the form of 33 rows by 33 columns; the elements of each row start from the first column, in order to compress the storage space. Each row indicates that the first probability is for the probability of transition in the same energy group, while the subsequent probabilities are for the probabilities of scattering to the lower energy groups. Up-scattering after a collision is not permitted in this case.

The transport medium air is assumed to consist of oxygen and nitrogen, while the transport medium water consists of hydrogen and oxygen. The mass numbers of hydrogen, nitrogen and oxygen are 1, 14 and 16, respectively. Since the mass number of air is higher than that of water, the energy interval of a neutron corresponding to a single collision in the air is narrower than that in the water, according to Equation 5.11. Consequently, each row of the air scattering probability matrix indicates that only the first two energy groups have high probabilities, while the rest probabilities are low. In the scattering probability matrix of water, on the other hand, the energy groups having high probabilities are more than two.

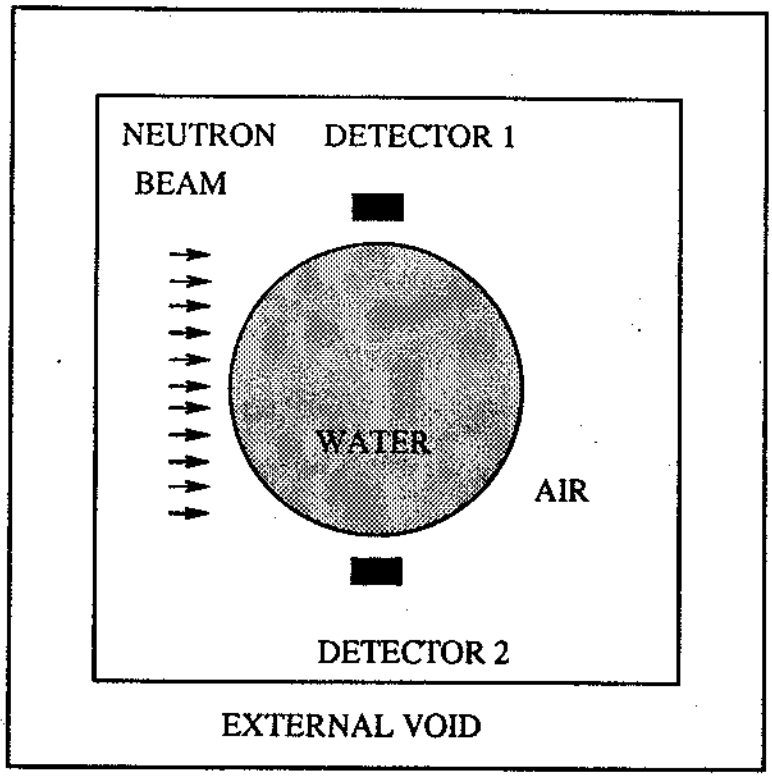

Figure 5.3: An illustration of MORSE Problem II.

		New Energy Group															
	1	.3975	.1867	.0886	.0257	.0554	.0389	.0616	.0404	.0302	.0149	.0151	.0110	.0072	.0162	.0087	.0013	.0006
	2	.3881	.3082	.0093	.0309	.0524	.0507	.0601	.0406	.0143	.0146	.0081	.0068	.0093	.0058	.0005	.0003	.0000
	3	.4645	.2828	.0326	.0144	.0411	.0535	.0425	.0157	.0153	.0099	.0073	.0111	.0038	.0024	.0033	.0000	.0000
	4	.4478	.3434	.0309	.0178	.0366	.0356	.0288	.0265	.0177	.0056	.0061	.0028	.0003	.0002	.0000	.0000	.0000
	5	.4600	.3888	.0492	.0076	.0278	.0187	.0095	.0112	.0119	.0106	.0038	.0003	.0005	.0000	.0000	.0000	.0000
	6	.4818	.4827	.0029	.0055	.0000	.0089	.0082	.0064	.0024	.0008	.0001	.0001	.0001	.0000	.0000	.0000	.0000
	7	.5472	.4431	.0007	.0017	.0025	.0009	.0000	.0027	.0012	.0001	.0000	.0000	.0000	.0000	.0000	.0000	.0000
	8	.5913	.4069	.0000	.0000	.0005	.0007	.0005	.0001	.0000	.0000	.0000	.0000	.0000	.0000	.0000	.0000	.0000
	9	.5310	.4288	.0400	.0000													
	10	.4303	.5697	.0000	.0000													
	11	.6277	.3723	.0000	.0000													
	12	.5809	.4191	.0000	.0000													
Old	13	.5210	.4790	.0000	.0000													
	14	.7815	.2185	.0000	.0000													
Energy	15	.8459	.1541	.0000	.0000													
	16	.6220	.3780	.0000	.0000													
Group	17	.8052	.1920	.0029	.0000													
	18	.4610	.5310	.0080	.0000													
	19	.4621	.5379	.0000	.0000													
	20	.7216	.2784	.0000	.0000													
	21	.7246	.2754	.0000	.0000													
	22	.7251	.2749	.0000	.0000													
	23	.8147	.1853	.0000	.0000													
	24	.8142	.1858	.0000	.0000													
	25	.8160	.1840	.0000	.0000													
	26	.8111	.1889	.0000	.0000													
	27	.8171	.1829	.0000	.0000													
	28	.8765	.1235	.0000	.0000													
	29	.8878	.1122	.0000	.0000													
	30	.8638	.1362	.0000	.0000													
	31	.8684	.1316	.0000														
	32	.9026	.0974															
	33	1.000																

Table 5.19: The scattering probability matrix of air for Problem II.

		New Energy Group																
	1	.234	.143	.140	.060	.093	.053	.070	.064	.051	.024	.021	.012	.010	.014	.009	.001	.002	.000
	2	.218	.281	.066	.061	.091	.071	.057	.052	.023	.022	.016	.011	.016	.009	.002	.002	.000	.000
	3	.237	.214	.088	.069	.095	.086	.069	.029	.032	.022	.016	.022	.014	.002	.003	.001	.000	.001
	4	.198	.233	.090	.104	.077	.070	.047	.054	.043	.024	.032	.018	.003	.004	.001	.001	.001	.001
	5	.170	.255	.150	.107	.079	.041	.048	.036	.030	.045	.026	.004	.005	.001	.001	.001	.001	.000
	6	.220	.304	.123	.091	.047	.056	.041	.031	.044	.028	.005	.006	.001	.001	.001	.001	.001	.000
	7	.296	.278	.110	.057	.067	.050	.037	.053	.033	.006	.007	.001	.001	.002	.001	.001	.001	.000
	8	.325	.239	.079	.092	.068	.051	.073	.046	.008	.010	.002	.002	.002	.001	.001	.001	.000	.000
	9	.199	.199	.159	.115	.085	.122	.076	.013	.017	.003	.003	.004	.002	.001	.001	.001	.000	.000
	10	.182	.310	.132	.098	.140	.088	.015	.019	.004	.003	.004	.003	.002	.001	.001	.000	.000	.000
	11	.276	.269	.118	.170	.106	.018	.023	.005	.004	.005	.003	.002	.002	.001	.000	.000	.000	.000
	12	.274	.315	.207	.129	.022	.028	.006	.004	.006	.004	.002	.002	.001	.000	.000	.000	.000	.000
	13	.235	.429	.213	.037	.046	.009	.007	.010	.006	.004	.003	.001	.001	.000	.000	.000	.000	.000
Old	14	.439	.375	.055	.069	.014	.011	.015	.009	.005	.004	.002	.001	.001	.000	.000	.000	.000	.000
	15	.422	.185	.207	.041	.032	.044	.027	.016	.013	.006	.003	.001	.001	.000	.000	.000	.000	.000
Energy	16	.220	.432	.077	.060	.083	.050	.031	.025	.012	.006	.003	.001	.001					
	17	.354	.157	.108	.150	.091	.055	.045	.021	.010	.005	.002	.001	.000					
Group	18	.153	.226	.244	.148	.090	.073	.035	.016	.008	.004	.002	.001	.000					
	19	.152	.362	.191	.116	.094	.045	.021	.010	.005	.003	.001	.000	.000					
	20	.261	.305	.171	.139	.066	.031	.015	.007	.004	.001	.000	.000	.000					
	21	.260	.305	.230	.109	.051	.024	.011	.007	.002	.001								
	22	.259	.401	.179	.085	.040	.019	.012	.004	.001	.000								
	23	.341	.354	.161	.076	.036	.023	.007	.002	.001	.000								
	24	.341	.354	.161	.076	.049	.014	.004	.001	.001									
	25	.341	.354	.161	.103	.029	.008	.003	.001	.001									
	26	.341	.354	.217	.062	.016	.006	.002	.001										
	27	.341	.474	.132	.034	.012	.005	.003											
	28	.465	.384	.095	.035	.013	.008												
	29	.466	.341	.122	.045	.026													
	30	.410	.377	.135	.078														
	31	.412	.376	.213															
	32	.424	.576																
	33	1.00																	

Table 5.20: The scattering probability matrix of water for Problem II.

Angular probability tables corresponding to the 33 energy groups are required to sample scattering angles, since the characteristic of scattering in this problem is anisotropic. Tables 5.21 and 5.22 show some angular scattering probabilty tables for air and water, respectively. Each table (row) gives the probabilities of five scattering angles.

Table 5.21: Partial angular scattering probability tables of air for Problem II.

Group-to-Group	Probabilities				
8→8	0.3458	0.3253	0.1586	0.1300	0.0402
8→9	0.3814	0.2689	0.2207	0.0987	0.0303
9→9	0.3441	0.2681	0.2304	0.1240	0.0335
9→10	0.3181	0.2987	0.1730	0.1678	0.0424
10→10	0.3899	0.3162	0.2230	0.0693	0.0016
10→11	0.3214	0.2768	0.1933	0.1642	0.0442

Table 5.22: Partial angular scattering probability tables of water for Problem II.

Group-to-Group	Probabilities				
8→8	0.4969	0.2811	0.1059	0.0866	0.0296
8→9	0.5007	0.1787	0.1370	0.0958	0.0879
9→9	0.6050	0.1850	0.1322	0.0610	0.0168
9→10	0.5753	0.1449	0.1106	0.1082	0.0611
10→10	0.5174	0.2564	0.1691	0.0546	0.0024
10→11	0.5077	0.1582	0.1210	0.1207	0.0924

5.8.1 Fluence Estimates

The inverse, Brown's and the weighted sampling methods are incorporated into the MORSE code for solving this problem. The notations for the computer codes corresponding to these sampling methods are the same as the notations used in Problem I, which are INVER, DISCS and WGHTS, respectively.

Table 5.23: The response of Detector 1 for Problem II.

Sample size	INVER Response(FSD)	DISCS Response(FSD)	WGHTS Response(FSD)
20,000	9.4123E-5(0.01798)	8.9986E-5(0.02010)	9.5923E-5(0.23082)
	±0.436E-5*	±0.466E-5	±5.704E-5
40,000	9.3477E-5(0.01304)	8.9303E-5(0.01315)	9.4677E-5(0.12472)
	±0.314E-5	±0.303E-5	±3.042E-5
60,000	9.4070E-5(0.01067)	8.9624E-5(0.01086)	1.0228E-4(0.15601)
	±0.259E-5	±0.251E-5	±4.110E-5
80,000	9.4417E-5(0.00921)	9.1041E-5(0.00993)	9.5690E-5(0.12650)
	±0.224E-5	±0.233E-5	±3.118E-5
100,000	9.3946E-5(0.00827)	9.0318E-5(0.00869)	9.1419E-5(0.10667)
	±0.200E-5	±0.202E-5	±2.512E-5

* statistical variability corresponds to a 99 % confidence interval.

In this problem, the next event estimator [17] is used to evaluate the response of point detectors. From each collision point, this estimator scores the probability of the next event being at the detector site, but the particle need not necessarily reach the detector site. The detector response here is all collided (no uncollided response), since source particles cannot directly reach the detector. This type of estimator is useful for this problem, since the probability of neutrons reaching the point detector is very low. Further, from each collision point, there is only one definite probability for the next collision event being at the detector site, since the detector is represented by a point.

The estimated values for the two detector responses obtained using the three sampling methods are shown in Tables 5.23 and 5.24. In these tables, the fractional

standard deviations and the 99 % confidence intervals of solutions are also reported together with the estimated detector response. Five sample sizes with an increasing value are utilized in order to analyze the convergence of the estimated response.

Table 5.23 shows that the estimated response of the first detector obtained using the three sampling methods converges, as indicated by the decrease in FSDs as the sample size increases, except for the FSD of WGHTS with 60,000 samples. With the increase in sample size from 40,000 to 60,000, the estimated response of WGHTS increases with the magnitude of 0.76E-5. This difference however is still less than the statistical variability, which is equal to 3.04E-5. The discrepancy therefore is still within the confidence interval, which means that the estimated response of WGHTS is still converging.

Table 5.24: The response of Detector 2 for Problem II.

Sample size	INVER	DISCS	WGHTS
	Response(FSD)	Response(FSD)	Response(FSD)
20,000	9.4501E-5(0.01819)	9.1574E-5(0.01947)	9.4539E-5(0.14149)
	±0.443E-5*	±0.459E-5	±3.446E-5
40,000	9.3784E-5(0.01249)	9.2950E-5(0.01369)	1.1238E-4(0.18857)
	±0.302E-5	±0.328E-5	±5.459E-5
60,000	9.4387E-5(0.01062)	9.2299E-5(0.01090)	9.9699E-5(0.14321)
	±0.258E-5	±0.259E-5	±3.678E-5
80,000	9.3646E-5(0.00916)	9.1331E-5(0.00937)	9.8497E-5(0.11364)
	±0.221E-5	±0.220E-5	±2.883E-5
100,000	9.3601E-5(0.00831)	9.1299E-5(0.00829)	9.7451E-5(0.09426)
	±0.200E-5	±0.195E-5	±2.366E-5

* statistical variability corresponds to a 99 % confidence interval.

The estimated response of INVER converges to the highest value, while that of DISCS converges to the lowest value. However, the confidence intervals obtained using the three sampling methods still overlap each other. This indicates that the solutions of DISCS and WGHTS are unbiased.

The estimated values of the second detector response are summarized in Table 5.24. For the three sampling methods, their estimates converge as indicated

by the decrease in FSDs as the sample size increases, except for the FSD of WGHTS with 40,000 samples. As the sample size increases to 40,000, the estimated response increases with the magnitude of 1.784E-5, which is less than the statistical variability (3.446E-5). Since the discrepancy is still within the confidence interval, the estimated response of WGHTS converges to the expected solution.

Since the locations of the two detectors are symmetrical, one would expect that the estimated values for both detector response are close to each other. Tables 5.23 and 5.24 show that for 100,000 samples the differences between solutions of Detectors 1 and 2 obtained using INVER, DISCS and WGHTS are 0.0345E-5, 0.0981E-5, 0.6032E-5, respectively. These differences however are still less than their statistical variability as indicated by the 99 % confidence intervals. Thus, the estimated values for both detector response converge to the expected solution.

In both detectors as shown in Tables 5.23 and 5.24, the convergence of estimated response obtained using INVER and DISCS is faster than that of WGHTS. This can be explained by the fact that the fractional standard deviations of INVER and DISCS to be close to each other, while WGHTS results in about 7 times larger value. The large FSD of the estimated response obtained using WGHTS can be explained by the fact that the adjustment factors utilized by the weighted sampling method vary significantly. Subsequently, the statistical weights of collided neutrons described by Equation (5.4) are affected. In the MORSE codes incorporating the inverse and Brown's sampling methods, the weights of neutrons are less or equal to one. On the other hand, the weights of neutrons can be larger than unity in WGHTS. Consequently, larger differences can occur more often since the magnitude interval of particle's weights is wider. Therefore, their values have a larger variability.

5.8.2 Processing Time and Speedups

Table 5.25 reports the scalar processing time of MORSE codes employing different sampling methods. The increase in the processing time required by the three sampling methods is linearly proportional with the increase in the sample size. That is, as the sample size increases by a factor of two, the processing time also increases by a factor about two.

Table 5.25: Scalar processing time of MORSE simulation for Problem II.

Sample size	Processing time in seconds		
	INVER	DISCS	WGHTS
20,000	19.454	19.986	13.793
40,000	39.397	39.595	28.021
60,000	59.013	60.066	42.692
80,000	78.973	80.899	55.019
100,000	97.457	100.285	69.817

As discussed in Problem I, the sampling methods affect the number of collisions in the MORSE simulations, which in turn affects the processing time. Similarly, the relationship between the processing time and the number of collisions is examined here. Table 5.25 shows that WGHTS requires the least processing time, followed by INVER and DISCS. The processing time reduction in WGHTS is affected by the reduction in the number of collisions. The DICSS code requires more processing time than INVER, even though it involves fewer collisions. This is due to the following reason. As discussed in Subsection 3.2.2, the processing time of the inverse method depends on the length of a probability table, while Brown's method requires a constant time regardless the length of a probability table. The inverse method performs well here, since it requires only few comparisons to construct a sample. The reason is that only the first few mass points in the probability table have high probabilities, while the rest mass points have low probabilities. Therefore, the COLISN routine incorporating Brown's method requires more processing time than that implementing the inverse

method.

Table 5.26: Number of collisions in MORSE simulation for Problem II.

Sample size	Number of collisions		
	INVER	DISCS	WGHTS
20,000	22,060	21,130	14,068
40,000	44,093	42,538	28,188
60,000	65,852	63,874	42,135
80,000	87,740	85,187	55,966
100,000	109,040	106,307	69,894

Table 5.26 reports the number of collisions occured during Monte Carlo simulations of Problem II. One can notice that there is a linear relationship between the number of collisions and the sample size. This is indicated by the fact that an increase in sample size by a factor of two produces an increase in the number of collisions by a factor about two.

The DISCS produces about 3 % fewer number of collisions than the INVER's collisions. However, WGHTS produces only about 64 % of the INVER's collisions. The reason is that the random walks in MORSE with weighted sampling can reach lower energy groups with higher probabilities than those employing the other sampling methods; since the original scattering probability tables (shown in Tables 5.19 and 5.20) have high probabilities only for the first few groups and low probabilities for the rest of the lower groups. This leads to earlier termination of random walks in WGHTS as low energy particles reach the energy cut-off. The weighted sampling method produces therefore fewer collisions.

The scalar speedups of different MORSE codes is shown in Table 5.27. These speedups are calculated according to the scalar processing time of DISCS and WGHTS relative to that of INVER. For each sampling method, the speedups for different sample sizes is constant. This is caused by the linear relationship between the sample size and the processing time required by each sampling method. One can notice that

Table 5.27: Speedups of different scalar MORSE codes relative to scalar INVER for Problem II.

Sample size	Speedups	
	DISCS	WGHTS
20,000	0.973	1.410
40,000	0.995	1.406
60,000	0.982	1.382
80,000	0.976	1.435
100,000	0.972	1.396

Brown's method is about 2 % slower than the inverse method, while the weighted sampling method is about 40 % faster. This fact further emphasizes the advantage of using the weighted sampling method to speed up Monte Carlo computations.

5.8.3 Local Speedups

In this subsection, the processing time performance of different sampling methods incorporated into subroutine COLISN is examined. The scalar processing time of Brown's and the weighted sampling methods is compared to that of the inverse method, with the objective to obtain the local speedups. These speedups provide a direct measure of the speeding achieved by the sampling methods, without the additional overhead of other routines in the MORSE code.

Table 5.28: Scalar processing time of different sampling methods in COLISN routine for Problem II.

Sample size	Processing time in milliseconds		
	INVER	DISCS	WGHTS
20,000	1718.969	1715.083	1612.504
40,000	3436.748	3423.066	3224.210
60,000	5158.662	5136.702	4842.191
80,000	6877.693	6846.168	6447.755
100,000	8598.855	8558.972	8068.646

Table 5.28 reports the scalar processing time of the three sampling methods. It can be seen that the processing time for each sampling method is linearly proportional to the sample size. For all sample sizes, the least processing time is required by WGHTS, followed by DISCS and INVER. One can notice that this order is different from that obtained in Problem I, where DISCS required larger processing time than that of INVER. This can be explained by the fact that the probability tables used in this problem are longer than those of Problem I. Consequently, the inverse sampling method requires more comparisons to locate an outgoing energy group of a collided particle, and therefore it requires larger processing time.

On average, the processing time of scalar DISCS and WGHTS is about 99.7 % and 93.8 % of that of scalar INVER, respectively. The processing time of WGHTS here is slightly less than that of INVER, while Table 5.27 shows that the global speedup of WGHTS relative to INVER is three. Thus, one can conclude that the global speedup achieved by the MORSE code incorporating the weighted sampling method is mostly affected by the reduction in the number of collisions.

5.8.4 Local Vectorization Speedups

This subsection examines the local vectorization speedups of different sampling methods in subroutine COLISN. As discussed in Subsection 5.7.4, this analysis provides indications of the potential vectorization speedups if the entire MORSE code is vectorized.

The processing time of vector INVER, DISCS and WGHTS is given in Table 5.29. The processing time for all sampling methods is linearly proportional to the sample size. For all sample sizes, the least processing time is for WGHTS, followed by DISCS and INVER. This order remains the same as that in the scalar processing. However, the processing time of vector DISCS and WGHTS is only 37 % and 32 %, respectively. This indicates that the vectorizability of Brown's and the weighted sampling methods is higher than that of the inverse method.

Table 5.29: Vector processing time of different sampling methods in COLISN routine for Problem II.

Sample size	Processing time in milliseconds		
	INVER	DISCS	WGHTS
20,000	522.803	192.452	165.880
40,000	1045.616	384.045	332.751
60,000	1569.946	577.747	501.057
80,000	2093.273	768.228	663.265
100,000	2615.586	960.814	833.938

Table 5.30 summarizes the speedups of different vector codes relative to scalar INVER. As the sample size increases these speedups are about constant, due to the linear relationship between the sample size and the scalar as well as vector processing time. In this problem, the speedups of DISCS and WGHTS are slightly higher than those of MORSE Problem I, since the scalar INVER requires larger processing time due to the longer probability tables utilized.

Table 5.30: Speedups of the vectorized sampling methods relative to the scalar inverse code for Problem II.

Sample size	Speedups		
	INVER	DISCS	WGHTS
20,000	3.288	8.932	9.721
40,000	3.287	8.949	9.690
60,000	3.286	8.929	9.664
80,000	3.286	8.953	9.721
100,000	3.288	8.950	9.675

5.8.5 Efficiency

Tables 5.31 and 5.32 show the efficiencies of DISCS and WGHTS relative to INVER for Detectors 1 and 2, respectively. For both detectors, the efficiencies of DISCS are close to those of INVER, while the efficiencies of WGHTS are much smaller. Table 5.31 shows that the efficiency of the weighted sampling relative to that of the inverse method is about 0.9 %. For detector 2, shown in Table 5.32, the efficiency of WGHTS relative to that of INVER is about 1.0 %. The weighted sampling code results in low efficiencies, since its scalar speedups cannot compensate for the increase in the sample variance of its solution.

Table 5.31: Efficiencies of different MORSE codes relative to those of INVER, for Detector 1 of Problem II.

Sample size	Relative Efficiency	
	DISCS	WGHTS
20,000	0.852	0.008
40,000	1.072	0.015
60,000	1.045	0.005
80,000	0.903	0.007
100,000	0.952	0.009

Table 5.32: Efficiencies of different MORSE codes relative to those of INVER, for Detector 2 of Problem II.

Sample size	Relative Efficiency	
	DISCS	WGHTS
20,000	0.905	0.023
40,000	0.843	0.004
60,000	0.975	0.007
80,000	0.981	0.008
100,000	1.026	0.010

Table 5.33: Efficiencies of different MORSE codes relative to those of INVER, for about 5 % FSD of Problem II.

Detector	Relative Efficiency	
	DISCS	WGHTS
Detector 1	0.845	*
Detector 2	0.901	*

* the relative efficiencies are not available.

Table 5.33 reports the efficiencies of solutions obtained by DISCS relative to those of INVER, for about 5 % FSD. The relative efficiencies of WGHTS are not available, since the FSDs of WGHTS have not reached 5 % until 100,000 samples. This indicates that the relative efficiencies of WGHTS are very low.

5.8.6 Remarks

The performance of MORSE codes incorporating different sampling methods for estimating the responses of two point detectors has been examined in this section. It was found that the estimated detector responses obtained using the weighted sampling are unbiased, but have larger fractional standard deviations than those of the inverse and Brown's methods. The MORSE code incorporating the weighted sampling resulted in very low relative efficiencies, since its scalar speedups did not compensate for the increase in the sample variances of the solutions.

On average, the weighted sampling in this problem entailed higher scalar speedups and lower relative efficiencies than those of Problem I. The former is caused by the fact that scattering occured mostly in water, while in Problem I the scattering occured in air. The reason for the latter is that the next event estimator used in this problem is more sensitive to large magnitudes of statistical weights than the boundary crossing estimator used in Problem I.

5.9 Problem III

The physical characteristics of this problem are the same as those of Problem II, except for the water density. The water density is increased here from 0.25 g/cm^3 to 1.00 g/cm^3. The objective here is to examine the effect of the density of a transport medium on the performance of MORSE codes incorporating different sampling methods. Increasing the density of a transport medium increases the number of collisions in a simulation. This may affect, in particular, the performance of the weighted sampling method, which was shown in the previous two problems to produce results, that are affected by the number of collisions.

5.9.1 Fluence Estimates

Tables 5.34 and 5.35 show the estimated response for Detectors 1 and 2, respectively, together with the fractional standard deviations and the 99 % confidence intervals. In Table 5.34, as the sourse particles (sample size) increased, the estimated values of detector response obtained using the three sampling methods converge to the expected solution. This is indicated by the fact that the fractional standard deviation of each sampling method decreases as the sample size increases, except for DISCS with 40,000 samples. As the sample size increased to 40,000, the estimated value increases 0.289E-4. This increment however is less than the statistical variability (0.396E-4), hence the estimated response of DISCS also converges to the solution. The estimated values of detector responses obtained by INVER and DISCS converge to almost the same value, while that of WGHTS converges to a smaller value. However, their confidence intervals still overlap each other. This indicates that the estimated responses of DISCS and WGHTS are unbiased.

Table 5.35 shows the estimated response for Detector 2 obtained using three different sampling methods converges, as indicated by the decrease in their FSDs as the

sample size increased, except for INVER with 100,000 samples. The increment of INVER's estimate is 0.104E-4, when the sample size increased from 80,000 to 100,000. This increment is less than the statistical variability, which is equal to 0.284E-4. Thus, the INVER's estimate also converges to the expected solution.

The estimated response of INVER converges to the largest value, while that of WGHTS converges to the lowest value. The confidence intervals of WGHTS, however, overlap those of INVER, except for a sample size equals 100,000. For a sample size of 100,000, the difference between the intervals of INVER and WGHTS is 0.03E-4. As the case of Problem I, WGHTS results in lower values of estimated detector response due to the utilization of uniform distributions for determining the outgoing energy groups and scattering angles. The random walks in WGHTS therefore tend to reach lower energy groups faster than in the other two sampling methods. This leads to earlier termination of the random walks. The accumulated fluence is therefore less than it should be. Thus, the estimated response has a lower value.

Table 5.34: The response of Detector 1 for Problem III.

Sample size	INVER	DISCS	WGHTS
	Response(FSD)	Response(FSD)	Response(FSD)
20,000	2.9084E-04(0.03503)	2.8084E-04(0.02370)	3.1197E-04(0.23935)
	±0.262E-4*	±0.172E-4	±1.923E-4
40,000	2.9783E-04(0.02320)	3.0970E-04(0.04964)	2.6646E-04(0.14501)
	±0.178E-4	±0.396E-4	±0.995E-4
60,000	2.9630E-04(0.01865)	3.0649E-04(0.03483)	2.5932E-04(0.10446)
	±0.142E-4	±0.275E-4	±0.698E-4
80,000	2.9853E-04(0.01594)	3.0534E-04(0.02752)	2.6476E-04(0.08875)
	±0.123E-4	±0.217E-4	±0.605E-4
100,000	2.9730E-04(0.01390)	2.9957E-04(0.02284)	2.5323E-04(0.07496)
	±0.107E-4	±0.176E-4	±0.489E-4

* statistical variability corresponds to a 99 % confidence interval.

Table 5.35: The response of Detector 2 for Problem III.

Sample size	INVER	DISCS	WGHTS
	Response(FSD)	Response(FSD)	Response(FSD)
20,000	2.9208E-04(0.02657)	2.8447E-04(0.02820)	3.0364E-04(0.15015)
	±0.200E-4*	±0.207E-4	±1.174E-4
40,000	2.9342E-04(0.02086)	2.8881E-04(0.01864)	2.4555E-04(0.09678)
	±0.158E-4	±0.139E-4	±0.612E-4
60,000	2.9011E-04(0.01607)	2.9009E-04(0.01598)	2.4281E-04(0.07404)
	±0.120E-4	±0.119E-4	±0.463E-4
80,000	2.9293E-04(0.01385)	2.8887E-04(0.01338)	2.4427E-04(0.06348)
	±0.105E-4	±0.100E-4	±0.399E-4
100,000	3.0329E-04(0.03637)	2.8855E-04(0.01202)	2.3885E-04(0.05381)
	±0.284E-4	±0.089E-4	±0.331E-4

* statistical variability corresponds to a 99 % confidence interval.

5.9.2 Processing Time and Speedups

The processing time required by the scalar MORSE codes incorporating different sampling methods is shown in Table 5.36. One can notice that for each sampling method the scalar processing time is proportional to the sample size. That is, the processing time is about doubled as the sample size increases by a factor of two.

Table 5.36: Scalar processing time of MORSE simulation for Problem III.

Sample size	Processing time in seconds		
	INVER	DISCS	WGHTS
20,000	97.623	101.046	32.531
40,000	198.854	202.056	64.354
60,000	295.307	303.351	97.408
80,000	394.268	406.188	126.900
100,000	495.063	510.348	160.957

The DISCS code requires more processing time than INVER, even though it simulates fewer collisions. The reason is that Brown's sampling method in the COLISN routine requires more processing time than the inverse method. The WGHTS code requires much less processing time than INVER, since it simulates much fewer collisions. Thus, similar to the processing time of scalar MORSE codes in Problem II, the processing time is affected by the number of collisions in the simulation.

Table 5.37: Number of collisions in MORSE simulation for Problem III.

Sample size	Number of collisions		
	INVER	DISCS	WGHTS
20,000	188,438	173,091	47,121
40,000	377,499	347,880	94,478
60,000	570,484	521,653	141,905
80,000	759,479	694,390	189,367
100,000	949,270	866,103	236,425

Table 5.37 reports the number of collisions occured in the simulations of MORSE

codes incorporating different sampling methods. It is shown that WGHTS produces the least number of collisions, followed by DISCS and INVER. One can notice that this order remains the same as in Problem II. This table shows that the number of collisions produced by WGHTS is only about 25 % of the INVER's collisions, while that of DISCS is about 91 %. The percentage of WGHTS here is much smaller than that in Problem II. This can be explained by the fact that the number of collisions of INVER and DISCS increase by factors of 8.6 and 8.2, respectively, while that of WGHTS increases by a factor of 3.4 only, as the water density increased by a factor of four.

Table 5.38: Speedups of different scalar MORSE codes relative to scalar INVER for Problem III.

Sample size	Speedups	
	DISCS	WGHTS
20,000	0.966	3.001
40,000	0.984	3.090
60,000	0.973	3.032
80,000	0.971	3.107
100,000	0.970	3.076

The scalar speedups of different MORSE codes obtained relative to INVER are summarized in Table 5.38. For different sample sizes, the speedup is about constant due to the linear relationship between the processing time and the sample size. By comparing these speedups and those in Problem II, one can notice that the speedups of DISCS in these two problems are about the same, while the speedup of WGHTS in this problem is 2.1 times larger than that of Problem II. This can be explained by the fact that, as the water density increased, the increase in the number of collisions of WGHTS is less than that of INVER, as discussed in the preceding paragraph.

In this section, the local scalar and vectorization speedups for Problem III are not examined, since it has the same results as those of Problem II. This is due to the fact that the scattering probability tables used in this problem are the same as those in

Problem II.

5.9.3 Efficiency

The efficiencies of DISCS and WGHTS relative to those of INVER for Detectors 1 and 2 are reported in Tables 5.39 and 5.40, respectively. For Detector 1, the relative efficiencies of DISCS and WGHTS for different sample sizes are less than one, except for DISCS with 20,000 samples. In general, the sample variances of DISCS and WGHTS are larger than that of INVER. The discrepancy occurs for 20,000 samples, where the sample variance of INVER is larger than that of DISCS.

The efficiencies of DISCS for Detector 2 are close to those of INVER, except for a sample size of 100,000. Except for 100,000 samples, the relative efficiencies of DISCS are about one, while those of WGHTS are less than one. As the sample size increases from 80,000 to 100,000, the sample variance of INVER significantly increases, while those of DISCS and WGHTS are about the same. In the calculation of a relative efficiency, the sample variance of INVER is as the numerator. For 100,000 samples therefore, the relative efficiencies of DISCS and WGHTS are higher than those of 80,000 samples.

Table 5.39: Efficiencies of different MORSE codes relative to those of INVER, for Detector 1 of Problem III.

Sample size	Relative Efficiency	
	DISCS	WGHTS
20,000	2.264	0.056
40,000	0.199	0.099
60,000	0.261	0.126
80,000	0.311	0.127
100,000	0.354	0.146

The efficiencies for about 5 % FSD are shown in Table 5.41. For both detectors, the relative efficiencies of DISCS are larger than one, while those of WGHTS are less

Table 5.40: Efficiencies of different MORSE codes relative to those of INVER, for Detector 2 of Problem III.

Sample size	Relative Efficiency	
	DISCS	WGHTS
20,000	0.904	0.087
40,000	1.272	0.205
60,000	0.985	0.204
80,000	1.069	0.213
100,000	9.812	2.266

Table 5.41: Efficiencies of different MORSE codes relative to those of INVER, for about 5 % FSD of Problem III.

Detector	Relative Efficiency	
	DISCS	WGHTS
Detector 1	1.479	0.016
Detector 2	1.970	0.032

than one. The sample variances of DISCS are smaller than those of INVER, while the sample variances of WGHTS are larger. The scalar speedup achieved by WGHTS does not compensate, once more, for the increase in the sample variance.

5.9.4 Remarks

This section examined a similar problem to Problem II. In this problem, the transport medium had a higher density. The effects of this higher density on the performance of MORSE codes incorporating different sampling methods were examined. It was found that the estimated response obtained using the weighted sampling converges to a lower value than that of the inverse method, due to early termination of the random walks in the simulations. The estimated values however are still unbiased since the 99 % confidence intervals overlap those of the inverse method.

The MORSE code incorporating the weighted sampling is three times faster than

that incorporating the inverse method. This further emphasizes the attractiveness of the weighted sampling method for vector processing. However, the efficiencies are smaller than those of the other methods, since its sample variances are larger.

5.10 Conclusions

Three sampling methods were incorporated into MORSE codes; namely, the inverse method, Brown's method and the weighted sampling method. The inverse method required extra time and storage to prepare cumulative probability tables. Brown's method required a complicated procedure and a large amount of memory to set up the probability tables in MORSE. The weighted sampling however did not require preprocessing of scattering and angular scattering probability tables, since it utilizes the original probability tables without any modifications.

It was shown, numerically, that MORSE code incorporating the weighted sampling method entails unbiased estimates. For shells with long radii in Problem I, the estimated fluence obtained using the weighted sampling converged to smaller values, due to earlier termination of the random walks in the simulations. The estimated detector response in Problem III also converged to a smaller value for the same reason.

The incorporation of the weighted sampling method was found to speed up the scalar computation of MORSE code by at most a factor of three. In its present state, the MORSE codes did not take full advantages of the weighted sampling method since only scalar processing was carried out. In order to show that the weighted sampling can enhance the vectorizability of the COLISN routine, three sampling methods incorporated in this routine were vectorized. The speedup of the vectorized weighted sampling code relative to the scalar inverse code was about 9.7. This means that one can expect such high speedups when event-based vectorization is implemented for the entire MORSE code.

Efficiencies were calculated based on the scalar processing time of the MORSE

codes. In the applications examined, the weighted sampling method resulted in lower efficiencies than Brown's or the inverse methods, since its scalar speedups could not compensate for the increase in the fractional standard deviations of its solutions. One can predict however, that the weighted sampling method will entail higher efficiencies when the entire MORSE code is vectorized.

In this and previous chapters, the weighted sampling method utilized a uniform distribution to speed up the construction of samples. This method often entailed scalar and vector speedups. However, in this chapter, the MORSE codes incorporating the weighted sampling method resulted in very low efficiencies due to the large sample variances of its estimates. In the following chapter, efforts are directed towards developing variants of the weighted sampling method which can increase the efficiency by reducing sample variance, while maintaining the vectorizability of the method.

Chapter 6

Variants of The Weighted Sampling Method

6.1 Introduction

In the preceding chapters, the performance of the weighted sampling method was examined for a variety of problems. The weighted sampling method utilized discrete uniform distributions to construct samples from probability tables, since discrete uniform random numbers could be generated relatively fast in scalar as well as vector processing. This method was found to be successful to speed up the construction of samples, and to enhance the vectorizability of vector Monte Carlo codes. However, this approach resulted in large sample variances in applications involving one and two-dimensional probability tables. Therefore, for these cases, the efficiencies of the weighted sampling method were very low. This chapter proposes four variants of the weighted sampling method, which can be used to increase efficiency of the weighted sampling method by reducing sample variances and by reducing the processing time for constructing samples. An effort is also made towards enhancing the vectorizability of these variants of the weighted sampling method.

The proposed four sampling methods are designated by *the weighted sampling*

with a stretched table (WSST), *the weighted sampling with a nonuniform distribution* (WSNU), *the combination of the inverse and weighted sampling methods* (INWS), and *the combination of the inverse method and the WSST method* (INWSST). The basic weighted sampling method used in the previous chapters is referred to here as *the direct weighted sampling method* (WGHTS).

This chapter is organized as follows. Section 6.2 discusses WSST. The WSNU is described in Section 6.3. This section also presents examples of WSNU employing two different nonuniform distributions. In Section 6.5, the vectorization of the weighted sampling variants is investigated. Section 6.6 reconsiders the problem involving a one-dimensional table discussed in Chapters 3, and demonstrates that WSST and WSNU can improve the performance of the direct weighted sampling. Sections 6.7 to 6.9 reexamine the three neutron transport problems, which were discussed in Chapter 5. Finally, some conclusions of this chapter are given in the last section.

6.2 Weighted Sampling With a Stretched Table (WSST)

The WSST utilizes the direct weighted sampling method to sample from a probability table which is stretched, so that the adjustment factors exhibit low variability. The sample variance of an estimated quantity would be then expected to be reduced.

The generation table is constructed by dividing bins with associated large probabilities such that the probabilities are close to uniform, while keeping the memory requirements as low as possible.

Table 6.1 is the same as Table 2.1. It is shown again to clarify the discussion. Based on Table 6.1, the WSST generation table shown in Table 6.2 is constructed by dividing the first four bins. Each of these bins is expanded into two bins, and the corresponding probability is divided evenly. The length of this generation table (n) is equal to 9. Array `w` stores the new mass points, which are equal to $x_j p_j n$, for

$j = 1, 2, \cdots, n.$

Table 6.1: An example of a probability table.

Mass points (x)	10	20	30	40	50
Probabilities (p)	0.40	0.20	0.30	0.08	0.02

Table 6.2: The generation table of the WSST.

x	10	10	20	20	30	30	40	40	50
p	0.20	0.20	0.10	0.10	0.15	0.15	0.04	0.04	0.02
w	18.0	18.0	18.0	18.0	40.5	40.5	14.4	14.4	9.0

For this probability table, the direct weighted sampling method results in a sample variance of 171.76, while the inverse method results in a sample variance of 118.56. However, when Table 6.2 is used, the WSST reduces the sample variance to 114.14. Thus, the WSST method can reduce the sample variance to a value closer to that of the inverse method, while providing a high speedup which will be shown in Section 6.6.

6.3 Weighted Sampling With A Nonuniform Distribution (WSNU)

The WSNU is essentially a direct weighted sampling method, but it utilizes a discrete nonuniform distribution. This distribution is to be constructed such that the sample variance is reduced and the sampling process is speeded up.

In order to reduce the sample variance, the probability mass function of the nonuniform distribution should have a similar shape to that of the $x_j p_j$ quantity, for $j = 1, 2, \cdots, n$, where x_j and p_j denote a mass point and the associated probability in the probability table. The closer the shape the smaller sample variance will be.

In WSNU, the time for constructing samples is minimized by employing a discrete nonuniform distribution, which has a closed form mathematical expression. Hence, the inverse function of the probability distribution function can be used to generate discrete random numbers in the interval $[1, n]$, which are subsequently used to select mass points and the associated probabilities from the probability table. With this procedure, WSNU does not involve any comparisons. Thus, some extra time is required to generate other discrete random numbers for selecting mass points and the associated probabilities from the probability table. This extra time however will be less than the time required by the inverse method for comparisons if the probability table is relatively long.

In the implementation, the WSNU uses directly the original probability tables without any alterations. Let us consider the construction of samples from Table 2.1 for illustration of this method. For this example, WSNU employs a binomial distribution with the probability mass function given as

$$p_j = \begin{cases} \binom{n}{j} a^j (1-a)^{n-j} & \text{if } j \in \{0, 1, \cdots, n\}, \\ 0 & \text{otherwise,} \end{cases} \tag{6.1}$$

where

$$\binom{n}{j} = \frac{n!}{j!(n-j)!} \tag{6.2}$$

while a is the shape parameter, and n is the range parameter. It is found that by employing a binomial distribution with n equals 5 and a equals 0.33, WSNU results in sample variance of about 51.38. The scalar code of WSNU employing the binomial distribution is described in Figure 6.1.

Subroutine `RNBIN` and function `BINPR` are taken from IMSL [46]. Subroutine `RNBIN(1,n,0.33,j)` generates a binomial random number in the interval $[1, n]$ with shape parameter 0.33, and stores it into variable `j`. Function `BINPR(j,n,0.33)` gives the probability of the selected binomial random number. This probability is used as

```
      do 10 i = 1,k
         call RNBIN(1,n,0.33,j)
         rv(i)=p(j)*x(j)/BINPR(j,n,0.33)
10    continue
```

Figure 6.1: Scalar code of WSNU employing a binomial distribution.

an adjustment factor, as shown in Figure 6.1.

```
C- construct the geometric probabilities
      do 10 j = 0,n-1
         pgeo(j)=a*(1.0-a)**j
 10   continue
C- generate samples from the probability table
      b=alog(1.0-a)
      do 20 i = 1,k
         j=int(alog(RNUNF())/b)
         rv(i)=x(j)*p(j)/pgeo(j)
 20   continue
```

Figure 6.2: Scalar code of WSNU employing a geometric distribution.

In Section 6.6, the WSNU method employs a geometric distribution. Its probability mass function is given as

$$p_j = \begin{cases} a(1-a)^j & \text{if } j \in \{0, 1, \cdots\}, \\ 0 & \text{otherwise,} \end{cases} \tag{6.3}$$

where a is the shape parameter. The scalar code of WSNU employing the geometric distribution is shown in Figure 6.2.

In Figure 6.2, geometrically distributed random numbers are generated using an algorithm described by Law [56, p. 266]. Note that the first indices of arrays p, x and pgeo are 0. The first loop is used to calculate the geometric probabilities. The second loop generates samples from a probability table represented by arrays x and p.

Each sample is subsequently divided by the adjustment factor, which is the geometric probability of the selected mass point.

6.4 Weighted Sampling With The Inverse Methods

In the combination of the weighted sampling and the inverse methods (INWS), the inverse method is used for the first few mass points, and the weighted sampling method is utilized for the rest of mass points.

```
       do 10 i = 1,k
         r1 = RNUNF()
C- construct samples using the inverse method
         do 20 j = 1,m
           if(c(j) .ge. r1) then
             rv(i) = x(j)
             go to 10
           endif
 20      continue
C- construct samples using the weighted sampling
         r1  = (r1-c(m))/(1.0-c(m))
         l   = m + int(r1 * (n-m)) + 1
         rv(i)=x(l)*p(l)*(n-m)/(1.0-c(m))
 10    continue
```

Figure 6.3: Scalar code of INWS method.

The partial utilization of the inverse method can reduce the sample variance by reducing the variability of the adjustment factors, while the weighted sampling method speeds up the construction of samples by reducing the time spent for comparisons in the inverse method. Thus, this time reduction will be significant if the probability table is long.

This method is useful when the probabilities for the first few mass points are

very different with those of the remaining mass points, and the rest probabilities are relatively uniform. Such a probability table is shown in Table 5.3. In this table, the probabilities for the first few mass points are much higher than those for the remaining mass points, which have probabilities close to those of a uniform distribution.

Figure 6.3 shows the scalar code of INWS method. The inverse method is used for the first m mass points, while the direct weighted sampling is used for the subsequent $(n-m)$ mass points. For this purpose, the associated probabilities of the $(n-m)$ mass points have to be adjusted so that their total value is equal to one. Two random numbers are used; the first one is generated using IMSL function `RNUNF`, while the second random number utilizes the former by adjusting the interval of the remainder. Therefore, the interval of the second random number is $(0,1)$. Note that the second random number can also be generated by utilizing the lower digits of the first random number (see Section 2.6).

The INWSST is the same as INWS, except that the weighted sampling method uses a stretched probability table. This stretching is required if the remaining probabilities vary significantly, as exemplified by the scattering probability tables of water shown in Table 5.20.

6.5 Vectorization

In this section, the vectorization of WSNU and INWS codes is investigated. WSST is not discussed here since the code is the same as the code of the direct weighted sampling implementing technique II (see Section 3.2). The vectorization of INWSST is not described separately, since the vector code is the same as that of INWS.

```
C- construct the binomial probabilities
      do 10 j = 0,n-1
         pbin(j)=BINPR(j,(n-1),a)
 10   continue
C- generate samples from the probability table
      call RNBIN(k,(n-1),a,m)
      do 20 i = 1,k
         rv(i)=x(m(i))*p(m(i))/pbin(m(i))
 20   continue
```

(a) The vector code.

```
            C- construct the binomial probabilities
UNAN                do 10 j=0,n-1
                      pbin(j)=BINPR(j,(n-1),a)
             10     continue
            C- generate samples from the probability table
                   call RNBIN(k,(n-1),a,m)
VECT +--------     do 20 i=1,k
     |_______          rv(i)=x(m(i))*p(m(i))/pbin(m(i))
```

(b) The vector compiler report.

Figure 6.4: Vector code of WSNU employing a binomial distribution.

The vectorizability of WSNU depends on the nonuniform distribution employed. When WSNU uses a binomial distribution the code is not entirely vectorizable, as

shown in Figure 6.4(b). In this figure, `UNAN` denotes that the corresponding loop is not vectorizable since subroutine `BINPR` is only for scalar processing. Subroutine `RNBIN(k,(n-1),a,m)` generates k binomial random numbers and stores them into array `m`. This subroutine is only for scalar processing mode.

In contrast with the binomial distribution, Figure 6.5(b) demonstrates that the code is entirely vectorized when WSNU employs a geometric distribution. Note that `SURAND(seed1,k,rn)` uses `seed1` to generate `k` uniform random numbers and stores them into array `rn`. In this figure, `ELIG` indicates that the corresponding loop is vectorizable, but it is not vectorized since scalar processing is faster than vector processing. This is due to the fact that the table length (`n`) is relatively short (`n` = 10).

IMSL provides a subroutine, called `RNGEO`, to generate geometric random numbers. We do not use this subroutine since it is not vectorized. Note that ESSL [43] only provides vectorized routines for generating random numbers according to uniform and normal distributions.

In the scalar code of the INWS method, the first part is for the inverse method and the second part of the code is for the weighted sampling method. If the vectorization is carried out in the same order, the vector code will require an `if`-statement for the direct weighted sampling method. Consequently, the vector code will not be vectorized, since it involves indirect addressing. Therefore, to increase the vectorizability of the INWS code, the order of the computation is reversed. That is, the direct weighted sampling method is first carried out to obtain k samples, and then the inverse method is used to replace some of the samples corresponding to the first m mass points.

Figures 6.6(a) and (b) show the vector code and vector compiler report of INWS method. This code utilizes two sets of uniform random numbers. The first set is generated using subroutine `SURAND`, while the second set of random numbers utilize the former by adjusting the range of the remainders. As shown in Figure 6.6(a),

two iterations are required. In the first iteration, the direct weighted sampling is carried out to construct k temporary samples, regardless of whether this method is really needed or not. In the second iteration, the inverse method is implemented to correct some of the temporary samples resulting from the first iteration. Therefore, some unnecessary processing time is spent in the first iteration, when the weighted sampling method is really not required.

One can predict that the inverse method is required more often if the first m mass points have much higher probabilities than the other remaining probabilities, since a uniform random number is most probably located in these mass points. Consequently, the extra processing time becomes larger since more unnecessary sampling is carried out by the weighted sampling method in the first iteration.

The advantage of this vector code is that the first loop, implementing the direct weighted sampling, is fully vectorizable. The innerloop of the code for the inverse method is also vectorizable. The vector compiler reports that this innerloop is eligible for vectorization; however, it is not vectorized since the scalar processing is faster than the vector processing, due to the fact that the value of m is only 3 in this run.

6.6 Estimation of a Distribution Mean

In this section, the performance of WSST and WSNU implemented on the IBM 3090-180 VF is investigated. These methods are used to estimate the mean of an arbitrary distribution represented by the probability table given in Table 2.6. This problem is reexamined since Chapter 3 showed that the direct weighted sampling (WGHTS) produced a large fractional standard deviation (FSD).

The inverse and Brown's methods are chosen for comparison with the WSST and WSNU methods. The inverse method is often used and performs well for this problem, while Brown's method performs well in scalar and vector processing and attains the highest efficiency in vector processing.

```
C- construct the geometric probabilities
      do 10 j = 0,n-1
         pgeo(j)=a*(1.0-a)**j
 10   continue
C- generate samples from the probability table
      b=alog(1.0-a)
      call SURAND(seed1,k,rn)
      do 20 i = 1,k
         j=int(alog(rn(i))/b)
         rv(i)=x(j)*p(j)/pgeo(j)
 20   continue
```

(a) The vector code.

```
           C- construct the geometric probabilities
ELIG +-------      do 10 j=0,n-1
     |_______         pgeo(j)=a*(1.0-a)**j
           C- generate samples from the probability table
                 b=alog(1.0-a)
                 call SURAND(seed1,k,rn)
VECT +-------     do 20 i=1,k
     |                j=int(alog(rn(i))/b)
     |_______         rv(i)=x(j)*p(j)/pgeo(j)
```

(b) The vector compiler report.

Figure 6.5: Vector code of WSNU employing a geometric distribution.

```
        call SURAND(seed1,k,rn)
        a = (n-m)/(1.0-c(m))
C-  construct samples using the weighted sampling
        do 10 i = 1,k
          r1  = (rn(i)-c(m))/(1.0-c(m))
          l   = m + int(r1 * (n-m)) + 1
          rv(i)=x(l) * p(l) * a
 10     continue

C-  construct samples using the inverse method
        do 20 i = 1,k
          do 30 j = 1,m
            if(c(j) .ge. rn(i)) then
              rv(i) = x(j)
              go to 20
            endif
 30       continue
 20     continue
```

(a) The vector code.

```
                call SURAND(seed1,k,rn)
                a = (n-m)/(1.0-c(m))
            C-  construct samples using the weighted sampling
VECT +--------      do 10 i = 1,k
     |                r1  = (rn(i)-c(m))/(1.0-c(m))
     |                l   = m + int(r1 * (n-m)) + 1
     |_______         rv(i)=x(l) * p(l) * a

            C-  construct samples using the inverse method
UNAN                 do 20 i = 1,k
ELIG +--------         do 30 j = 1,m
     |_______             if(c(j) .ge. rn(i)) then
              20     continue
```

(b) The vector compiler report.

Figure 6.6: Vector code of INWS method.

Notations for the scalar codes are as follows:

1. INVER: code of the inverse method,
2. DISCS: code of Brown's method,
3. WGHTS1: code of the direct weighted sampling implementing technique I,
4. WGHTS2: code of the direct weighted sampling implementing technique II,
5. WSST: code of the weighted sampling using a stretched table, and
6. WSNU: code of the weighted sampling employing a geometric distribution.

The notations for the corresponding vector codes are the same as those for the scalar codes, except for INVER; the vector code of the inverse method is denoted by INVR2, and is given in Figure 2.3. The INVR2 code is chosen instead of INVR1, since Section 2.8 reported that INVR2 requires less processing time for a short probability table as given in Table 2.6, on page 22.

The WSST code is the same as shown in Figure 3.3, while the generation table used is given in Table 6.3. The term $(m\times)$ denotes that the corresponding mass point is repeated m times and accordingly the associated probability is equally distributed (similar to as discussed in Section 2.4).

Table 6.3: Generation table for WSST.

x	100(12×)	90(4×)	70(2×)	50	20	15	10	5	2	1
p	.05(12×)	.05(4×)	.05(2×)	.030	.025	.016	.013	.010	.005	.001

6.6.1 Estimates of the Distribution Mean

Table 6.4 reports the estimates of the distribution mean and their %FSDs obtained using different scalar codes. The results obtained by the corresponding vector codes

are not shown since they are the same as those of the scalar codes. Since the two codes of the weighted sampling methods, WGHTS1 and WGHTS2, have the same results, only one result is shown in the WGHTS column. The WSNU obtains the estimated values by employing a geometric distribution with the shape parameter (a) equals 0.68, since it was found that this parameter results in the smallest FSD (the sample variance is only about 39).

Table 6.4: Mean estimates and their %FSDs.

Sample size	INVER	DISCS	WGHTS	WSST	WSNU
	$\overline{X}$(%FSD)	$\overline{X}$(%FSD)	$\overline{X}_w$(%FSD)	$\overline{X}_w$(%FSD)	$\overline{X}_w$(%FSD)
20,000	87.44(0.19)	87.40(0.19)	87.76(1.44)	87.56(0.41)	87.49(0.05)
40,000	87.42(0.14)	87.58(0.13)	87.36(1.02)	87.48(0.29)	87.45(0.04)
60,000	87.41(0.11)	87.56(0.11)	87.54(0.84)	87.45(0.24)	87.43(0.03)
80,000	87.39(0.10)	87.57(0.09)	87.47(0.73)	87.38(0.20)	87.43(0.03)
100,000	87.43(0.09)	87.50(0.08)	87.49(0.65)	87.48(0.18)	87.43(0.02)

In Table 6.4, five sample sizes from 20,000 to 100,000 are used for the estimation in order to obtain small FSDs and examine the convergence. One can notice that all of the mean estimates converge to the distribution mean, which is equal to 87.431, as indicated by the fact that their % FSDs decrease as the sample size increases. The estimated values therefore converge to the unbiased solution.

One can notice that the FSDs produced by WSST are lower than those of WGHTS. These FSDs however are still higher than those of INVER and DISCS. In contrast with WGHTS, WSNU gives the smallest FSD. Thus, the WSNU method reduces significantly the sample variance of the direct weighted sampling method.

6.6.2 Processing Time

The processing time of different scalar codes is shown in Table 6.5. It shows that the increase in processing time of each sampling method is linearly proportional to the increase in sample size. In this table, one can notice that the least processing time

is for WGHTS2 as well as WSST, followed by WGHTS1; then the processing time increases respectively for the DISCS, INVER, and WSNU codes. The difference in the processing time of WGHTS2 and WSST is of the order of microseconds. This can be explained by the fact that WGHTS2 and WSST have the same codes, and the lengths of their generation tables are not much different. These two codes require the least processing time, since they only carry out fetching from memory during the construction of samples. On the other hand, the WSNU code requires the largest processing time due to the extra time spent for generating discrete random numbers from the geometric distribution.

Table 6.5: Scalar processing time of various codes to sample from Table 2.6, in milliseconds.

Samples	INVER	DISCS	WGHTS1	WGHTS2	WSST	WSNU
20,000	100.58	98.28	85.19	83.28	83.28	138.44
40,000	201.25	196.55	170.12	166.51	166.51	275.86
60,000	302.24	294.98	255.12	249.97	249.97	414.26
80,000	402.52	392.81	340.02	333.11	333.11	554.52
100,000	504.30	491.25	425.39	416.58	416.58	691.00

Table 6.6: Vector processing time of various codes to sample from Table 2.6, in milliseconds.

Samples	INVR2	DISCS	WGHTS1	WGHTS2	WSST	WSNU
20,000	18.23	10.98	8.42	7.02	7.02	36.06
40,000	36.38	21.83	16.81	13.93	13.93	71.93
60,000	54.72	32.74	25.28	20.90	20.90	107.72
80,000	72.75	43.65	33.53	27.89	27.89	143.52
100,000	90.94	54.53	42.07	34.80	34.80	179.93

Table 6.6 summarizes the processing time of different vector codes. The WGHTS2 and WSST codes require the least processing time, which is only about 38 % of INVER's processing time. In the scalar processing case, however, their processing

time is about 83 % of that of INVER. This indicates that the vectorizability of WGHTS2 and WSST is better than that of INVER. The processing time further increases for WGHTS1, DISCS, INVR2, and WSNU codes, respectively.

6.6.3 Speedups

This subsection investigates the minimum and maximum vectorization speedups of different vector codes. The objective here is to examine the range of possible speedups, which can be achieved by the vector codes of WSST and WSNU methods. As discussed in Subsection 3.4.3, the minimum vectorization speedups (V_{min}) of a vector code is obtained relative to the scalar code having the least processing time. The speedup of a vector code obtained relative to the scalar code having the largest processing time is referred to as the maximum vectorization speedup (V_{max}).

Table 6.7: Minimum vectorization speedups (V_{min}) of vector codes relative to the scalar WSST.

Samples	INVR2	DISCS	WGHTS1	WGHTS2	WSST	WSNU
20,000	4.568	7.585	9.891	11.863	11.863	2.309
40,000	4.577	7.628	9.905	11.953	11.953	2.315
60,000	4.568	7.635	9.888	11.960	11.960	2.321
80,000	4.579	7.631	9.935	11.944	11.944	2.321
100,000	4.581	7.639	9.902	11.971	11.971	2.315

Table 6.8: Maximum vectorization speedups (V_{max}) of vector codes relative to the scalar WSNU.

Samples	INVR2	DISCS	WGHTS1	WGHTS2	WSST	WSNU
20,000	7.594	12.608	16.442	19.721	19.721	3.839
40,000	7.583	12.637	16.410	19.803	19.803	3.835
60,000	7.571	12.653	16.387	19.821	19.821	3.846
80,000	7.622	12.704	16.538	19.882	19.882	3.864
100,000	7.598	12.672	16.425	19.856	19.856	3.840

Table 6.7 shows the vectorization speedups achieved by the various vector codes with respect to the scalar WSST. These speedups represent the minimum vectorization speedups, since the scalar WSST code requires the smallest scalar processing time. This table demonstrates that, as the sample size increases, the speedups of the various sampling methods are almost constant. This can be explained by the fact that the relationship between the sample size and the scalar as well as vector processing time is linear. Note that the vector codes of WGHTS2 and WSST entail the highest speedups, while WSNU attains the lowest speedups for different sample size.

The vectorization speedups of different vector codes with respect to the scalar WSNU code are shown in in Table 6.8. Since the processing time of the scalar WSNU code is the smallest, these speedups represents V_{max}. As the case of the V_{min} speedups, the V_{max} speedups for different sample sizes are also almost constant. One can notice that these speedups are about 1.66 times larger than the corresponding V_{min} speedups since the scalar WSNU's processing time is about 1.66 times larger than the scalar WGHTS2's processing time.

6.6.4 Efficiency

This subsection summarizes the relative efficiencies of various scalar and vector codes. These relative efficiencies are obtained with respect to the inverse method, since it is the most commonly used method. Therefore, the relative efficiencies of different scalar codes are obtained with respect to the scalar INVER code, and those of vector codes are obtained with respect to the vector INVR2 code.

The minimum and maximum relative efficiencies of different vector codes are also presented in this subsection. The objective here is to investigate the range of possible relative efficiencies, which can be achieved by various vector codes and especially WSST and WSNU methods. As discussed in Subsection 2.7, the minimum relative efficiency, denoted by η_{min}, is obtained with respect to the highest efficiency of a scalar code. The maximum relative efficiency (η_{max}) is obtained using the lowest

efficiency of a scalar code as a reference. It can be noted that these efficiencies are calculated based on the sample variances, which are obtained from Table 6.4 (see Equation (2.4)).

Table 6.9: Efficiencies of scalar codes relative to those of scalar INVER.

Samples	DISCS	WGHTS1	WGHTS2	WSST	WSNU
20,000	1.024	0.020	0.021	0.259	10.479
40,000	1.183	0.022	0.023	0.281	8.931
60,000	1.021	0.020	0.021	0.254	9.804
80,000	1.260	0.022	0.023	0.302	8.058
100,000	1.297	0.023	0.023	0.302	14.779

Table 6.9 presents the efficiencies of different scalar codes relative to the scalar INVER code. This table shows that as the sample size increases the relative efficiencies of DISCS, WSST and WSNU do not remain constant. The reason is that the decrease in the sample variances of INVER, DISCS, WSST and WSNU does not have the same pattern. One can notice that WSST achieves higher efficiencies than those of the direct weighted sampling codes (WGHTS1 and WGHTS2). Its relative efficiencies however are still lower than one. This means that the scalar speedup achieved by WSST relative to INVER is still smaller than the increase in the sample variance of WSST.

In contrast with WSST, for all sample sizes, WSNU entails the highest relative efficiencies even though its processing time is the largest. This implies that WSNU results in the smallest sample variance. Thus, the WSNU method reduces significantly the sample variance of the direct weighted sampling method.

As shown in Table 6.10, the relative efficiencies of all vector codes, except for WSNU, are larger than those of the corresponding scalar codes shown in the preceding table. The reason is that these codes attain higher vectorization speedups than INVR2, while WSNU entails lower speedups. The WSNU code results in the highest relative efficiencies, since for different sample sizes its sample variances are

the smallest.

The minimum and maximum relative efficiencies of different vector codes are given in Tables 6.11 and 6.12, respectively. The minimum relative efficiencies are obtained relative to the efficiencies of the scalar WSNU, which gives the highest scalar efficiencies for all sample sizes. The maximum relative efficiencies, on the other hand, are obtained relative to the lowest efficiencies of the scalar code, which is provided by WGHTS. These relative efficiencies show the range of possible efficiencies that one can achieve through vectorization and implementation of a good sampling method.

In Table 6.11, the relative efficiencies of the sampling methods are less than their corresponding vectorization speedups shown in Table 6.8, except for WSNU. This can be explained by the fact that their vectorization speedups are less than the increase in their sample variances. The relative efficiencies of WSNU here are the same as the vectorization speedups, since its scalar and vector codes have the same magnitudes of sample variances.

Table 6.12 reports the efficiencies of various vector codes relative to the efficiencies of scalar WGHTS1. Since the efficiency of scalar WGHTS1 is the lowest, these relative efficiencies represent η_{max}. The relative efficiencies of WGHTS1 and WGHTS2 are the same as their vectorization speedups, since the sample variances of their scalar and the corresponding vector codes have the same magnitudes. In comparison with the corresponding minimum relative efficiencies, these maximum relative efficiencies are about 500 times higher. This is due to the fact that the efficiency of scalar

Table 6.10: Efficiencies of vector codes relative to those of vector INVR2.

Samples	DISCS	WGHTS1	WGHTS2	WSST	WSNU
20,000	1.662	0.037	0.045	0.556	7.292
40,000	1.926	0.041	0.049	0.608	6.191
60,000	1.666	0.037	0.045	0.549	6.826
80,000	2.049	0.041	0.049	0.652	5.627
100,000	2.107	0.041	0.050	0.653	10.235

Table 6.11: Minimum relative efficiencies (η_{min}) of vector codes relative to the efficiency of scalar WSNU.

Samples	INVR2	DISCS	WGHTS1	WGHTS2	WSST	WSNU
20,000	0.527	0.875	0.020	0.024	0.293	3.839
40,000	0.619	1.193	0.025	0.031	0.376	3.835
60,000	0.563	0.938	0.021	0.025	0.310	3.846
80,000	0.687	1.407	0.028	0.034	0.448	3.864
100,000	0.375	0.791	0.016	0.019	0.245	3.840

WSNU is about 500 times higher than that of WGHTS1. Based on these results one can notice that weighted sampling using a stretched table increases the efficiency of the direct weighted sampling at most by a factor of 16, while the efficiency of the weighted sampling employing a geometric distribution increases at most by a factor of 247.

6.6.5 Remarks

The weighted sampling using a stretched table (WSST) and the weighted sampling employing a geometric distribution (WSNU) have been implemented in this section. The efficiency of WSST is found to be lower than that of the inverse code. The WSNU significantly reduced the sample variance of its solution, hence it resulted in the highest efficiency. Based on these results we conclude that the vector processing

Table 6.12: Maximum relative efficiencies (η_{max}) of vector codes relative to the efficiency of scalar WGHTS1.

Samples	INVR2	DISCS	WGHTS1	WGHTS2	WSST	WSNU
20,000	270.391	449.339	10.118	12.135	150.380	1971.625
40,000	247.879	477.343	10.120	12.212	150.666	1534.728
60,000	272.686	454.193	10.092	12.207	149.840	1861.471
80,000	249.524	511.315	10.141	12.191	162.756	1404.082
100,000	244.326	514.872	10.111	12.224	159.437	2500.612

together with the right choice of a sampling method can significantly enhance the resulting efficiency.

6.7 MORSE Problem I

MORSE Problem I has been discussed in Section 5.7. In this problem, the MORSE code was used to estimate the fast-neutron fluence at several radial distances from a point isotropic source in an infinite medium of air (see the illustration in Figure 5.2). This problem is reexamined here since the direct weighted sampling resulted in comparatively low efficiencies in the previous investigation. Thus, the objective here is to enhance the efficiencies of Monte Carlo solutions by incorporating the INWS method into the MORSE code.

The scattering probability matrix of air is shown in Table 5.3. For each group within energy groups 1 to 9, only the first three mass points have much higher probabilities than the remaining probabilities. In the INWS method employed, therefore, the inverse method is used for the first three mass points and the direct weighted sampling is employed for the subsequent mass points.

It should be noted here that the inverse method is still used for sampling scattering angles after collisions, since the number of discrete angles in this problem is at most three and the corresponding probabilities vary significantly.

This section examines the performance of the MORSE code incorporating the INWS method, and compare it with that of the other sampling methods. For this purpose, the performance of the inverse method (INVER) and the direct weighted sampling method (WGHTS), reported in Section 5.7, are shown again here as ready references.

6.7.1 Fluence Estimates

Table 6.13 summarizes the total fluence crossing Shell 1. This table shows that, for all sample sizes, the estimated fluence obtained by INVER, WGHTS and INWS are very close in magnitude, while their 99 % confidence intervals overlap each other. The FSDs of INWS decreases as the sample size increases. Moreover, the convergence of INWS's FSD is faster than those of the other sampling methods. These facts indicate that the solutions of INWS are unbiased. Generally, INWS results in the smallest FSD, which enhances the performance of the direct weighted sampling by reducing the sample variance.

Table 6.13: Fluence of neutrons crossing Shell 1 (30 m in radius) of Problem I.

Sample size	INVER	WGHTS	INWS
	Estimate(FSD)	Estimate(FSD)	Estimate(FSD)
20,000	1.3774(0.0175)	1.3435(0.0227)	1.3560(0.0186)
	±0.062*	±0.079	±0.065
40,000	1.3752(0.0109)	1.3253(0.0147)	1.3466(0.0120)
	±0.039	±0.050	±0.042
60,000	1.3787(0.0120)	1.3447(0.0173)	1.3389(0.0093)
	±0.043	±0.060	±0.032
80,000	1.3649(0.0114)	1.3296(0.0144)	1.3426(0.0085)
	±0.040	±0.049	±0.029
100,000	1.3596(0.0097)	1.3317(0.0132)	1.3407(0.0076)
	±0.034	±0.045	±0.026

* statistical variability corresponds to a 99 % confidence interval.

In Table 6.14, the FSD of INWS decreases as the sample size increases. This indicates that its estimated fluence crossing Shell 2 converges to the expected solution. The INVER and INWS codes result in almost the same estimated values. The INWS's results are therefore unbiased. Moreover, this table shows that INWS successfully reduces the FSDs of WGHTS. The FSD of INWS converges faster than that of INVER.

Table 6.14: Fluence of neutrons crossing Shell 2 (200 m in radius) of Problem I.

Sample size	INVER Estimate(FSD)	WGHTS Estimate(FSD)	INWS Estimate(FSD)
20,000	2.1078(0.0249)	1.9839(0.1173)	2.1457(0.0293)
	±0.135*	±0.599	±0.162
40,000	2.0480(0.0178)	1.8611(0.0688)	2.1283(0.0191)
	±0.094	±0.330	±0.105
60,000	2.0496(0.0148)	1.8640(0.0562)	2.0866(0.0163)
	±0.078	±0.270	±0.088
80,000	2.0671(0.0143)	1.8510(0.0471)	2.0967(0.0145)
	±0.076	±0.225	±0.078
100,000	2.0776(0.0145)	1.8137(0.0399)	2.0894(0.0131)
	±0.078	±0.186	±0.071

* statistical variability corresponds to a 99 % confidence interval.

Table 6.15: Fluence of neutrons crossing Shell 3 (450 m in radius) of Problem I.

Sample size	INVER Estimate(FSD)	WGHTS Estimate(FSD)	INWS Estimate(FSD)
20,000	1.1906(0.03296)	1.0298(0.20122)	1.1683(0.03171)
	±0.1011*	±0.5338	±0.0954
40,000	1.1855(0.02305)	0.9170(0.14068)	1.1854(0.02305)
	±0.0704	±0.3323	±0.0704
60,000	1.1712(0.01868)	0.9319(0.11716)	1.1556(0.01792)
	±0.0564	±0.2813	±0.0533
80,000	1.1741(0.01810)	0.8849(0.09502)	1.1490(0.01570)
	±0.0547	±0.2166	±0.0465
100,000	1.1751(0.01612)	0.9123(0.07903)	1.1523(0.01474)
	±0.0488	±0.1857	±0.0438

* statistical variability corresponds to a 99 % confidence interval.

Tables 6.15 shows the estimated fluence crossing Shell 3. In this table, for all sample sizes, the estimated values of INVER and INWS are very close to each other. The confidence intervals of INWS overlap with those of INVER. This indicates that the estimated values obtained by INWS are unbiased. The INWS method entails the smallest FSD, except for 40,000 samples the FSD of INWS is the same as that of INVER. For sample size of 40,000, the FSDs of INVER and INWS are the same.

6.7.2 Processing Time and Speedups

Table 6.16 shows that, on average, the scalar processing time required by INWS is about 87 % of that of INVER. As shown in Table 6.17, the number of collisions produced by INWS is about 87 % of the INVER's collisions. This indicates that the processing time of INVER and INWS is proportional to the number of collisions.

Table 6.16: Scalar processing time of MORSE simulation for Problem I.

Sample size	Processing time in seconds		
	INVER	WGHTS	INWS
20,000	79.278	25.117	65.406
40,000	155.973	49.674	135.474
60,000	228.802	75.793	206.196
80,000	305.631	101.569	273.012
100,000	378.767	125.696	344.577

The speedups of scalar WGHTS and INWS codes relative to scalar INVER code are summarized in Table 6.18. These speedups are calculated based on the scalar processing time shown in Table 6.16. On average, the INWS code is about 14 % faster than INVER. One can notice that the speedup of INWS is lower than that of WGHTS. This is due to the fact that INWS requires more processing time than WGHTS.

Table 6.17: Number of collisions in MORSE simulation for Problem I.

Sample size	Number of collisions		
	INVER	WGHTS	INWS
20,000	384,994	99,118	334,236
40,000	761,302	197,157	667,772
60,000	1,142,531	295,382	998,583
80,000	1,521,797	392,965	1,331,179
100,000	1,903,822	490,279	1,662,201

Table 6.18: Speedups of scalar MORSE codes relative to scalar INVER for Problem I.

Sample size	Scalar speedups	
	WGHTS	INWS
20,000	3.156	1.212
40,000	3.140	1.151
60,000	3.019	1.110
80,000	3.009	1.119
100,000	3.013	1.099

6.7.3 Local Speedups

In this subsection, the processing time of various sampling methods in the subroutine COLISN is investigated. The objective is to obtain the local speedups in the subroutine COLISN. Thus, these speedups provide a direct measure of the speeding of the sampling methods, without the additional overhead of other routines in the MORSE code.

Table 6.19: Scalar processing time of sampling methods in COLISN routine for Problem I.

Sample size	Processing time in milliseconds		
	INVER	WGHTS	INWS
20,000	1263.860	1251.416	1253.466
40,000	2530.249	2509.246	2506.891
60,000	3791.720	3756.862	3761.909
80,000	5054.139	5006.014	5012.591
100,000	6318.385	6258.126	6267.143

Table 6.19 shows the scalar processing time of various sampling methods. One can notice that the processing time of INWS is proportional to the sample size, as that of the other sampling methods. In general, the processing time of INWS is slightly larger than that of WGHTS, while the processing time of INWS is smaller than that of INVER. This indicates that the direct weighted sampling method in INWS method can reduce the processing time of the inverse method.

In Table 6.19, the processing time of INVER is larger than that of INWS by a factor of 0.8 %. However, Table 6.18 shows that the INWS method entailed global speedup by a factor of 1.14. Thus, the global speedup of the MORSE code incorporating the INWS method is mostly affected by the reduction in the number of collisions.

6.7.4 Local Vectorization Speedups

This subsection investigates the vectorization of various sampling methods in subroutine COLISN. As discussed in Subsection 5.7.4, this local speedup should provide some indications of the global vectorization speedup if the MORSE code is vectorized based on an event-based Monte Carlo algorithm.

Table 6.20: Vector processing time of sampling methods in COLISN routine for Problem I.

Sample size	Processing time in milliseconds		
	INVER	WGHTS	INWS
20,000	364.371	136.364	369.926
40,000	729.298	273.995	737.231
60,000	1095.980	412.343	1106.003
80,000	1460.923	545.436	1474.755
100,000	1824.385	684.846	1863.324

Table 6.21: Speedups of the vectorized sampling methods relative to scalar INVER for Problem I.

Sample size	Vector speedups		
	INVER	WGHTS	INWS
20,000	3.469	9.177	3.388
40,000	3.469	9.158	3.400
60,000	3.460	9.111	3.401
80,000	3.460	9.178	3.399
100,000	3.463	9.138	3.363

Table 6.20 summarizes the processing time of vector INVER, WGHTS and INWS. For all the sample sizes, the least processing time is for WGHTS, followed by INVER and INWS. This order is not the same as that in the scalar processing. The processing time of vector INWS is slightly larger than that of INVER. This can be explained as follows. As shown in Figure 6.6, the vector code of the INWS method has two iterations used for the direct weighted sampling method and the inverse method,

respectively. Since the first three mass points have much higher probabilities than the remaining probabilities, the inverse method is often used. Consequently, the extra processing time spent by INWS becomes higher since more unnecessary sampling is carried out by the weighted sampling method in the first iteration. Thus, the extra processing time required by INWS is larger than the processing time reduction attained by the direct weighted sampling in this code.

Table 6.21 shows the speedups of different vector codes relative to scalar INVER code. One can notice that for all sample sizes the speedups of INWS are about constant. The speedup of INWS is about 3.4, while that of INVER is about 3.5.

6.7.5 Efficiency

Tables 6.22 to 6.24 report the efficiencies of WGHTS and INWS relative to those of INVER, for shells 1, 2 and 3, respectively. Note that the efficiencies of WGHTS, taken from Tables 5.15 to 5.17, are shown again as ready references.

In Table 6.22, on average, the relative efficiencies of WGHTS and INWS are about 1.8 and 1.6, respectively. The WGHTS code entails higher efficiency than that of INWS, since it has higher scalar speedup.

Table 6.22: Efficiencies of MORSE codes relative to those of INVER, for Shell 1 of Problem I.

Sample size	Relative Efficiency	
	WGHTS	INWS
20,000	1.967	1.104
40,000	1.845	0.991
60,000	1.515	1.961
80,000	1.962	2.087
100,000	1.696	1.832

As shown in Table 6.23, on average, the relative efficiencies of WGHTS and INWS are 0.31 and 1.01, respectively. The INWS code increases the efficiency of the direct

weighted sampling by reducing the sample variance. The efficiency of INWS is slightly higher than that of INVER due to the fact that INWS achieves higher speedup than that of INVER.

Table 6.23: Efficiencies of MORSE codes relative to those of INVER, for Shell 2 of Problem I.

Sample size	Relative Efficiency	
	WGHTS	INWS
20,000	0.161	0.847
40,000	0.254	0.924
60,000	0.253	0.879
80,000	0.344	1.054
100,000	0.522	1.324

Table 6.24: Efficiencies of MORSE codes relative to those of INVER, for Shell 3 of Problem I.

Sample size	Relative Efficiency	
	WGHTS	INWS
20,000	0.113	1.360
40,000	0.141	1.152
60,000	0.121	1.239
80,000	0.192	1.554
100,000	0.208	1.367

. Table 6.24 shows that, on average, the relative efficiencies of WGHTS and INWS are 0.16 and 1.34, respectively. For the estimated fluence crossing Shell 3, one can notice that INWS significantly increases the efficiency of direct weighted sampling by a factor of about 8.4. This is achieved by reducing the sample variance.

As discussed in Subsection 5.7.5, the relative efficiencies of DISCS and WGHTS decrease significantly with respect to the increase in shell's radii. However, the relative efficiencies of INWS have a different trend. In general, the efficiencies of INWS relative to those of INVER for Shells 1 to 3 are 1.6, 1.01 and 1.34, respectively. The

interesting fact here is that the relative efficiency of INWS for Shell 3 is higher than that of Shell 2. This can be explained by the fact that the direct weighted sampling method is used more often at low energy, in which the transition probabilities are close to uniform. Consequently, the statistical variability of the adjustment factors is low.

Table 6.25: Efficiencies of MORSE codes relative to those of INVER, for about 5 % FSD of Problem I.

Shell	Relative Efficiency	
	WGHTS	INWS
Shell 1	0.452	1.038
Shell 2	0.071	1.242
Shell 3	0.037	1.360

Table 6.25 summarizes the efficiencies of WGHTS and INWS relative to those of INVER for FSD of about 5 %. One can notice that the relative efficiency of INWS increases with the increase in the shell's radius. This supports the argument in the previous paragraph. The efficiency of INWS relative to that of INVER is equal to 1.213, on average. Thus, INWS increases the efficiency of WGHTS by a factor of 6.5, and also performs better than INVER.

In summary, the efficiencies of INWS are higher than those of INVER for all radii. The relative efficiency of INWS generally increases with the increase in the shell's radius. This indicates the attractiveness of INWS over the other sampling methods, if the simulations involve shells with long radii.

6.7.6 Remarks

In this section, the combination of the inverse and the direct weighted sampling methods (INWS) significantly enhanced the efficiency of the direct weighted sampling method. Moreover, the efficiency obtained by INWS was higher than that of INVER due to the smaller processing time and the smaller sample variance. In general, the

relative efficiency of INWS increases as the shell's radius increases. This indicates that INWS performs well if more collisions occur at low energy.

6.8 MORSE Problem II

This problem has been discussed in Section 5.8. The illustration is depicted in Figure 5.3. In this problem, a fast neutron beam is directed towards the test section, which is water in the form of a cylindrical body. The test section is surrounded by air, which is in turn surrounded by artificial external void. The transport of neutrons is terminated when the neutrons reach external void. There are two point detectors located perpendicular to both the neutron beam and the axis of the cylindrical test section.

In this section, MORSE Problem II is reinvestigated since the direct weighted sampling resulted in relatively low efficiencies. Thus, the objective here is to examine whether the efficiencies of Monte Carlo solutions can be increased by incorporating the INWSST method into the MORSE code.

The scattering probability matrix for the transport medium water is shown in Table 5.20. This matrix shows that the neutron cross section consists of 33 energy groups. In this probability matrix, only the first four mass points in the energy groups 1 to 10 have high probabilities, while the rest mass points have relatively low probabilities. Therefore, based on the previous section, the combination of the inverse method and the weighted sampling method can be used to enhance the efficiency of the Monte Carlo solutions.

For each energy group, the low probabilities (starting from the fifth mass point to the rest) vary significantly. These probabilities have to be stretched in order to reduce the sample variance. Therefore, we investigate the INWSST method in which the inverse method is used for the first four mass points and the weighted sampling using stretched tables is employed for the subsequent mass points.

It should be noted that the inverse method is still utilized for sampling scattering angles after collisions, since the number of discrete angles is at most five and the corresponding probabilities vary significantly.

6.8.1 Fluence Estimates

This section examines the performance of the MORSE code incorporating the INWS method, and compare it with that of the other sampling methods. For this purpose, the performance of the inverse method (INVER) and the direct weighted sampling method (WGHTS), reported in Section 5.8, are shown again here as ready references.

Tables 6.26 and 6.27 show the estimated response of Detectors 1 and 2, respectively. The estimated values of detector response are reported together with the fractional standard deviations and the 99 % confidence intervals. In these tables, the FSDs of INWSST decrease as the sample size increases. This indicates that the solutions converge to the expected solution. INWSST results in slightly higher FSDs than those of INVER. However, the FSDs of INWSST are much lower than those of WGHTS for all sample sizes. This fact indicates that INWSST reduces significantly the sample variance of the direct weighted sampling method.

As shown in Tables 6.26 and 6.27, the estimated values of INVER and INWSST are slightly different, but their confidence intervals overlap each other. The estimated values of detector response obtained by INWSST are therefore unbiased.

Table 6.26: Response of Detector 1 for Problem II.

Sample size	INVER	WGHTS	INWSST
	Response(FSD)	Response(FSD)	Response(FSD)
20,000	9.4123E-05(0.01798)	9.5923E-05(0.23082)	9.2905E-05(0.01930)
	±0.436E-5*	±5.704E-5	±0.462E-5
40,000	9.3477E-05(0.01304)	9.4677E-05(0.12472)	9.1406E-05(0.01347)
	±0.314E-5	±3.042E-5	±0.317E-5
60,000	9.4070E-05(0.01067)	1.0228E-04(0.15601)	9.2022E-05(0.01094)
	±0.259E-5	±4.110E-5	±0.259E-5
80,000	9.4417E-05(0.00921)	9.5690E-05(0.12650)	9.1579E-05(0.00930)
	±0.224E-5	±3.118E-5	±0.219E-5
100,000	9.3946E-05(0.00827)	9.1419E-05(0.10667)	9.1395E-05(0.00920)
	±0.200E-5	±2.512E-5	±0.217E-5

* statistical variability corresponds to a 99 % confidence interval.

Table 6.27: Response of Detector 2 for Problem II.

Sample size	INVER	WGHTS	INWSST
	Response(FSD)	Response(FSD)	Response(FSD)
20,000	9.4501E-05(0.01819)	9.4539E-05(0.14149)	9.3001E-05(0.01807)
	±0.443E-5*	±3.446E-5	±0.433E-5
40,000	9.3784E-05(0.01249)	1.1238E-04(0.18857)	9.2817E-05(0.01283)
	±0.302E-5	±5.459E-5	±0.307E-5
60,000	9.4387E-05(0.01062)	9.9699E-05(0.14321)	9.1431E-05(0.01098)
	±0.258E-5	±3.678E-5	±0.259E-5
80,000	9.3646E-05(0.00916)	9.8497E-05(0.11364)	9.2488E-05(0.00945)
	±0.221E-5	±2.883E-5	±0.225E-5
100,000	9.3601E-05(0.00831)	9.7451E-05(0.09426)	9.2852E-05(0.00841)
	±0.200E-5	±2.366E-5	±0.201E-5

* statistical variability corresponds to a 99 % confidence interval.

6.8.2 Processing Time and Speedups

Table 6.28 shows the scalar processing time of MORSE codes employing different sampling methods. This table shows that INWSST requires slightly less processing time than that of INVER. As shown in Table 6.29, the number of collisions produced by INWSST is also slightly less than that of INVER. One can infer therefore that the processing time required by INVER and INWSST for each collision is about the same. Thus, the processing time reduction in INWSST is caused by the reduction in the number of collisions.

Table 6.28: Scalar processing time of MORSE simulation for Problem II.

Sample size	Processing time in seconds		
	INVER	WGHTS	INWSST
20,000	19.454	13.793	19.317
40,000	39.397	28.021	38.649
60,000	59.013	42.692	58.124
80,000	78.973	55.019	78.378
100,000	97.457	69.817	96.318

Table 6.29: Number of collisions in MORSE simulation for Problem II.

Sample size	Number of collisions		
	INVER	WGHTS	INWSST
20,000	22,060	14,068	21,645
40,000	44,093	28,188	43,477
60,000	65,852	42,135	65,543
80,000	87,740	55,966	87,528
100,000	109,040	69,894	109,367

Table 6.30 shows scalar speedups of WGHTS and INWSST relative to INVER. The speedup of INWSST for different sample sizes is constant, since the processing time of INVER and INWSST is linearly proportional to the sample size. The speedup of INWSST is equal to one. This implies that the inverse method in INWSST is used

more often than the weighted sampling method, since most of the collided neutrons have high energy during the simulation.

Table 6.30: Speedups of scalar MORSE codes relative to scalar INVER for Problem II.

Sample size	Speedups	
	WGHTS	INWSST
20,000	1.410	1.007
40,000	1.406	1.019
60,000	1.382	1.015
80,000	1.435	1.008
100,000	1.396	1.012

6.8.3 Local Speedups

The processing time performance of different sampling methods in subroutine COLISN is investigated in this subsection. The scalar processing time of Brown's and the INWSST methods is compared to that of the inverse method, with the objective to obtain the local speedups. These speedups provide a direct measure of the speeding of the sampling methods, without the additional overhead of other routines in the MORSE code.

Table 6.31: Scalar processing time of sampling methods in COLISN routine for Problem II.

Sample size	Processing time in milliseconds		
	INVER	WGHTS	INWSST
20,000	1718.969	1612.504	1712.627
40,000	3436.748	3224.210	3430.066
60,000	5158.662	4842.191	5140.457
80,000	6877.693	6447.755	6851.574
100,000	8598.855	8068.646	8563.725

The processing time of the scalar codes is shown in Table 6.31. In order to

compare the performance of INWSST, the processing time of INVER and WGHTS from Subsection 5.8.3 is reported again as ready references. The processing time of INWSST is linearly proportional to the sample size. For all sample sizes, the processing time of INWSST is only slightly less than that of INVER but larger than that of WGHTS. Therefore, the processing time of INVER and INWSST to construct a sample is about the same. This supports the argument that the processing times of INVER and INWSST for each sampling after a collision are close to each other. Thus, one can conclude that the global scalar speedup of INWSST relative to INVER is one.

6.8.4 Local Vectorization Speedups

The vectorization of various sampling methods in the COLISN routine is examined in this subsection. The objective is to investigate the local vectorization speedup of INWSST, and to compare it with those of INVER and WGHTS. This speedup should be able to provide some predictions of the global vectorization speedup one would achieve if the entire MORSE code is vectorized by implementing an event-based Monte Carlo algorithm.

Table 6.32: Vector processing time of sampling methods in COLISN routine for Problem II.

Sample size	Processing time in milliseconds		
	INVER	WGHTS	INWSST
20,000	522.803	165.880	392.098
40,000	1045.616	332.751	783.381
60,000	1569.946	501.057	1172.598
80,000	2093.273	663.265	1563.363
100,000	2615.586	833.938	1971.798

The processing time of vector INVER, WGHTS and INWSST is shown in Table 6.32. For all sample sizes, the least processing time is for WGHTS, followed by

INWSST and INVER. This order remains the same as that in the scalar processing time. However, the processing time of vector INWSST is only 75 % of that of vector INVER; this is not the case of the scalar processing. Thus, this implies that the vectorizability of the INWSST method is higher than that of INVER.

Table 6.33: Speedups of the vectorized sampling methods relative to scalar INVER for Problem II.

Sample size	Speedups		
	INVER	WGHTS	INWSST
20,000	3.288	9.721	4.368
40,000	3.287	9.690	4.379
60,000	3.286	9.664	4.384
80,000	3.286	9.721	4.383
100,000	3.288	9.675	4.343

Table 6.33 reports the speedups of different vector codes relative to scalar INVER. The speedups of INWSST for all sample sizes are about constant, due to the linear relationship between the sample size and the processing time in scalar as well as vector processing. It is shown that the speedup of INWSST is higher than that of INVER. This indicates that the weighted sampling method in INWSST code significantly reduces the vector processing time. Thus, this fact provides some indications of the ability of the INWSST method to speedup the construction of samples in a vector MORSE code while maintaining a small sample variance of its solution.

6.8.5 Efficiency

Tables 6.34 and 6.35 reports the efficiencies of WGHTS and INWSST relative to those of INVER for Detectors 1 and 2, respectively. In Table 6.34, for sample size of 20,000 and 100,000, the relative efficiencies of INWSST are about 12 % lower than those of INVER. The reason is that the processing times of INVER and INWSST are about the same, while the sample variances of INWSST for these sample sizes are larger

than those of INVER.

The average of the relative efficiency of INWSST for both detectors is about 0.985. This magnitude is 104 times higher than that of WGHTS. Thus, the combination of the inverse method and the weighted sampling using a stretched table increases significantly the efficiency of WGHTS.

The efficiency of INWSST is close to that of INVER. This indicates that the performance of INWSST approaches that of INVER. This can be explained by the fact that the inverse method in INWSST is more often used since more collisions occur at high energy.

Table 6.34: Efficiencies of MORSE codes relative to those of INVER, for Detector 1 of Problem II.

Sample size	Relative Efficiency	
	WGHTS	INWSST
20,000	0.008	0.897
40,000	0.015	0.999
60,000	0.005	1.009
80,000	0.007	1.050
100,000	0.009	0.864

Table 6.35: Efficiencies of MORSE codes relative to those of INVER, for Detector 2 of Problem II.

Sample size	Relative Efficiency	
	WGHTS	INWSST
20,000	0.023	1.054
40,000	0.004	0.986
60,000	0.007	1.012
80,000	0.008	0.971
100,000	0.010	1.004

For a FSD of about 5 %, the efficiencies of WGHTS and INWSST relative to those of INVER are given in Table 6.36. For both detectors, the efficiency of INWSST relative to that of INVER is equal to 0.768, which is much larger than that of WGHTS. Thus, the combination of the inverse method and the weighted sampling using a stretched table also significantly increases the efficiency of the direct weighted sampling for a given FSD.

Table 6.36: Efficiencies of MORSE codes relative to those of INVER, for about 5 % FSD of Problem II.

Detector	Relative Efficiency	
	WGHTS	INWSST
Detector 1	*	0.697
Detector 2	*	0.838

*: the relative efficiency is not available.

6.8.6 Remarks

This section reexamined MORSE Problem II, in which the responses of two point detectors were estimated. The performance of INWSST approaches that of INVER due to the fact that the inverse method in INWSST is more often used since more collisions occur at high energy. It was found that the efficiencies of solutions obtained by INWSST are much higher than those of WGHTS. Thus, the combination of the inverse method and the weighted sampling using a stretched table increases significantly the efficiency of the direct weighted sampling method.

6.9 MORSE Problem III

This problem has the same physical characteristics as those of MORSE Problem II, except for the water density. The water density here is four times higher than that of Problem II. As discussed in Section 5.9, direct weighted sampling resulted in low efficiency for the solution of MORSE Problem III. The combination of the inverse method and the weighted sampling using a stretched table (INWSST) is utilized in this problem to examine the effect on the solution variance and the processing time. The INWSST code used here is the same as the code used in the previous section, since the scattering probability tables involved are the same.

6.9.1 Fluence Estimates

Tables 6.37 and 6.38 demonstrate the estimated response of Detectors 1 and 2, respectively. The estimated values of both detector responses are reported together with the fractional standard deviations and the 99 % confidence intervals. In these tables, the FSDs of INWSST decrease as the sample size increases. This indicates that the estimated values converge to the expected solution. Note that the FSDs of INWSST are smaller than those of INVER for all sample sizes. This indicates that INWSST significantly reduces the sample variance of the direct weighted sampling method.

In Tables 6.37 and 6.38, the confidence intervals of INWSST overlap those of INVER for all sample sizes. Therefore, for both detectors the estimated responses of INWSST are unbiased.

An interesting fact about INWSST is that, for each sample size, the estimated values of the two detectors are very close to each other. This can be explained by the fact that the statistical weights of collided neutrons in INWSST do not vary significantly. Consequently, the sample variances of the estimated values for both detectors are small, which is not the case of WGHTS.

Table 6.37: Response of Detector 1 for Problem III.

Sample size	INVER Response(FSD)	WGHTS Response(FSD)	INWSST Response(FSD)
20,000	2.9084E-04(0.03503)	3.1197E-04(0.23935)	2.9026E-04(0.02671)
	±0.262E-4*	±1.923E-4	±0.200E-4
40,000	2.9783E-04(0.02320)	2.6646E-04(0.14501)	2.9036E-04(0.01946)
	±0.178E-4	±0.995E-4	±0.146E-4
60,000	2.9630E-04(0.01865)	2.5932E-04(0.10446)	2.8841E-04(0.01581)
	±0.142E-4	±0.698E-4	±0.118E-4
80,000	2.9853E-04(0.01594)	2.6476E-04(0.08875)	2.8989E-04(0.01363)
	±0.123E-4	±0.605E-4	±0.102E-4
100,000	2.9730E-04(0.01390)	2.5323E-04(0.07496)	2.8772E-04(0.01206)
	±0.107E-4	±0.489E-4	±0.089E-4

* statistical variability corresponds to a 99 % confidence interval.

Table 6.38: Response of Detector 2 for Problem III.

Sample size	INVER Response(FSD)	WGHTS Response(FSD)	INWSST Response(FSD)
20,000	2.9208E-04(0.02657)	3.0364E-04(0.15015)	2.9228E-04(0.02613)
	±0.200E-4*	±1.174E-4	±0.197E-4
40,000	2.9342E-04(0.02086)	2.4555E-04(0.09678)	2.8737E-04(0.01914)
	±0.158E-4	±0.612E-4	±0.142E-4
60,000	2.9011E-04(0.01607)	2.4281E-04(0.07404)	2.9062E-04(0.01581)
	±0.120E-4	±0.463E-4	±0.118E-4
80,000	2.9293E-04(0.01385)	2.4427E-04(0.06348)	2.8677E-04(0.01354)
	±0.105E-4	±0.399E-4	±0.100E-4
100,000	3.0329E-04(0.03637)	2.3885E-04(0.05381)	2.8553E-04(0.01203)
	±0.284E-4	±0.331E-4	±0.088E-4

* statistical variability corresponds to a 99 % confidence interval.

6.9.2 Processing Time and Speedups

The scalar processing time of the three MORSE codes is shown in Table 6.39. The processing time of INWSST is linearly proportional to the sample size. The INWSST code requires more processing time than the INVER code, even though it simulates fewer collisions, as shown in Table 6.40. The number of collisions produced by INWSST is about 2 % less than that of INVER. Therefore, for each collision, the processing time required by INWSST is slightly larger than that of INVER. This can be explained by the fact that, in this problem, a large number of collisions occur at low energy due to the high magnitude of water density, see Table 6.45 shown in the next section. This indicates that the weighted sampling using a stretched table is more often used for sampling. Thus, the processing time of INWSST is higher than that of INVER, since INWSST first carries out the inverse method, followed by the weighted sampling method. One can predict however that this combination will reduce the processing time of the inverse method if the probability table is relatively long.

Table 6.39: Scalar processing time of MORSE simulation for Problem III.

Sample size	Processing time in seconds		
	INVER	WGHTS	INWSST
20,000	97.623	32.531	101.206
40,000	198.854	64.354	200.103
60,000	295.307	97.408	300.030
80,000	394.268	126.900	398.273
100,000	495.063	160.957	496.878

The scalar speedups of WGHTS and INWSST relative to INVER are reported in Table 6.41. For different sample sizes, the speedups of INWSST is about constant. The reason is that the processing time of INVER, as well as INWSST, is linearly proportional to the sample size. The speedup of INWSST is less than one, since it requires slightly more processing time as discussed in the preceding paragraphs.

Table 6.40: Number of collisions in MORSE simulation for Problem III.

Sample size	Number of collisions		
	INVER	WGHTS	INWSST
20,000	188,438	47,121	185,973
40,000	377,499	94,478	370,496
60,000	570,484	141,905	557,141
80,000	759,479	189,367	741,368
100,000	949,270	236,425	925,001

The local scalar and vectorization speedups for this problem are not examined, since the results are the same as those of MORSE Problem II. This is due to the fact that the scattering probability tables used in this problem are the same as those in Problem II.

Table 6.41: Speedups of scalar MORSE codes relative to scalar INVER for Problem III.

Sample size	Speedups	
	WGHTS	INWSST
20,000	3.001	0.965
40,000	3.090	0.994
60,000	3.032	0.984
80,000	3.107	0.990
100,000	3.076	0.996

6.9.3 Efficiency

Tables 6.42 and 6.43 show the relative efficiencies of estimated response for Detectors 1 and 2, respectively. For different sample sizes, the relative efficiencies of INWSST for Detector 1 are higher than one. In Table 6.43, for all sample sizes except for 20,000 samples, the efficiency for Detector 2 obtained by INWSST is also larger than that of INVER.

Table 6.42: Efficiencies of MORSE codes relative to those of INVER, for Detector 1 of Problem III.

Sample size	Relative Efficiency	
	WGHTS	INWSST
20,000	0.056	1.666
40,000	0.099	1.486
60,000	0.126	1.446
80,000	0.127	1.436
100,000	0.146	1.413

Table 6.43: Efficiencies of MORSE codes relative to those of INVER, for Detector 2 of Problem III.

Sample size	Relative Efficiency	
	WGHTS	INWSST
20,000	0.087	0.996
40,000	0.205	1.231
60,000	0.204	1.013
80,000	0.213	1.081
100,000	2.266	10.275

In Table 6.43, the relative efficiency of INWSST for 100,000 samples is quite different. This discrepancy is caused by the fact that as the sample size increases from 80,000 to 100,000 the sample variance of INVER significantly increases, while the sample variance of INWSST is about the same.

Table 6.44: Efficiencies of MORSE codes relative to those of INVER, for about 5 % FSD of Problem III.

Detector	Relative Efficiency	
	WGHTS	INWSST
Detector 1	0.016	1.367
Detector 2	0.032	1.760

Table 6.45: The probability of collisions produced by INWSST in MORSE Problem II and III.

Problem	Energy group												
	8	9	10	11	12	13	14	15	16	17	18	19	20
II	.603	.045	.021	.032	.025	.024	.059	.055	.015	.027	.006	.007	.011
III	.137	.026	.014	.021	.019	.019	.045	.051	.017	.033	.010	.009	.019
Problem	Energy group												
	21	22	23	24	25	26	27	28	29	30	31	32	33
II	.009	.007	.010	.009	.007	.005	.004	.005	.004	.002	.002	.001	.004
III	.019	.018	.026	.024	.023	.022	.020	.031	.028	.020	.019	.018	.312

The average relative efficiency of INWSST for both detectors is 2.2. This is 6.2 times higher than that of WGHTS. Thus, in this problem, the combination of the inverse method and the weighted sampling using a stretched table successfully enhances the efficiency of the direct weighted sampling.

Table 6.44 summarizes the efficiencies of WGHTS and INWSST relative to those of INVER for FSD of about 5 %. The efficiency of INWSST relative to that of INVER is 1.564, on average. This magnitude is 65.2 times larger than the relative efficiency of WGHTS. Thus, INWSST increases significantly the efficiency for a particular fractional standard deviation by reducing the sample variance of the estimated response.

In this problem, the relative efficiency of INWSST is higher than that of MORSE Problem II. This can be explained by the fact that the number of collisions for each energy group in both simulations are different; the collisions occur in energy groups 8 to 33. Table 6.45 shows the frequency of collisions occured in energy groups 8 to 33 for each neutron source. One can notice that Problem III has more number of

collisions occuring at low energy groups than Problem II, since the transport medium in Problem III has higher density than that of Problem II. In this context, the relative efficiency of INWSST in Problem III is 2.23 times higher than that of MORSE Problem II. For a FSD of about 5 %, the relative efficiency of INWSST in Problem III is also 2.03 times higher than that of Problem II. This indicates that the INWSST method works better in such a problem involving a large number of collisions at low energy, in which the weighted sampling method in INWSST is used more often. Thus, INWSST achieves better performance when the problem involves a large number of collisions at low energy.

Based on this result, one can conclude that the performance of INWSST approaches that of INVER if the simulation involves a large number of collisions at high energy. However, INWSST performs better than INVER if more collisions occur at low energy. One can expect, therefore, that the performance of INWSST will approach that of INVER if the transport medium has a high mass number, since the inverse method in INWSST will be more often employed due to the fact that a large number of collisions will occur at high energy.

6.9.4 Remarks

In this problem, the density of water is higher than that of MORSE Problem II. This problem therefore involves more number of collisions at low energy than in Problem II. It was found that INWSST entails unbiased estimated response for both detectors. The INWSST code increased significantly the efficiency of the direct weighted sampling. Moreover, INWSST resulted in higher efficiency than that of INVER. The relative efficiency of INWSST in this problem was higher than that of Problem II. This indicates that the combination of the inverse method and weighted sampling using stretched tables is effective in reducing the sample variance when the problem involves a large number of collisions at low energy.

6.10 Conclusions

This chapter proposed some variants of the direct weighted sampling. In the problem involving a one-dimensional probability table, the weighted sampling employing a geometric distribution entailed much higher efficiency than that of the direct weighted sampling as well as that of the inverse method. This was achieved by significantly reducing the sample variance. An effort was also made to reduce the processing time of generating geometric random numbers by increasing the vectorizability of the code.

For MORSE Problem I, the combination of the inverse and the weighted sampling methods (INWS) enhanced the performance of the direct weighted sampling. The INWS method resulted in higher efficiency than those of the inverse method and the direct weighted sampling method.

In MORSE Problems II and III, the MORSE code incorporating the combination of the inverse method and weighted sampling using stretched tables (INWSST) significantly improved the performance of the direct weighted sampling method. In Problem II the efficiency of INWSST was about the same as that of the inverse method, while in Problem III its efficiency was higher than that of the inverse method. This is due to the smaller sample variance obtained by the INWSST method. One can conclude, therefore, that the combination of the inverse method and weighted sampling using a stretched table performed better in such a problem involving a large number of collisions at low energy.

In this chapter, only the scalar processing of MORSE codes incorporating various sampling methods is examined, since in its present state the MORSE code is not vectorizable. However, the local vectorization speedup was investigated by measuring the scalar and vector processing time of different sampling methods in the COLISN routine. Thus, this local speedup provides some indications of the global vectorization speedup, if the entire MORSE code is vectorized by implementing an event-based Monte Carlo algorithm.

In summary, the performance of INWS and INWSST approaches that of INVER, if the simulation involves a large number of collisions at high energy. On the other hand, they perform better than INVER, if more collisions occur at low energy due to the fact that the weighted sampling in these methods will be more often employed. Finally, it should be emphasized that both methods are expected to significantly reduce the processing time, if the probability table is sufficiently long.

Chapter 7

Conclusion

7.1 Summary

In this thesis, we have investigated some of the existing methods for sampling from arbitrary discrete distributions represented as probability tables. The inverse method, the equiprobable method, the alias and Brown's methods were considered. Since these methods have some disadvantages, we have proposed a new sampling method, called the weighted sampling method, which is especially suited for vector processing. All these sampling methods have been applied to Monte Carlo solutions of a large sparse system of linear equations and neutron transport problems. The codes were written in VS Fortran and implemented on the IBM 3090-180VF. Their performance was evaluated based on the scalar as well as vector processing time, the fractional standard deviation, and the efficiency of the Monte Carlo solutions. In order to provide a measure of both the statistical and computational performance, the efficiency was defined in terms of both the processing time and the sample variance of a solution.

For the estimation of the distribution mean of a one-dimensional probability table, it was found that the computing time of the inverse method increases as the length of the probability table increases due to the searching process in the method. For a long probability table this method requires therefore much larger processing time compared

to the other methods. The equiprobable method requires the least processing time in scalar as well as vector processing. However, this method requires a large storage of size of 10^b if the smallest probability is a b-digit decimal number. The alias method is not fully vectorizable since it requires `if`-statements involving indirect addressing. Brown's method is suitable for vectorization as it does not involve any `if`-statements. It was found that the alias and Brown's methods require a complicated procedure to set up the tables for sampling, and these tables require storage of size three times of the probability table's size. All the existing sampling methods results in about the same sample variances.

In order to overcome the computational drawbacks of the existing methods, we have proposed a new sampling method for sampling from probability tables, named as the weighted sampling method. The weighted sampling method utilizes a uniform distribution to construct samples from a probability table. Subsequently, each sample is multiplied by an adjustment factor. Thus, this method eliminates searching, and contains only fetching and arithmetic operations. The scalar and vector codes of the weighted sampling method are simpler than those of the other methods. It was shown that the weighted sampling method enhances the vectorizability of a vector Monte Carlo code, and achieves therefore the highest vectorization speedups.

For estimating the mean of a one-dimensional probability table, the weighted sampling method results in a low efficiency if the values of mass points have the same trend as that of the associated probabilities. On the other hand, this method achieves a high efficiency if the values of mass points and the associated probabilities have opposite trends.

Two Monte Carlo methods based on absorbing and ergodic Markov chains for solving a system of linear equations have been investigated. The Monte Carlo solutions incorporating the inverse method, Brown's method and the weighted sampling method have been examined. It was shown that the weighted sampling method reduces the time complexities of these methods from $\Theta(n^3)$ in the case of the inverse

method to $\Theta(n^2)$, for solving a system of n linear equations. The weighted sampling method also enhances the vectorizability of the Monte Carlo codes for solving a sparse linear system. Major restructuring of the scalar codes is required to achieve high vectorization speedups. For this purpose, a stack processing scheme (an event-based algorithm) was implemented to carry out concurrently a set of random walks. The weighted sampling method speeds up the computation of the Monte Carlo methods in scalar as well as vector processing. In general, the efficiencies of Monte Carlo solutions incorporating the weighted sampling method are higher than those incorporating the inverse and Brown's methods. Thus, the weighted sampling method is suited for the Monte Carlo methods for solving a large sparse system of linear equations.

Three neutron transport problems with different physical characteristics have been examined and solved using the MORSE code. The standard code uses the inverse method for sampling energy groups and neutron directions after collisions. Brown's method and the weighted sampling method were incorporated into the MORSE code. It was found that Brown's method requires a complicated procedure and a large amount of memory to set up the probability tables in the MORSE code. In contrast, the weighted sampling method does not require preprocessed scattering and angular scattering probability tables, since it can utilize the original probability tables. The weighted sampling method was found to speed up the scalar processing of the MORSE code. In order to predict the potential vectorization speedup, we have investigated the local vectorization speedup of the weighted sampling method incorporated into the subroutine which handles collision process. The weighted sampling method increases the vectorizability of this subroutine, and achieves high speedup of about 10. In this case, the weighted sampling method results in a large sample variance, since the statistical weights of neutrons vary significantly. Consequently, this method results in lower efficiencies than those of Brown's and the inverse methods.

In an attempt to improve the efficiency of the weighted sampling method, four variants of the weighted sampling method were developed. These variants involve

stretching of a probability table, nonuniform distributions and the combination of two sampling methods. For the mean estimation from a one-dimensional probability table, the variant which utilizes a nonuniform discrete distribution entails much higher efficiency than those of the other sampling methods. This is achieved by reducing the sample variance and vectorizing the generation of random numbers from binomial and geometric distributions. The combination of the inverse method and the weighted sampling method with a stretched probability table achieves high efficiency, if the neutron transport application involving a large number of collisions at low energy. This is typically true for many applications.

The computational study has been carried out on the IBM 3090-180 with a vector facility. The architectures and the techniques of vectorizing compilers of other vector computers are similar to those of the IBM 3090. Therefore, we expect that the implementations and qualitative results in this thesis are applicable to other vector computers.

In conclusion, we have made the first major attempt to investigate the vectorizability of some of the existing discrete sampling methods, applied them to the various problems, and assessed their performance for the scalar and vector processing. To overcome the computational drawbacks of these sampling methods, the weighted sampling method was proposed. It was shown that this method enhances the vectorizability of the vector Monte Carlo codes and achieves better performance for the scalar as well as vector processing. The study of vectorizing the generation of discrete nonuniform random numbers has been carried out as well.

7.2 Future Work

Future work may include:

- The implementation of the codes developed in this work on other univector processors. This should be a straightforward task, due to the fact that the architectures and the techniques of vectorizing compilers of other vector computers are similar to those of the IBM 3090-180VF.

- The implementation of the codes developed in this work on parallel-vector processors. The sampling methods were implemented on the IBM 3090-180VF which has only a single vector processor. The vectorization speedups attained in this work indicate that high speedups can be achieved if the codes are implemented on parallel-vector computers.

- The development of variants of the weighted sampling method using distributions other than geometric and binomial distributions. Other discrete distributions are worth investigated since a reasonable sample variance of a Monte Carlo solution can only be achieved by using a suitable distribution.

- Further study of determining the most suited variant of the weighted sampling method for a given application. The study should be directed towards utilizing a mathematical model for minimizing the sample variance of a Monte Carlo solution.

- The vectorization of the entire MORSE code. The local vectorization of the weighted sampling method in the MORSE code achieved high speedups. This fact provides an indication that vectorization of the entire MORSE code is worth considering.

- The investigation of an efficient geometry algorithm to track particles throughout the system and relate the positions of particles to the materials encountered

in the MORSE code. This process is time consuming and contains many vectorization inhibitors. Its vectorization is therefore essential, if the MORSE code is to be fully vectorized.

- Minimization of time for processing multistack on univector as well as multivector processors. If there is only one vector processor, the problem is to obtain the optimal sequence of processing the stacks since only one stack can be processed at a time. For a multivector processor with the assumption that the number of processors is less than the number of stacks, the problem is to assign a set of stacks to the processors. Optimization of these processes will be useful to reduce the computing time.

References

[1] Ahmad, M., Giri, N., and Sinha, B.K. (**1980**), *Estimation of The Mixing Proportion of Two Known Distributions*, Rapports De Recherches Du Department De Mathematiques Et De Statistique, Universite De Montreal, Quebec.

[2] Aho, A.V., Hopcroft, J.E., and Ullman, J.D. (**1974**), *The Design And Analysis of Computer Algorithms*, Addison–Wesley Publishing Company, Reading, Massachusetts.

[3] Bell, G.I. and Glasstone, S. (**1970**), *Nuclear Reactor Theory*, Van Nostrand Reinhold Company, New York.

[4] Berg, P.W. and McGregor, J.L (**1966**), *Elementary Partial Differential Equations*, Holden–Day Inc., San Francisco, California.

[5] Bhavsar, V.C. (**1981**), *Parallel Algorithms For Monte Carlo Solutions of Some Linear Operator Problems*, Ph.D. Thesis, Department of Electrical Engineering, Indian Institute of Technology, Bombay, India.

[6] Bhavsar, V.C. and Isaac, J.R. (**1987**), *Design And Analysis of Parallel Monte Carlo Algorithms*, SIAM J. Sci. Stat. Comput, Vol. 8, No. 1, pp. s73–s95.

[7] Bhavsar, V.C., Noye, G., and Hussein, E.M.A. (**July 1988**), *Performance of a Monte Carlo Code on Two Supercomputers*, Proc. of 12th IMACS World Congress on 'Scientific Computation', Paris, Vol. 4, pp. 408–410.

[8] Binder, K. and Heermann, D.W. (**1988**), *Monte Carlo Simulation in Statistical Physics: An Introduction*, Springer–Verlag, Berlin, Heidelberg.

[9] Bobrowicz, F.W., Lynch, J.E., Fisher, K.J., and Tabor, J.E. (**1984**), *Vectorized Monte Carlo Photon Transport*, Parallel Computing, Vol. 1, pp. 295–305.

[10] Brown, F.B. (**1981**), *Vectorized Monte Carlo*, Ph.D. Thesis, Department of Nuclear Engineering and the Department of Electrical and Computer Engineering, University of Michigan.

[11] Brown, F.B., Martin, W.R., and Calahan, D.A. (**1981**), *A Discrete Sampling Method for Vectorized Monte Carlo Calculations*, Trans. Am. Nucl. Soc. Vol. 38, pp. 354–355.

[12] Brown, F.B., Martin, W.R., and Calahan, D.A. (**1981**), *A Computer Program for A Discrete Sampling Method for Vectorized Monte Carlo Calculations*, Personal Communication.

[13] Brown, F.B. and Martin, W.R. (**1984**), *Monte Carlo Methods For Radiation Transport Analysis on Vector Computers*, Progress in Nuclear Energy. Vol. 14, No. 3, pp. 269–299.

[14] Brown, F.B. and Martin, W.R. (**1987**), *Status of Vectorized Monte Carlo For Particle Transport Analysis*, The International Journal of Supercomputer Applications, Vol.1,No.2,11–32.

[15] Carnevali, P. and Kindelan, M. (**1990**), *A Simplified Model to Predict the Performance of FORTRAN vector Loops on the IBM 3090/VF*, Parallel Computing, Vol. 13, pp. 35–46.

[16] Carter, L.L. and Cashwell, E.D. (**1977**), *Particle–Transport Simulation with the Monte Carlo Method*, Technical Information Center, Energy Research and Development Administration, Oak Ridge, Tennessee.

[17] Cassidy, B. (**Oct. 1986**), *UNB Installs IBM 3090 Computer And Vector Facility*, Compilation, Vol. 19, No. 3, pp. 2–3.

[18] Cheng, H. (**September 1989**), *Vector Pipelining, Chaining, and Speed on the IBM 3090 and Cray X-MP*, IEEE Computer, Vol. 22, No. 9, pp. 31–46.

[19] Cheney, W. and Kincaid, D. (**1985**), *Numerical Mathematics and Computing*, Second edition, Brooks / Cole Publishing Co., Monterey, California.

[20] Clark, R.S. and Wilson, T.L. (**1986**), *Vector System Performance of The IBM 3090*, IBM Systems Journal, Vol.25,No.1, pp. 63–82.

[21] Cooper, L. (**1974**), *Applied Nonlinear Programming For Engineers And Scientists*, Aloray Inc., Englewood, New Jersey.

[22] David, H.A. (**1981**), *Order Statistics*, Second edition, John Wiley & Sons Inc., New York.

[23] Devroye, L. (**1986**), *Non-Uniform Random Variate Generation*, Springer-Verlag, New York.

[24] Dongarra, J.J., Duff, I.S., Sorensen, D.C., and Van Der Vorst, H.A. (**1991**), *Solving Linear Systems on Vector and Shared Memory Computers*, SIAM, Philadelphia.

[25] Duff, I.S., Erisman, A.M., and Reid, J.K. (**1989**), *Direct Methods for Sparse Matrices*, Oxford University Press, New York.

[26] Edmundson, H.P. (**1952**), *Monte Carlo Matrix Inversion and Recurrent Events*, Math. Tables Other Aids Comput., Vol. 7, pp. 18–21.

[27] Forsythe, G.E. and Leibler, R.A. (**1950**), *Matrix Inversion by a Monte Carlo Method*, Math. Tables Other Aids Comput., Vol. 4, pp. 127–129.

[28] Gerald, C.F. and Wheatley, P.O. (**1989**), *Applied Numerical Analysis*, Fourth edition, Addison–Wesley Publishing Co., Reading, Massachusetts.

[29] Gibson, D.H., Rain, D.W., and Walsh, H.F. (**1986**), *Engineering and Scientific Processing on the IBM 3090*, IBM Systems Journal, Vol. 25, No. 1, pp. 36–50.

[30] Glynn, P.W. and Iglehart, D.L. (**Nov. 1989**), *Importance Sampling for Stochastic Simulations*, Management Science, Vol. 35, No. 11, pp. 1367–1392.

[31] Halton, J.H. (**1970**), *A Retrospective and Prospective Survey of The Monte Carlo Methods*, SIAM Review, Vol.12, No.1, pp. 1–63.

[32] Hammersley, J.M. and Handscomb, D.C. (**1964**), *Monte Carlo Methods*, Methuen, London.

[33] Heller, D. (**October 1978**), *A Survey of Parallel Algorithms In Numerical Linear Algebra*, SIAM Review Vol. 20, No.4, pp. 740–773.

[34] Heidelberger, P. (**1987**), *Discrete Event Simulations and Parallel Processing: Statistical Properties*, IBM Thomas J. Watson Research Center, Yorktown Heights, New York.

[35] Hillier, F.S. and Lieberman, G.J. (**1986**), *Introduction To Operations Research*, Fourth edition, Holden Day Inc., Oakland, California.

[36] Hussein, E.M.A. (**1983**), *Fast Neutron Scattering Method for Local Void Fraction Distribution Measurement In Two-Phase Flow*, Ph.D. Thesis, McMaster University.

[37] Hussein, E.M.A. (**1988**), *Modelling and Design of a Neutron Scatterometer for Void–Fraction Measurement*, Nuclear Engineering and Design, 105, pp. 333-348.

[38] Hussein, E.M.A. (**1990**), *Center-of-Mass Monte Carlo Simulation of Neutron Scattering Experiments*, Appl. Radiat. Isot., Vol. 41, No. 10/11, pp 1033–1039, Int. J. Radiat. Appl. Instrum. Part A.

[39] Hussein, E.M.A. (**1991**), *Approximate Estimators for Fluence at a Point in Monte Carlo Center-of-Mass Neutron Transport*, Nuclear Science and Engineering, Vol. 109, No. 4, pp. 416–422.

[40] Hwang, K. and Briggs, F.A. (**1984**), *Computer Architecture And Parallel Processing*, McGraw–Hill Computer Science Series, New York.

[41] IBM (**1986**), *Designing and Writing FORTRAN Programs for Vector and Parallel Processing*, IBM SC23–0337–00, Kingston, New York.

[42] IBM (**March 1990**), *Engineering and Scientific Subroutine Library, Guide and Reference Release 4*, Fifth Edition, IBM Corp., Kingston, New York.

[43] IBM (**March 1988**), *VS FORTRAN Version 2 Release 3, Language and Library Reference*, Fourth Edition, IBM Corp., Kingston, New York.

[44] IBM (**August 1989**), *VS FORTRAN Version 2 Release 4, Language and Library Reference*, Fifth Edition, IBM Corp., Kingston, New York.

[45] IMSL (**August 1989**), *Fortran Subroutines for Statistical Analysis*, Version 1.0, IMSL Inc., Houston, Texas.

[46] Ishiguro, M., Harada, H., Makino, M., and Martin, J.L. (**Fall 1987**), *Performance Analysis of Vectorized Nuclear Codes On A FACOM VP–100 At The Japan Atomic Energy Research Institute*, The International Journal of Supercomputer Applications, Vol. 1, No.3, pp. 45–56.

[47] Johnson, N.L. and Kotz, S. (**1977**), *Urn Models and Their Application*, John Wiley & Sons Inc., New York.

[48] Kahn, H. (**1956**), *Applications of Monte Carlo*, The Rand Corporation.

[49] Karmarkar, N. (**June 1991**), *A New Parallel Architecture for Sparse Matrix Computation Based on Finite Projective Geometries*, Proceedings of Supercomputing Symposium '91, V.C. Bhavsar (Ed.), Faculty of Computer Science, University of New Brunswick, Fredericton, N.B., pp. 309–321.

[50] Kemeny, J.G. and Snell, J.L. (**1960**), *Finite Markov Chains*, D. Van Nostrand Company, Princeton, New Jersey.

[51] Knuth, D.E. (**1968**), *The Art of Computer Programming: Volume 1 / Fundamental Algorithms*, Addison–Wesley Publishing Company, Reading, Massachusetts.

[52] Kogge, P.M. (**1981**), *The Architecture of Pipelined Computers*, McGraw–Hill Book Company, New York.

[53] Kronmal, R.A. and Peterson,Jr. A.V. (**1979**), *On the Alias Method for Generating Random Variables from a Discrete Distribution*, The American Statistician, Vol. 33, No. 4, pp. 214–218.

[54] Kronsjo, L. (**1987**), *Algorithms: Their Complexity and Efficiency* , Second Edition, John Wiley & Sons Inc., Chichester.

[55] Law, A.M. and Kelton, W.D. (**1982**), *Simulation Modeling and Analysis* McGraw-Hill Book Company, New York.

[56] Liu, B. and Strother, N. (**June 1988**), *Programming In VS Fortran On The IBM 3090 For Maximum Vector Performance*, IEEE Computer, Vol.21, No.6, pp. 65–76.

[57] Los Alamos Scientific Laboratory (**1963**), *Los Alamos Group-Averaged Cross Section*, LAMS-2941, Los Alamos.

[58] Lux, I. and Koblinger, L. (**1991**), *Monte Carlo Particle Transport Methods: Neutron and Photon Calculations*, CRC Press Inc., Boca Raton, Florida.

[59] Marsaglia, G. (**1963**), *Generating Discrete Random Variables in a Computer*, Comm. ACM, Vol. 6, No. 1, pp. 37–38.

[60] Martin, W.R., Rathkopf, J.A., and Nowak, P.F. (**Nov. 1985**), *Monte Carlo Photon Transport On The Cray-XMP*, Trans. American Nuclear Society.

[61] Martin, W.R., Nowak, P.F., and Rathkopf, J.A. (**1986**), *Monte Carlo Photon Transport On A Vector Supercomputer*, IBM Journal of Research and Development, Vol.30, No.2, 193–201.

[62] Martin, W.R., Wan, T.C., Abdel Rahman, T.S., and Mudge, T.N. (**1987**), *Monte Carlo Photon Transport on Shared Memory and Distributed Memory Parallel Processors*, The International Journal of Supercomputer Applications, Vol. 1, No. 3, pp. 57–74.

[63] Mathur, A.P. and Galiano, E. (**1987**), *Inducing Vectorization: A Formal Analysis*, Technical Report, SERC-TR-6-P, Software Engineering Research Center, Purdue University, West Lafayette, Indiana 47907.

[64] Mathur, A.P., Galiano, E., Ligon III, W., and Greenlaw, T. (**1987**), *Concurrent Execution over Multiple Data Sets on Vector Processors*, Technical Report, SERC-TR-7-P, Software Engineering Research Center, Purdue University, West Lafayette, Indiana 47907.

[65] MORSE-CG (**1971**), *General Purpose Monte Carlo Multigroup Transport Code with Combinatorial Geometry*, CCC-203, RSIC, Oak Ridge National Laboratory.

[66] Nakagawa, M., Mori, T., and Sasaki, M. (**1991**), *Comparison of Vectorization Methods Used in a Monte Carlo Code*, Nuclear Science and Engineering, Vol. 107, No. 1, pp. 58–66.

[67] Neuts, M.F. (**1981**), *Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach*, Johns Hopkins Univ. Press, Baltimore, MD.

[68] Niederreiter, H. (**November 1978**), *Quasi-Monte Carlo Methods And Pseudo-Random Numbers*, Bulletin of The American Mathematical Society, Vol. 84, No. 6, pp. 957–1041.

[69] Niederreiter, H. (**1988**), *Low-Discrepancy and Low-Dispersion Sequences*, Journal of Number Theory, Vol. 30, pp. 51–70.

[70] Noye, G. (**1987**), *Optimizing Monte Carlo Codes on IBM 3090–180VF*, Work Report, School of Computer Science, University of New Brunswick.

[71] Papoulis, A. (**1984**), *Probability, Random Variables, and Stochastic Processes*, Second Edition, McGraw–Hill Book Company, New York.

[72] Radicati, G. and Vitaletti, M. (**March 1989**), *Conjugate-gradient Subroutines for the IBM 3090 Vector Facility*, IBM Journal of Research and Development, Vol. 33, No. 2, pp. 125–135.

[73] Rego, V.J. and Mathur, A.P. (**March 1990**), *Concurrency Enhancement through Program Unification: A Performance Analysis*, Journal of Parallel and Distributed Computing, Vol. 8, No. 3, pp. 201–217.

[74] Rego, V.J. (**Dec. 1989**), *Some Efficient Computational Algorithms Related to Phase Models*, Purdue CSD-TR727, Acta Informatica.

[75] Rego, V.J. and Mathur, A.P. (**October 1990**), *Exploiting Parallelism Across Program Execution: A Unification Technique and Its Analysis*, IEEE Transactions On Parallel and Distributed System, Vol. 1, No. 4, pp. 399–414.

[76] Rubinstein, R.Y. (**1981**), *Simulation and The Monte Carlo Method*, John Wiley & Sons Inc., New York.

[77] Sadeh, E. (**April 1974**), *Monte Carlo Solution of Partial Differential Equations by Special Purpose Digital Computer*, IEEE Trans. Computers, Vol. c-23, No.4, pp. 389–397.

[78] Sarno, R. (**Sept. 1988**), *Performance Modelling of Vectorized Monte Carlo Codes*, M.Sc. Thesis, Technical Report TR88-043, Faculty of Computer Science, University of New Brunswick, Fredericton, N.B., Canada.

[79] Sarno, R., Bhavsar, V.C., and Banerjee, P.K. (**August 1989**), *Design and Analysis of Vector Processing in Monte Carlo Particle Transport Codes*, Proceedings of the 1989 International Conference on Parallel Processing, The Pennsylvania State University Press, University Park, pp. III.80–III.87.

[80] Sarno, R., Bhavsar, V.C., and Banerjee, P.K. (**Sept. 1989**), *Design and Analysis of Vectorized Monte Carlo Codes*, Technical Report TR89-048, Faculty of Computer Science, University of New Brunswick, Fredericton, N.B., Canada, 48 pages.

[81] Sarno, R., Bhavsar, V.C., and Hussein, E.M.A (**Nov. 1989**), *Performance of Discrete Random Variable Generators on IBM 3090-180VF*, Proceedings of APICS Annual Computer Science Conference, V.C. Bhavsar (Ed.), Faculty of Computer Science, University of New Brunswick, Fredericton, N.B., Canada, pp. 114–125.

[82] Sarno, R., Bhavsar, V.C., and Hussein, E.M.A (**December 1990**), *Generation of Discrete Random Variables on Vector Computers For Monte Carlo Simulations*, International Journal of High Speed Computing, Vol. 2, No. 4, pp. 335–350.

[83] Sarno, R., Bhavsar, V.C., and Hussein,E.M.A (**June 1991**), *Monte Carlo Solutions of Linear Equations Using Weighted Sampling*, Proceedings of Supercomputing Symposium '91, V.C. Bhavsar (Ed.), Faculty of Computer Science, University of New Brunswick, Fredericton, N.B., Canada, pp. 151–163.

[84] Schaeffer, N.M. (**1973**), *Reactor Shielding for Nuclear Engineers*, U.S. Atomic Energy Commission, Springfield, Virginia.

[85] Sedgewick, R. (**1990**), *Algorithms in C*, Addison-Wesley Publishing Company, Reading, Massachusetts.

[86] Singh, Y., King, G.M., and Anderson, J.W. (**1986**), *IBM 3090 Performance: A Balanced System Approach*, IBM Systems Journal, Vol. 25, No. 1, pp. 20–35.

[87] Spanier, J. and Gelbard, E.M. (**1969**), *Monte Carlo Principles and Neutron Transport Problems*, Addison–Wesley Publishing Company.

[88] Stinson, D.R. (**1987**), *An Introduction to The Design and Analysis of Algorithms*, Second edition (Revised), The Charles Babbage Research Centre, Winnipeg, Canada.

[89] Tassou, T.A. (**1986**), *Adaptation of A Monte Carlo Radiation Transport Code To Supercomputers*, M.Sc. Thesis, Faculty of Computer Science, University of New Brunswick, Fredericton.

[90] Tassou, T., Bhavsar, V.C., Hussein, E., Gallivan, K.A. (**1986**), *Monte Carlo Neutron Transport on the Alliant FX/8*, School of Computer Science, University of New Brunswick, Fredericton, NB, TR86–035.

[91] Tassou, T.A., Hussein, E.M.A., Bhavsar, V.C., and Gujar, U.G. (**1986**), *Monte Carlo Neutron Transport on the Cyber-205 Supercomputer*, Proceedings of Second International Conference on Simulation Methods in Nuclear Engineering, Brais, A. (Ed.), Groupe d'analyse Nucleaire Ecole Polytechnique, Montreal, Canada, Vol. 2, pp.619–637.

[92] Thinking Machines Corporation (**Oct. 1991**), *The Connection Machine CM-5 Technical Summary*, Cambridge, Massachusetts.

[93] Topham, N.P., Omondi, A., and Ibbett, R.N. (**1988**), *On the Design and Performance of Conventional Pipelined Architectures*, The Journal of Supercomputing, No. 1, pp. 353–393.

[94] Trivedi, K.S. (**1982**), *Probability & Statistics With Reliability, Queuing, And Computer Science Applications*, Prentice–Hall Inc., Englewood Cliffs, New Jersey.

[95] Troubetzkoy, E., Steinberg, H., and Kalos, M. (**1973**), *Monte Carlo Radiation Penetration Calculations on a Parallel Computer*, Trans. Am. Nucl. Soc., Vol. 17, p.260,1973.

[96] Tucker, S.G. (**1986**), *The IBM 3090 System: An Overview*, IBM System Journal, Vol. 25, No. 1, pp. 4–19.

[97] Varga, R.S. (**1962**), *Matrix Iterative Analysis*, Prentice-Hall, Englewood, Cliffs, N.J.

[98] Walker, A.J. (**1977**), *An Efficient Method for Generating Discrete Random Variables with General Distributions*, ACM Trans. on Mathematical Software, Vol. 3, No. 3, pp. 253–256.

[99] Wang, A.K.F. (**Sept. 1989**), *Some Models for Vectorized Algorithms*, M.Sc. Thesis, Faculty of Computer Science, University of New Brunswick, Fredericton, N.B., Canada.

[100] Wasow, W.R. (**1952**), *A Note on the Inversion of Matrices by Random Walks*, Math. Tables Other Aids Comput., Vol. 6, pp. 78–81.

[101] Weberpals, H. (**1990**), *Architectural Approach to the IBM 3090E Vector Performance*, Parallel Computing, Vol. 13, pp. 47–59.

[102] Wilks, S.S. (**1963**), *Mathematical Statistics*, Second printing, John Wiley & Sons Inc., New York.

[103] Woo, V.M.C. (**May 1989**), *Monte Carlo Codes and Vectorization Techniques*, M.Sc. Thesis, School of Computer Science, University of New Brunswick, Fredericton, N.B., Canada.

[104] Ya, P.V. (**1980**), *New Fast Algorithms For Matrix Operation*, SIAM J. Computing, Vol. 9, pp. 321–342.

[105] Zee, S.K., Turinsky, P.J., and Shayer, Z. (**1989**), *Vectorized and Multitasked Solution of the Few-Group Neutron Diffusion Equations*, Nuclear Science and Engineering, Vol. 101, No. 1, pp. 58–66.

Appendix I

The IBM 3090-180VF

Some of the features of the IBM 3090-180 with a Vector Facility available at the University of New Brunswick are as follows: [17]

- 64 MB of central storage,
- 16 channels,
- 18.5 nanosecond cycle time,
- a 64-KB high speed buffer,
- 64-bit wide data paths.

The Vector Facility has the following features:

- 32-bit binary operands (INTEGER*4 or LOGICAL*4),
- 32-bit floating-point operands (REAL*4),
- 64-bit floating point operands (REAL*8),
- 16 vector registers, each containing 128 32-bit operands,
- the couple of vector registers to process 64-bit operands,

- masked vector addressing,
- 171 new instructions for vector processing,
- pipelined add, subtract, compare and multiply arithmetic units.

VITA

Candidate's full name	: Riyanarto Sarno
Place and date of birth	: Surabaya, Indonesia, August 3, 1959.
Permanent address	: Jl. Tales 4 No.14, Surabaya (60244), Indonesia.
Schools attended	: Surabaya Senior High School II, Surabaya, Indonesia, 1975 – 1977.
Universities attended	: Bandung Institute of Technology, Bandung, Indonesia, Ir.(EE), 1978 – 1983.
	University of Padjadjaran, Bandung, Indonesia, Drs.(Economics), 1980 – 1985.
	University of New Brunswick, Fredericton, Canada, MSc.(CS), 1987 – 1988.

Publications:

1. Sarno, R. (**Oct. 1983**), *System Softwares for Mini Computer WANG 2200 VP*, Ir. Thesis, Department of Electrical Eng., Bandung Institute of Technology, Bandung, Indonesia.

2. Sarno, R. (**Nov. 1985**), *Forecasting of Indonesian Economic Variables Using Dynamic Input–Output Models*, Drs. Thesis, Faculty of Economics, University of Padjadjaran, Bandung, Indonesia.

3. Sarno, R. (**Sept. 1988**), *Performance Modelling of Vectorized Monte Carlo Codes*, M.Sc. Thesis, Technical Report TR88-043, Faculty of Computer Science, University of New Brunswick, Fredericton, N.B., Canada.

4. Sarno, R., Bhavsar, V.C., and Banerjee, P.K. (**August 1989**), *Design and Analysis of Vector Processing in Monte Carlo Particle Transport Codes*, Proceedings of the 1989 International Conference on Parallel Processing, The Pennsylvania State University Press, University Park, pp. III.80–III.87.

5. Sarno, R., Bhavsar, V.C., and Banerjee, P.K. (**Sept. 1989**), *Design and Analysis of Vectorized Monte Carlo Codes*, Technical Report TR89-048, Faculty of Computer Science, University of New Brunswick, Fredericton, N.B., Canada, 48 pages.

6. Sarno, R., Bhavsar, V.C., and Hussein, E.M.A. (**Nov. 1989**), *Performance of Discrete Random Variable Generators on IBM 3090-180VF*, Proceedings of APICS Annual Computer Science Conference, V.C. Bhavsar (Ed.), Faculty of Computer Science, University of New Brunswick, Fredericton, N.B., Canada, pp. 114–125.

7. Sarno, R., Bhavsar, V.C., and Hussein, E.M.A. (**Dec. 1990**), *Generation of Discrete Random Variables on Vector Computers For Monte Carlo Simulations*,

International Journal Of High Speed Computing, Vol. 2, No. 4, pp. 335–350.

8. Sarno, R., Bhavsar, V.C., and Hussein, E.M.A. (**June 1991**), *Monte Carlo Solutions of Linear Equations Using Weighted Sampling*, Proceedings of Supercomputing Symposium '91, V.C. Bhavsar (Ed.), Faculty of Computer Science, University of New Brunswick, Fredericton, N.B., Canada, pp. 151–163.