# DEDICATIONS

*To my dear parents, and sisters.*

*To my advisor Dr. Mahmoud Alrefaei.*

*To my friends in and out the university.*

# TABLE OF CONTENTS

# List of Tables

# List of Figures

# ABSTRACT


**INTEGRATION USING MONTE CARLO METHODS**

**By: Hossam Abdelrahman**

**Supervisor: Dr. Mahmoud Alrefaei**

We consider the problem of estimating the value of a $d$-dimensional integral ($d \geq 1$) over a multidimensional hyper-rectangular region, using the Monte Carlo methods. One can define the Monte Carlo methods to be a technique that uses pseudo-random numbers for estimating solution of a mathematical or physical problem. In the past, several adaptive techniques have been applied to the Monte Carlo methods. We present a deep description of an iterative adaptive Monte Carlo integration algorithm; and apply it on a set of multidimensional integrals. The adaptive Monte Carlo algorithm may move to a worse estimated integral in few iterations. We propose a corrector adaptive Monte Carlo integration algorithm that prevents the adaptive algorithm from moving to a worse estimate. Moreover, we use the Mersenne Twister generator of pseudo-random numbers, which is believed to be one of the best random number generators known today. We implement the corrector algorithm for estimating values of a set of numerical multidimensional integrals. The numerical results show that this algorithm performs better than the usual adaptive algorithm.

# Chapter One

# Introduction

In this thesis, we consider the $d$-dimensional integral

$$I = \int_{\Gamma} f(\mathbf{x}) d\mathbf{x} \tag{1.0.1}$$

where $\mathbf{x} \in \Re^d$, $d \geq 1$, $\Gamma$ denotes the $d$-dimensional hyper-rectangular $[a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_d, b_d] \subset \Re^d$ and $f(\mathbf{x}) : \Gamma \longrightarrow \Re$ is square integrable on $\Gamma$. We study the use of Monte Carlo methods for estimating the value of this integral. A *Monte Carlo* (MC) method is a procedure that involves the use of statistical sampling procedures to estimate the solution of mathematical or physical problems, this method solves non-probabilistic problems using probabilistic methods. MC methods are not known for all problems, and on the other hand, one problem can have more than one MC procedure, see Hammersley and Handscomb [1] and Fishman [2]. They are now one of the most powerful and commonly used techniques for analyzing complex problems. It has been showed that MC methods are very efficient in solving multidimensional integrals in composite domains.

MC integration is the general estimation of the value of a definite integral by sampling the function to be integrated at random points in the domain of integration. The problem of evaluating integrals of high dimensions is very important since it appears in many applications of control theory, statistical physics and mathematical economics, see Miguel [3] and Erich [4]. For example, in physics: If there is a system of $n$ particles with pair potential energy $\phi(r_{i,j})$ then the classical partition function $Z = \int \int \ldots \int \int \exp\left\{\sum_{pairs} \phi(r_{i,j})/kT\right\} d^3r_1 \ldots d^3r_n$ involves integration in $3n$ dimensions, see Woolfson and Pert [5]. Moreover, in control theory, one of the numerical

approaches for solving stochastic systems leads to a large number of multidimensional integrals with dimensionality up to $d = 30$, see Karaivanova and Dimov [6]. It would be extremely expensive to evaluate such problems by conventional quadrature techniques. For instance, one needs several seconds for evaluating 30-$d$ integrals using the MC method. However, the evaluation of the same integral using Gaussian quadrature will need more than $10^6$ billion years if CRAY Y-MP C92A (a super computer) is used, see Karaivanova and Dimov [6]. MC methods are very good approaches in these cases and can be applied for estimating integrals with up to few hundred dimensions when parallel computers are used. Moreover, the MC integration can be implemented for queuing networks which appear in some manufacturing systems, high speed communication networks and in computer systems, see Ross and Wang [7].

The idea of MC calculation is a lot older than the computer, and its previous name was "Statistical Sampling". The first use of MC idea (Statistical Sampling) was in the second half of the 19th century by estimating the value of $\pi$. In (1899) Lord Rayleigh showed that a one dimensional random walk without absorbing barriers could provide an approximate solution to a parabolic differential equation. In (1933) Petrowsky showed the asymptotic connection between a random walk whose sequence of locations form a Markov chain and the solution to an elliptic partial differential equation. For more historical notes, see Fishman [2]. The actual use of MC methods as a research tool is due to the work on the atomic bomb during the second world war, where the term "Monte Carlo" was introduced by Von Neumann and Ulam as a codeword for the secret work at Los Alamos. It was suggested by the gambling casinos at the city of Monte Carlo in Monaco, see Kahaner et al. [8]. Shortly thereafter, MC methods have been used to evaluate complex multidimensional integrals, to solve certain integral equations, and partial differential equations that were not amenable to analytic solution.

Several refinements of the MC integration methods have been proposed. In order to reduce the probable error of integration, the variance reduction techniques were applied to the MC methods such as, the *Importance Sampling* technique. The Impor-

tance Sampling technique concentrates the sampled points on the important parts of the domain of integration. Other variance reduction technique include the *Stratified Sampling* technique that is based on dividing the domain of integration into subregions, then the MC rule is applied in each of these subregions, see Rubinstein [9] and Lux and Koblinger [10]. In order to generate more uniform samplings, the *quasi-random numbers* were used instead of the *pseudo-random numbers* (PRN) in the MC methods, introducing the *Quasi Monte Carlo* methods. The quasi-random numbers are generated using infinite sequences. There are many quasi-random sequences that are considered to have greater uniformity than the PRNs, and the most common sequences are the Halton, Sobol and Faure sequences, see Marokoff and Caflisch [11]. Many adaptive algorithms have used the MC rule as their building block. The adaptive MC integration algorithms are (most of the time) *iterative* algorithms aim to reach a specific small error estimate of a definite integral, see Dahl [12], Karaivanova and Dimov [6] and Schürer [13, 14]. A difficulty in the MC integration occurs when integrating a function that has sharp peak in very small regions. A naive approach to overcome this problem can be done by increasing the number of sampling points until finding such regions, or by finding an effective dividing rule of the domain of integral (the stratified sampling). An effective direction to overcome this problem, is by using the *superposing method* that concentrates the sampled points in these hard subregions, see Kaneko and Tobimatsu [15].

In this thesis, we consider the adaptive algorithm that uses iteratively the basic idea of the Stratified Sampling, see Dahl [12] and Schürer [13, 14]. A deep discussion of the adaptive algorithm will be given, and a detailed algorithm based on arbitrary number of subdivisions will be proposed. The proposed algorithm is tested numerically on a set of multidimensional integrals. It is noted that the adaptive algorithm may move to a worse estimated integral in few iterations. We propose a corrector adaptive MC integration algorithm that is guaranteed to move to a better estimate in each iteration. Therefore, less iterations and less time is needed to reach a specific error estimate, and a stable decreasing graph of the estimated error is observed.

The rest of the thesis is organized as follows; in Chapter Two, we give a general description of the MC integration method, in which we review two MC integration methods in one dimensional case, and discuss the "basic MC integration". We discuss the generalization into the multidimensional case, and talk about the convergence rate of the MC integration. We also give a brief comparison between the basic MC and some standard numerical integration techniques. Then, we describe two common variance reduction techniques applied to the MC procedure. In Chapter Three, we present an adaptive MC algorithm, and give a detailed adaptive MC algorithm. Then apply the detailed adaptive MC algorithm on a set of multidimensional integrals. In Chapter Four, we propose a corrector adaptive MC algorithm that prevents the usual adaptive algorithm from moving to a worse estimate in each iteration. It also contains numerical results obtained by applying the corrector algorithm on a set of multidimensional integrals. Chapter Five contains the conclusion.

The MC procedure is based on generating random variables, which can be done using uniform random numbers. In Appendix A, we review two PRN generators; the *Linear Congruential* generator, which is widely used in generating PRNs; and the *Mersenne Twister* (MT19937) generator. MT19937 is believed to be one of the best random number generators known today, and the most suitable generator of PRNs for the MC simulation, see Matsumoto and Nishimura [16], several recent researches have used the MT19937 for generating PRNs in MC integration, see Dahl [12] and Schürer [13, 14]. Throughout this thesis, we also use the MT19937 for generating PRNs. For more information about MT19937, see the Mersenne Twister home page [21]. In the rest of the Appendix, we review some basic methods for generating random variables in a single variate and in a multivariate case.

# Chapter Two

# General Description of Monte Carlo Integration

In this chapter, we describe how to use MC procedures for estimating the value of the definite integral given in equation (1.0.1). The MC integration is the general estimation of the value of a definite integral by sampling the integrand at random points in the domain of integration. In order to understand the basic idea of the MC integration, we consider a one dimensional integral. We review two MC procedures for estimating this integral; the first MC procedure is "the hit or miss" or the "acceptance rejection" MC method. This procedure is based on the geometric interpretation of an integral as an area. The second procedure is "the sample mean" MC method, and it is based on the representation of an integral as a mean value. We show that, both of these MC procedures are *consistent* estimators of the integral. The consistent estimator is defined as follows, see Lehmann and Casella [17].

**Definition 2.0.1.** (**Consistent Estimator**) The static $\hat{\theta}_n$ is a *consistent estimator* of the parameter $\theta$ if for each $\varepsilon > 0$

$$\lim_{n\to\infty} P(|\hat{\theta}_n - \theta| < \varepsilon) = 1.$$

Then, we describe the generalization of the problem to $d$-dimensional case, and we see that the MC methods converge to the actual solution with rate $N^{-1/2}$, where $N$ is the total number of generated points in the domain of integration. The *rate of convergence* is defined as follows.

**Definition 2.0.2.** (**Rate of Convergence**) A family of integration routines $\tilde{I}_n$ is said to have a *rate of convergence* of $g(n)$, if

$$|\tilde{I}_n - I| \leq O(g(n))$$

We make a brief comparison between the MC integration and some standard numerical integration techniques. Finally, in order to reduce the probable error of MC integration, we review two important variance reduction techniques that can be applied to the MC integration procedure, namely; the "Importance Sampling" and the "Stratified Sampling" techniques. We use the notation $x \sim U(a, b)$ to denote that $x$ is uniformly distributed in the interval $[a, b]$.

## 2.1 The Hit or Miss MC method

The *Hit or Miss* or the *Acceptance Rejection* MC method is the easiest technique to explain and understand the MC method. Consider the following one dimensional integral

$$I = \int_a^b f(x)dx \tag{2.1.1}$$

where $a, b \in \Re$, $0 \leq f(x) \leq M$ , $\forall\ x \in [a, b]$, with $f(x) : \Re \to \Re$ and $M \in \Re^+$. So, we are interested in finding the area under the curve of $f(x)$. We can enclose the function $f$ with a rectangle $A$ whose area is $(b - a)M$, see Figure 2.1.

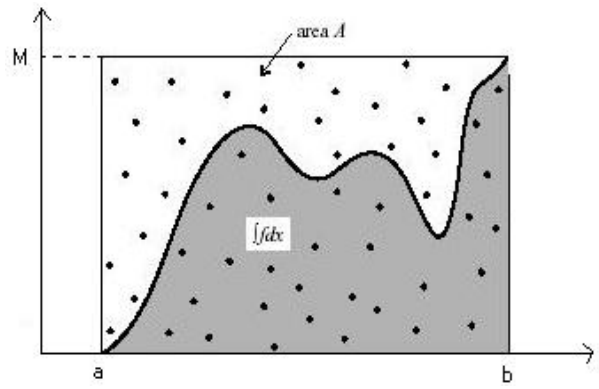


Figure 2.1: The Hit or Miss MC method

Begin by taking random points within the rectangle $A$, if a point falls under the curve of $f$ then accept the point (hit), and if it falls above the curve then reject the point (miss).

Theorem (2.1.2) below shows that a consistent estimator of the integral $I$ is the ratio of the number of accepted points to the total number of random points taken within the rectangle $A$.

To prove Theorem (2.1.2), we need the following theorem given by Lehmann and Casella [17].

**Theorem 2.1.1.** *If $\hat{\theta}_n$ is an unbiased estimator of the parameter $\theta$ and $var(\hat{\theta}_n) \to 0$ as $n \to \infty$, then $\hat{\theta}_n$ is a consistent estimator of $\theta$.*

**Theorem 2.1.2.** *Consider the one dimensional integral (2.1.1) where $0 \leq f(x) \leq M$, $\forall\ a \leq x \leq b$. Then if $(x_i, y_i), i = 1, \ldots, N$ are random points uniformly distributed over the region $A$, where $A = \{(x, y),\ a \leq x \leq b,\ 0 \leq y \leq M\}$ then*

$$\hat{I} = M(b-a)\frac{n}{N}$$

*is a consistent estimator of $I$, where $n$ is the number of cases where the condition $y_i \leq f(x_i)$ is fulfilled*

*Proof.* Define $S = \{(x, y), y \leq f(x)\}$, then the area under $f(x) = area(S) = \int_a^b f(x)dx$. Let $(x, y)$ be a random vector uniformly distributed over the rectangle $A$, the probability $p$ that it falls within $S$ (hit) is given by

$$\begin{aligned} p &= \frac{Area(S)}{Area(A)} \\ &= \frac{\int_a^b f(x)dx}{M(b-a)} \\ &= \frac{I}{M(b-a)} \Rightarrow I = M(b-a)p \end{aligned}$$

Now,

$$\begin{aligned} E(\hat{I}) &= M(b-a)E(\frac{n}{N}) \\ &= \frac{M}{N}(b-a)E(n) \end{aligned}$$

Since each of the $N$ trials constitutes a Bernoulli trial with probability $p$ of a hit, then, $n$ is a binomial random variable $B(N,p)$ with $E(n) = Np$ and $var(n) = Np(1-p)$.

$$\begin{aligned} E(\hat{I}) &= \frac{M}{N}(b-a)pN \\ &= M(b-a)p = I \end{aligned}$$

So, $\hat{I}$ is an unbiased estimator of $I$, see Lux and Koblinger [10].

Now,

$$\begin{aligned} var(\frac{n}{N}) &= \frac{1}{N^2}var(n) \\ &= \frac{1}{N}p(1-p) \end{aligned}$$

So,

$$\begin{aligned} var(\hat{I}) &= (M(b-a))^2\frac{1}{N}p(1-p) \\ &= \frac{1}{N}[(M(b-a))^2p - (M(b-a))^2p^2] \\ &= \frac{1}{N}[M(b-a)I - I^2] \\ &= \frac{I}{N}[M(b-a) - I] \end{aligned}$$

Therefore

$$\lim_{N\to\infty} var(\hat{I}) = 0$$

Since $\hat{I}$ is an unbiased estimator of $I$ and $\lim_{N\to\infty} var(\hat{I}) = 0$, then by Theorem (2.1.1), $\hat{I}$ is a consistent estimator of $I$. □

Note that, Theorem (2.1.2) is concerned only with nonnegative functions. In cases where the function $f(x)$ in equation (2.1.1) assumes also negative values, the function $f(x)$ can be rewritten as follows.

$$f(x) = f^+(x) - f^-(x)$$

where

$$f^+(x) = \max\{f(x), 0\} \quad \text{and} \quad f^-(x) = \max\{0, -f(x)\}$$

So, Theorem (2.1.2) applies to both $f^+(x)$ and $f^-(x)$ and the difference of the two estimates determines the estimate of the integral of $f(x)$. A similar trick can also be

applied in all the integration methods below.

The following algorithm describes the hit or miss MC procedure for estimating the integral given in equation (2.1.1), see Rubinstein [9].

**Hit or Miss MC algorithm**

**Step 0:** Generate $U_i \sim U(0,1)$, $i = 1, 2, \ldots, 2N$.

**Step 1:** Arrange the random numbers into $N$ pairs $(U_1, \acute{U}_1), (U_2, \acute{U}_2), \ldots, (U_N, \acute{U}_N)$ in any fashion such that each random number $U_i$ is used exactly once.

**Step 2:** Compute $X_i = a + U_i(b-a)$, see Appendix (A.2.1), $Y_i = M\acute{U}_i$ and $f(X_i)$, $i = 1, \ldots, N$.

**Step 3:** Count the number of cases $n$ for which $f(X_i) > Y_i$.

**Step 4:** Estimate the integral $I$ by

$$\hat{I} = M(b-a)\frac{n}{N}.$$

## 2.2 The sample-mean MC method

The sample-mean MC method is another way for estimating the integral given in equation (2.1.1), and it is based on representing the integral as an expected value of some random variable, see Lux and Koblinger [10] and Rubinstein [9]. We can rewrite equation (2.1.1) as

$$I = \int_a^b h(x)\phi(x)dx \tag{2.2.1}$$

where $\phi(x) \geq 0, \int_a^b \phi(x)dx = 1$, (i.e., $\phi(x)$ is a *probability density function* (p.d.f.)), and $h(x) = \frac{f(x)}{\phi(x)}$.

**Theorem 2.2.1.** *Consider equation (2.2.1). If $N$ independent $x_i$ samples are selected from $\phi(x)$ then the average of the $h(x_i)$ weights*

$$\hat{I} = \frac{1}{N}\sum_{i=1}^{N} h(x_i)$$

*is a consistent estimator of $I$.*

*Proof.* Since $x_i$'s are i.i.d. from $\phi(x)$ then

$$\begin{aligned} E(\hat{I}) &= E\left(\frac{1}{N}\sum_{i=1}^{N} h(x_i)\right) \\ &= \frac{1}{N}\sum_{i=1}^{N} E(h(x_i)) \\ &= \frac{1}{N}\sum_{i=1}^{N} \int_a^b h(x)\phi(x)dx \\ &= \int_a^b h(x)\phi(x)dx \\ &= I \end{aligned}$$

So, $\hat{I}$ is an unbiased estimator of $I$, see Lux and Koblinger [10].

Now, the variance of $\theta$ is $E(\theta^2) - E^2(\theta)$, so that

$$\begin{aligned} var(\hat{I}) &= var\left[\frac{1}{N}\sum_{i=1}^{N} h(x_i)\right] \\ &= \frac{1}{N^2} var\left[\sum_{i=1}^{N} h(x_i)\right] \\ &= \frac{1}{N^2}\sum_{i=1}^{N} var(h(x_i)) \\ &= \frac{1}{N^2}\sum_{i=1}^{N} \left(E(h^2(x_i)) - E^2(h(x_i))\right) \\ &= \frac{1}{N^2}\left[N\int_a^b h^2(x)\phi(x)dx - NI^2\right] \\ &= \frac{1}{N}\left[\int_a^b h^2(x)\phi(x)dx - I^2\right] \end{aligned}$$

see Rubinstein [9]. Therefore

$$\lim_{N\to\infty} var(\hat{I}) = 0$$

Since $\hat{I}$ is an unbiased estimator of $I$ and $\lim_{N\to\infty} var(\hat{I}) = 0$, then from Theorem (2.1.1), $\hat{I}$ is a consistent estimator of $I$. □

**Corollary 2.2.2. (The basic MC integration)** *Consider the one dimensional integral (2.1.1). If $x_1, x_2, \ldots, x_N$ are i.i.d. uniformly distributed on $[a, b]$, then*

$$I = \int_a^b f(x)dx \simeq (b-a) < f > \pm (b-a)\frac{\hat{\sigma}_f}{\sqrt{N}}$$

*where*

$$< f >= \frac{1}{N}\sum_{i=1}^{N} f(x_i), \quad \hat{\sigma}_f^2 =< f^2 > - < f >^2$$

*Proof.* From Theorem (2.2.1), choose $\phi(x) = \frac{1}{b-a}$ (i.e., the p.d.f. of the uniform distribution on $[a, b]$) and $h(x) = (b-a)f(x)$, then an estimate of $I$ is

$$\begin{aligned} \hat{I} &= \tfrac{(b-a)}{N}\textstyle\sum_{i=1}^{N} f(x_i) \\ &= (b-a) < f > \end{aligned} \tag{2.2.2}$$

And the error

$$\begin{aligned} var(\hat{I}) &= var\left(\frac{(b-a)}{N}\sum_{i=1}^{N} f(x_i)\right) \\ &= \frac{(b-a)^2}{N^2}\sum_{i=1}^{N} var(f(x_i)) \\ &= \frac{(b-a)^2}{N^2}\sum_{i=1}^{N} \sigma_f^2 \\ &= \frac{(b-a)^2}{N}\sigma_f^2 \end{aligned}$$

We have

$$\int_a^b f(x)dx = \frac{b-a}{N}\sum_{i=1}^{N} f(x_i) \pm \frac{(b-a)\sigma_f}{\sqrt{N}}, \tag{2.2.3}$$

where

$$\begin{aligned} \sigma_f^2 &= E(f^2) - E^2(f) \\ &= \frac{1}{(b-a)}\int_a^b f^2(x)dx - \left(\frac{1}{(b-a)}\int_a^b f(x)dx\right)^2 \\ &\simeq \frac{1}{N}\sum_{i=1}^{N} f^2(x_i) - \left(\frac{1}{N}\sum_{i=1}^{N} f(x_i)\right)^2 \\ &=< f^2 > - < f >^2 \\ &= \hat{\sigma}_f^2 \end{aligned}$$

By substituting the value of $\hat{\sigma}_f$ in equation (2.2.3), we get

$$\int_a^b f(x)dx \simeq (b-a) < f > \pm (b-a)\frac{\hat{\sigma}_f}{\sqrt{N}}$$

The "plus-or-minus" term is a one standard error bound estimate. □

The following algorithm describes a sample-mean MC method for estimating the integral given in equation (2.1.1), see Rubinstein [9].

**Sample-mean MC algorithm (The basic MC integration)**

**Step 0:** Generate $U_i \sim U(0,1)$, $i = 1, 2, \ldots, N$.

**Step 1:** Compute $X_i = a + U_i(b-a)$, $i = 1, \ldots, N$.

**Step 2:** Compute $f(X_i)$, $i = 1, \ldots, N$.

**Step 3:** Compute the sample mean $\hat{I}$ according to equation (2.2.2).

*Remark* 2.2.1. For a reliable error estimate, the function $f$ must be square integrable. If the function $f$ is integrable (not square integrable), then the MC estimate $\hat{I}$ will still converge to the true value, but the error estimate becomes unreliable.

**Example 2.1.** Suppose that we have the following one dimensional integral

$$I = \int_0^1 f(x)dx$$

where

$$f(x) = \begin{cases} \frac{1}{\sqrt{x}} & \text{if } \ 0 < x \leq 1 \\ 0 & \text{if } \ x = 0 \end{cases}$$

It is clear that $f(x)$ is integrable on $(0, 1]$, but not square integrable (i.e., $\int_0^1 f^2(x)dx$ does not exist). It is known that the exact value of $I$ is 2. We performed the MC integration of $f(x)$ and we got the results in Table 2.1.

| N | Basic MC | Absolute error |
|---|---|---|
| 1,000 | 2.056815 | 0.056815 |
| 10,000 | 2.050371 | 0.050371 |
| 100,000 | 1.990308 | 0.009692 |
| 1000,000 | 1.996605 | 0.003395 |

Table 2.1: The basic MC integration output

where the *absolute error* is the absolute value of the difference between the exact and the estimated value (using the MC method) of the integral.

From Table 2.1, we can get an acceptable error for our MC estimate of integration, but we can not find the estimated error because $f(x)$ is not square integrable. We also can see the slow convergence to the true value (this is because of the convergence rate of the MC method, $O(\frac{1}{\sqrt{N}})$ as we will see in the next section, see Remark 2.3.1).

Note that, the basic MC integration is not competitive with the classical numerical integration methods in the one dimensional case.

## 2.3 Generalization to multidimensional cases

In this section, we generalize the sample-mean MC method from one dimension into $d$-dimensions. We talk about the convergence rate of the basic MC and give a 10-dimensional integral example.

Consider the multidimensional integral (1.0.1). We can rewrite the integral as

$$I = \int_{\Gamma} h(\mathbf{x})\phi(\mathbf{x})d\mathbf{x} \tag{2.3.1}$$

where

$$\phi(\mathbf{x}) \geq 0, \quad \int_{\Gamma} \phi(\mathbf{x})d\mathbf{x} = 1$$

The proof of the following theorem is identical to the proof of Theorem (2.2.1).

**Theorem 2.3.1.** *If $N$ independent samples $\boldsymbol{x}_i$ are selected from $\Gamma$, according to the multidimensional p.d.f. $\phi(\boldsymbol{x})$ then*

$$\hat{I} = \frac{1}{N}\sum_{i=1}^{N} h(\boldsymbol{x}_i)$$

*is a consistent estimator of the integral $I$ in equation (1.0.1), where*

$$var(\hat{I}) = \frac{1}{N}\left[\int_{\Gamma} h^2(\boldsymbol{x})\phi(\boldsymbol{x})d\boldsymbol{x} - I^2\right] \tag{2.3.2}$$

**Corollary 2.3.2. (The basic MC integration)** *Consider the multidimensional integral (1.0.1) then, if we pick $N$ points $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$ randomly from the multidimensional region $\Gamma$, then*

$$I \simeq V < f > \pm V\frac{\hat{\sigma}_f}{\sqrt{N}}$$

*where $V$ is the volume of $\Gamma$ and*

$$< f >= \frac{1}{N}\sum_{i=1}^{N} f(\boldsymbol{x}_i), \quad \hat{\sigma}_f^2 =< f^2 > - < f >^2$$

*Proof.* Since $\Gamma$ is a $d$-dimensional hyper-rectangular $[a_1, b_1] \times \ldots \times [a_d, b_d] \subset \Re^d$, then the integral $I$ is

$$I = \int_{a_d}^{b_d} \int_{a_{d-1}}^{b_{d-1}} \ldots \int_{a_1}^{b_1} f(\mathbf{x}) d\mathbf{x}$$

where $a_i$ and $b_i \in \Re$, $\forall\ i = 1, \ldots, d$; from Theorem (2.3.1), choose

$$\phi(\mathbf{x}) = \frac{1}{\prod_{i=1}^{d}(b_i - a_i)} = \frac{1}{V}$$

(the p.d.f. of the multivariate uniform distribution on $\Gamma$, see Appendix (A.3.1)) and $h(\mathbf{x}) = \prod_{i=1}^{d}(b_i - a_i) f(\mathbf{x}) = V f(\mathbf{x})$. Then

$$\hat{I} = \frac{V}{N}\sum_{i=1}^{N} f(\mathbf{x}_i)$$

is an estimate of $I$,

$$\begin{aligned} var(\hat{I}) &= \frac{V^2}{N^2} var\left(\sum_{i=1}^{N} f(\mathbf{x}_i)\right) \\ &= \frac{V^2}{N^2}\sum_{i=1}^{N} var(f(\mathbf{x}_i)) \\ &= \frac{V^2}{N}\sigma_f^2 \end{aligned}$$

Now, the error

$$I = \hat{I} \pm V\frac{\sigma_f}{\sqrt{N}}$$

$$\begin{aligned} \sigma_f^2 &= E(f^2(\mathbf{x})) - E^2(f(\mathbf{x})) \\ &= \frac{1}{V}\int_{\Gamma} f^2(\mathbf{x}) d\mathbf{x} - \left(\frac{1}{V}\int_{\Gamma} f(\mathbf{x}) d\mathbf{x}\right)^2 \\ &\simeq \frac{1}{N}\sum_{i=1}^{N} f^2(\mathbf{x}_i) - \left(\frac{1}{N}\sum_{i=1}^{N} f^2(\mathbf{x}_i)\right)^2 \\ &=< f^2 > - < f >^2 \\ &= \hat{\sigma}_f^2 \end{aligned}$$

So,

$$I \simeq V < f > \pm V\frac{\hat{\sigma}_f}{\sqrt{N}}$$

□

The *Strong Law of Large Numbers* (SLLN) guarantees an *almost sure convergence* for the MC method as $N \to \infty$. Let $\varepsilon > 0$, then by the SLLN,

$$P\left(\lim_{N\to\infty}\left|\frac{V}{N}\sum_{n=1}^{N} f(\mathbf{x}_n) - I\right| < \varepsilon\right) = 1$$

Furthermore, the *Central Limit Theorem* (CLT) says that for a random variable $\mathbf{x}$ with mean $\mu$ and variance $\sigma^2$, the sum of $N$ samples of $\mathbf{x}$ (itself a random variable) is approximately distributed according to a Guassian with mean $N\mu$ and variance $N\sigma^2$ as $N \to \infty$. The random variable $f(\mathbf{x})$, where averaged is therefore approximately Guassian as well, with mean $I$ and variance $\frac{V^2}{N}\sigma_f^2$, and for large enough $N$, we have

$$\begin{aligned} p\left(\frac{-V\sigma_f}{\sqrt{N}} \le \frac{V}{N}\sum_{n=1}^{N} f(\mathbf{x}_n) - I \le \frac{V\sigma_f}{\sqrt{N}}\right) &\simeq \frac{1}{\sqrt{2\pi}}\int_{-1}^{1} \exp(\frac{-t^2}{2})dt \\ &\simeq 0.68268 \end{aligned}$$

Therefore, we have probabilistic error bounds; and since we haven't the exact value of $\sigma_f$, then there is less than 68.27% chance that the absolute error lies inside the estimated error bounds.

*Remark* 2.3.1. The CLT implies that the rate of convergence of the basic MC is $O(\frac{1}{\sqrt{N}})$, which is independent of the dimension of the problem, and the same accuracy can be expected in all dimensions for integrands. This makes the MC integration the preferred method for integrals in high dimensions.

**Example 2.2.** Consider the multidimensional integral, see Woolfson and Pert [5].

$$I = \int_0^1\int_0^1\int_0^1\int_0^1\int_0^1\int_0^1\int_0^1\int_0^1\int_0^1\int_0^1 2^{10}\prod_{i=1}^{10}\{\sin^3(\pi x_i)dx_i\} \tag{2.3.3}$$

The integral can be factorized into a product of 10 similar integrals ($\int_0^1 2\sin^3(\pi x)dx$) and its true value is easily found as 0.1942. We used the basic MC for this integral and got the results in Table 2.2.

From Table 2.2, we see that the absolute error is less than the estimated error, except when $N = 10^5$, this is because we have probabilistic error bounds. Note that, to get better estimated error and of course absolute error, we increase the number of samples $N$ or reduce the variance as we will see in Section(2.5). It is clear that we got

| N | Basic MC | Estimated error | Absolute error |
|---|---|---|---|
| 10,000 | 0.174743 | 0.022086 | 0.019457 |
| 100,000 | 0.182407 | 0.008792 | 0.011743 |
| 1000,000 | 0.195159 | 0.003042 | 0.000959 |

Table 2.2: Basic MC integration output of the integral in equation (2.3.3).

a very good estimate with absolute error up to three decimal places when using $10^6$ random points.

## 2.4 MC integration and the standard numerical integration techniques

Consider the $d$-dimensional integral $I$, given in equation (1.0.1). All common integration routines follow the same basic principle; an approximation $\tilde{I}$ of $I$, is calculated by the following basic formula

$$\tilde{I} = \tilde{I}_n = \sum_{i=1}^{n} w_i f(\mathbf{x}_i)$$

where $\mathbf{x}_i \in \Gamma$, $i = 1, 2, \ldots, n$, are called the *abscissas* or the *integration nodes* and $w_i \in \Re$, $i = 1, 2, \ldots, n$, are called the *weights*. The weights take values that correspond with the appropriate selection of the abscissas, with $\sum_{i=1}^{n} w_i = V$.

Note that, the abscissas of the basic MC integration are the random points taken in $\Gamma$; and the weights $w_i = \frac{V}{n}$ for $i = 1, 2, \ldots, n$.

In this section, we make a brief comparison between the basic MC integration and some standard numerical integration techniques. The comparison shows the computational advantages of the MC integration in three fields of integration:

1. Integration over complicated shapes.

2. High dimensional integration.

3. Integration of non-differentiable integrands.

### 2.4.1 Integration over complicated shapes

Suppose we want to integrate a function $g$ over a region $\mathcal{W}$ which is considered to be a complicated domain of integration (complicated shape). Most of the standard numerical integration techniques are based upon choosing the abscissas according to some even pattern in the domain of integration. This can lead to difficulties when the domain is of an unusual shape (like $\mathcal{W}$). A MC approach to this problem is by finding a region $\mathcal{V}$ which includes $\mathcal{W}$ that can easily be sampled, and then define $f$ to be equal $g$ for points in $\mathcal{W}$ and equal zero for points outside of $\mathcal{W}$ (but still inside the sampled region $\mathcal{V}$). And we try to make $\mathcal{V}$ enclose $\mathcal{W}$ as possible.

**Example 2.3.** See Press et al. [18]. Suppose we want to find the weight of the object in Figure 2.2.

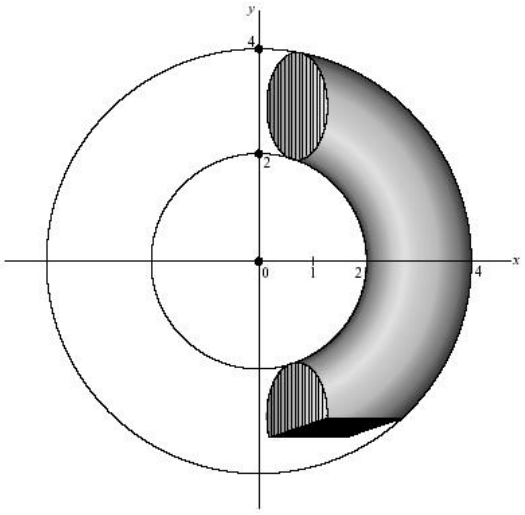


Figure 2.2: A portion of a torus

In particular the object is defined by the three simultaneous conditions

$$z^2 + (\sqrt{x^2 + y^2} - 3)^2 \leq 1$$

(torus centered on the origin with major radius=4 and minor radius=2)

$$x \geq 1, \quad y \geq -3$$

and suppose that the object has a constant density $\rho$. Then, we want to estimate the following integral:

$$\int \rho \, dxdydz$$

Note that the limits of integration of the region in Figure 2.2 cannot be written easily analytically in closed form, so we can't apply any numerical method, and the MC approach is one of the few methods that can be applied for this problem. We choose the region $\mathcal{V}$ (which enclose the piece of torus $\mathcal{W}$) to be the rectangular box extended from 1 to 4 in the $x$ axis, -3 to 4 in the $y$ axis, and from -1 to 1 in the $z$ axis, and we use the following algorithm to estimate the weight.

**Algorithm:**

**Parameters** $N, \rho$

**Step 0:** Initialization, $i = 0, n = 0$.

**Step 1:** Generate $(x_i, y_i, z_i)$ uniformly distributed inside the rectangular $\mathcal{V}$.

**Step 2:** If $z_i^2 + (\sqrt{x_i^2 + y_i^2} - 3)^2 \leq 1$ (the point is inside $\mathcal{W}$) then $n = n + 1$.
If $i < N$ then $i = i + 1$, Goto Step 1.

**Step 3:** Return $\frac{n}{N}\rho$

Thus, one of the strengths of MC integration is that it is simple to code as a computer algorithm regardless of the complexity of the domain of integration.

### 2.4.2 High dimensional integration

Consider the efficiency of the standard numerical integration in one dimension. The typical approach is to use one of the "numerical quadrature" techniques, for example, the trapezoidal and the Simpson's rules. Recall that the trapezoidal rule for one dimension integration problem is given by

$$\int_{x_0}^{x_0+\Delta x} f(x)dx = \frac{\Delta x}{2}[f(x_0) + f(x_0 + \Delta x)] - \frac{(\Delta x)^3}{12} f''(\xi)$$

where $x_0 \leq \xi \leq x_0 + \Delta x, f \in C^2[x_0, x_0 + \Delta x]$.

The Simpson's rule for this integral is given by

$$\int_{x_0}^{x_2} f(x)dx = \frac{\Delta x}{3}[f(x_0) + 4f(x_1) + f(x_2)] - \frac{(\Delta x)^5}{90} f^{(4)}(\xi)$$

where $\Delta x = x_2 - x_1 = x_1 - x_0$, $f \in C^4[x_0, x_2]$ and $\xi \in [x_0, x_2]$.

To approximate an integral over a finite interval $[x_0, x_n]$ with the help of these formulas, one can divide the interval into $n$ sub-intervals, and applies these methods into each sub-interval.

When the trapezoidal rule is used we get the composite trapezoidal rule

$$\int_{x_0}^{x_n} f(x)dx = \frac{x_n - x_0}{n} \sum_{j=0}^{n} w_j f(x_j) - \frac{1}{12} \frac{(x_n - x_0)^3}{n^2} f''(\xi)$$

where $w_0 = w_n = 1/2$, $w_j = 1$ for $1 \leq j \leq n-1$, $\xi \in [x_0, x_n]$ and $f \in C^2[x_0, x_n]$

When the Simpson's rule is used we get the composite Simpson's rule

$$\int_{x_0}^{x_n} f(x)dx = \frac{x_n - x_0}{n} \sum_{j=0}^{n} w_j f(x_j) - \frac{1}{180} \frac{(x_n - x_0)^5}{n^4} f^{(4)}(\xi)$$

where $n$ is an even number, $w_j = \begin{cases} \frac{1}{3} & \text{if } j = 0, n \\ \frac{4}{3} & \text{if } j \text{ is odd, } 1 \leq j < n \\ \frac{2}{3} & \text{if } j \text{ is even, } 1 \leq j < n \end{cases}$

$\xi \in [x_0, x_n]$ and $f \in C^4[x_0, x_n]$.

The generation of one dimensional integration formulas into $d$-dimensional integration formulas is by viewing the $d$-dimensional integral as an integral of one-dimension and apply a one dimensional integration in each iteration (the product form), see Evans [19]. The integral over the $d$-dimensional hypercube $[0, 1]^d$ evaluated with the help of the trapezoidal rule is given by

$$\int f(u_1, \ldots, u_d)d^d u = \frac{1}{n^d} \sum_{j_1=0}^{n} \cdots \sum_{j_d=0}^{n} w_{j_1} \ldots w_{j_d} f\left(\frac{j_1}{n}, \ldots \frac{j_d}{n}\right) + O(\frac{1}{n^2})$$

And when we use the Simpson's rule, the integral is given by

$$\int f(u_1, \ldots, u_d)d^d u = \frac{1}{n^d} \sum_{j_1=0}^{n} \cdots \sum_{j_d=0}^{n} w_{j_1} \ldots w_{j_d} f\left(\frac{j_1}{n}, \ldots \frac{j_d}{n}\right) + O(\frac{1}{n^4})$$

Now, we discuss the rate of convergence for the three approaches given in this section.

In both cases (trapezoidal and Simpson's rules), we need $N = (n+1)^d \approx n^d$ function evaluations. Since the necessary computing time is proportional to $N$, we conclude that the error scales as $N^{-2/d}$ for the trapezoidal method and as $N^{-4/d}$ for the Simpson's method.

As we have seen in the previous section, the MC integration converges to the true value with rate $N^{-1/2}$. The following table summarizes our discussion, see Lux and Koblinger [10] and Yakowitz [20].

| | conv. rate in one-dim. | conv. rate in $d$-dim. |
|---|---|---|
| Basic MC | $N^{-1/2}$ | $N^{-1/2}$ |
| Trapezoidal rule | $N^{-2}$ | $N^{-2/d}$ |
| Simpson's rule | $N^{-4}$ | $N^{-4/d}$ |

Table 2.3: The convergence rate in one and in $d$-dimensions for the three methods.

We can see in Table 2.3 that the MC integration has a convergence rate that is independent of the number of dimensions. The MC integration converges faster than the trapezoidal rule in five dimensions or more and faster than the Simpson's rule in nine or more dimensions. Therefore, MC integration becomes the method of choice for higher dimensional integration.

Now, suppose we want to apply any quadrature formula to approximate a 30-dimensional integral. So, we generate a grid in the domain of integration and we take the sum (with the respective weights to the chosen formula) of the function values in the grid points. Suppose a grid with 10 nodes on each coordinate axis in the 30-dimensional cube $[0,1]^{30}$ be chosen. In this case, we have $10^{30}$ abscissas. Suppose a time of $10^{-7}$ second is necessary for calculating one value of the function. Therefore, a time of $10^{23}$ second will be necessary for evaluating the integral (recall that 1 year= $31536 \times 10^3$ second) then we need more than $10^{15}$ years which equals $10^6$ billion years to approximate the integral, see Karaivanova and Dimov [6]. On the other hand, few seconds are needed to estimate a 30-dimensional integral using the MC rules.

### 2.4.3 Integration of non-differentiable integrals

It is noted that we need the integrand to have continuous second derivative over the domain of integration to calculate the error of the trapezoidal rule, and to have continuous fourth derivative to calculate the error of the Simpson's rule. Most of the numerical techniques need the integrands to have continuous derivatives of a specific order. On the other hand, the MC method does not need the function to be differentiable. So, if the integrand fails to be regular (i.e., to have continuous derivative of moderate order), then the numerical analytic techniques such as trapezoidal and Simpson's rules become less attractive (especially in high dimensional integrals) and the MC method becomes more acceptable to use in this case.

## 2.5 Variance reduction techniques

Consider the multidimensional integral given in equation (1.0.1). We have seen that the error estimate of a basic MC integration equals $V\hat{\sigma}_f/\sqrt{N}$, see Section (2.4.2); so, the MC estimate for the integral converges relatively slow to the true value (with rate $N^{-1/2}$). In this section, we review several techniques to improve this situation, by reducing the variance of MC integration; such techniques are called *The Variance Reduction Techniques*. There are several variance reduction techniques that can be applied to MC methods such as, importance sampling, correlated sampling, stratified sampling and quota sampling. Below, we review the most two common used techniques; the importance sampling and the stratified sampling.

### 2.5.1 Importance sampling

Importance sampling is the most common variance reduction technique used in MC methods. The basic idea of this technique, see Rubinstein [9], consists of concentrating the distribution of the sample points in the parts of the region $\Gamma$ that are of most "importance" instead of spreading them out eventually. Let $\mathbf{x}$ be a random vector

with p.d.f. $g(\mathbf{x})$, then we can represent the integral (1.0.1) as

$$I = \int_\Gamma \frac{f(\mathbf{x})}{g(\mathbf{x})} g(\mathbf{x}) d\mathbf{x} = E\left(\frac{f(\mathbf{x})}{g(\mathbf{x})}\right)$$

Now, from Theorem (2.3.1), choose $\phi(\mathbf{x}) = g(\mathbf{x})$ and $h(\mathbf{x}) = \frac{f(\mathbf{x})}{g(\mathbf{x})}$. So, if $N$ independent samples $\mathbf{x}_i$ are selected from $\Gamma$, according to the p.d.f. $g(\mathbf{x})$, then the MC estimate $\hat{I}$ of the integral $I$ is

$$\hat{I} = \frac{1}{N} \sum_{i=1}^{N} \frac{f(\mathbf{x}_i)}{g(\mathbf{x}_i)}$$

with variance, see equation (2.3.2).

$$\begin{aligned} var(\hat{I}) &= \frac{1}{N}\left[\int_\Gamma \frac{f^2(\mathbf{x})}{g^2(\mathbf{x})} g(\mathbf{x}) dx - I^2\right] \\ &= \frac{1}{N}\left[E\left(\frac{f^2(\mathbf{x})}{g^2(\mathbf{x})}\right) - E^2\left(\frac{f(\mathbf{x})}{g(\mathbf{x})}\right)\right] \\ &= \frac{1}{N}\sigma^2_{f/g} \end{aligned}$$

So, the statistical error of the MC integration with Importance Sampling is given by $\frac{\sigma_{f/g}}{\sqrt{N}}$, where an estimator of $\sigma^2_{f/g}$ is

$$\begin{aligned} \hat{\sigma}^2_{f/g} &= \frac{1}{N}\sum_{i=1}^{N}\left(\frac{f(\mathbf{x}_i)}{g(\mathbf{x}_i)}\right)^2 - \left(\frac{1}{N}\sum_{i=1}^{N}\frac{f(\mathbf{x}_i)}{g(\mathbf{x}_i)}\right)^2 \\ &= \frac{1}{N}\sum_{i=1}^{N}\left(\frac{f(\mathbf{x}_i)}{g(\mathbf{x}_i)}\right)^2 - \hat{I}^2 \end{aligned}$$

$g(\mathbf{x})$ is called the *importance sampling distribution*. The variance can be essentially reduced if $g(\mathbf{x})$ is chosen in order to have a shape similar to that of $f(\mathbf{x})$, see Rubinstein [9].

### The Importance Sampling procedure

**Step 0:** Specify the importance sampling distribution $g(\mathbf{x})$.

**Step 1:** Generate $X_1, X_2, \ldots, X_N$ as random points in $\Gamma$, according to the p.d.f. $g(\mathbf{x})$.

**Step 2:** Estimate the integral $I$ using the formula:

$$\hat{I} = \frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{g(X_i)}$$

**Example 2.4.** Consider the integral

$$I = \int_0^1 \frac{4}{1+x^2} dx \tag{2.5.1}$$

which represents the value of $\pi$. Firstly, we apply the basic MC without any variance reduction techniques, then we apply the MC with Importance Sampling, where we use the importance sampling distribution $g(x) = \frac{1}{3}(4-2x)$ for $0 \leq x \leq 1$ and zero otherwise, so we need to generate random variables from the distribution function $G(x) = \frac{x}{3}(4-x)$. We use the inverse transformation method, see Appendix (A.2.1).

$$u = \frac{x}{3}(4-x) \Rightarrow x = 2 \pm \sqrt{4-3u}$$

Since $x$ must take values in the range from 0 to 1, then $x = 2 - \sqrt{4-3u}$, where $u \sim U(0,1)$. Then we use the formula

$$\hat{I} = \frac{1}{N} \sum_{i=1}^{N} \frac{12}{(1+x_i^2)(4-2x_i)}$$

Tables 2.4 and 2.5 show the results:

| N | MC estimate | Estimated error | Absolute error |
|---|---|---|---|
| 100 | 3.258927 | 0.060424 | 0.117334 |
| 1,000 | 3.130409 | 0.020641 | 0.011184 |
| 10,000 | 3.138297 | 0.006422 | 0.003295 |
| 100,000 | 3.142365 | 0.002035 | 0.000772 |
| 1000,000 | 3.141100 | 0.000643 | 0.000492 |

Table 2.4: The basic MC output for the integral in equation (2.5.1) without any variance reduction techniques.

| N | MC estimate | Estimated error | Absolute error |
|---|---|---|---|
| 100 | 3.154019 | 0.007395 | 0.012427 |
| 1,000 | 3.139069 | 0.002545 | 0.002523 |
| 10,000 | 3.141479 | 0.000253 | 0.000113 |
| 100,000 | 3.142138 | 0.000802 | 0.000545 |
| 1000,000 | 3.141521 | 0.000080 | 0.000071 |

Table 2.5: The basic MC output of the integral in equation (2.5.1) using the Importance Sampling technique.

When we apply the MC procedure without any variance reduction technique, we get an acceptable result when we use $10^4$ random points where we get an absolute

error equals to 0.003295. When using 100 random points, we get a bad result. A good estimate of $\pi$ is reached with absolute error equals to 0.000492 when using $10^6$ random points.

On the other hand, when using the MC method with the Importance Sampling technique, we get an acceptable result when we use only 1000 random points, in which we reach the absolute error 0.002523. When using 100 random points we get a relatively small error (0.02427). We get a good estimate of $\pi$ with absolute error equals to 0.000545 when we use $10^5$ random points. We get a very good result when using $10^6$ random points where we get absolute error equals to 0.000071.

It is also noted that the estimated error gives always a good picture about the absolute error.

### 2.5.2 Stratified sampling

The Stratified Sampling is an attempt to ensure that important regions of the domain get sampled. This can be done by dividing the whole integration region into subregions, performing a basic MC integration in each subregion, and adding up the partial results at the end, see Lux and Koblinger [10] and Rubinstein [9]. Mathematically, this is based on the fundamental property of the Riemann integral

$$\int_a^b f(x)dx = \int_a^c f(x)dx + \int_c^b f(x)dx, \quad a < c < b$$

More generally, we split the region $\Gamma$ into $k$ subregions $\Gamma_i$, $i = 1, 2, \ldots, k$ each of volume $V_i$. In each region, we perform a basic MC integration with $N_i$ points, for the integral $I$ and we obtain the estimate

$$\hat{I} = \sum_{i=1}^{k} \frac{V_i}{N_i} \sum_{j=1}^{N_i} f(\mathbf{x}_{ij})$$

and

$$\begin{aligned} var(\hat{I}) &= \sum_{i=1}^{k} \frac{V_i^2}{N_i^2} \sum_{j=1}^{N_i} \sigma_f^2|_{\Gamma_i} \\ &= \sum_{i=1}^{k} \frac{V_i^2}{N_i} \sigma_f^2|_{\Gamma_i}, \end{aligned}$$

where $\sigma_f|_{\Gamma_i}$ is the standard deviation of the function $f$ over the subregion $\Gamma_i$.

$$\begin{aligned}\sigma_f^2|_{\Gamma_i} &= E(f^2)|_{\Gamma_i} - E^2(f)|_{\Gamma_i} \\ &= \frac{1}{V_i}\int_{\Gamma_i} f^2(\mathbf{x})d\mathbf{x} - \left(\frac{1}{V_i}\int_{\Gamma_i} f(\mathbf{x})d\mathbf{x}\right)^2.\end{aligned}$$

An estimator of $\sigma_f^2|_{\Gamma_i}$ is

$$\hat{\sigma}_f^2|_{\Gamma_i} = \frac{1}{N_i}\sum_{j=1}^{N_i} f^2(\mathbf{x}_{ij}) - \left(\frac{1}{N_i}\sum_{j=1}^{N_i} f(\mathbf{x}_{ij})\right)^2.$$

If the subregions and the number of points in each subregion are chosen carefully, then this can lead to a dramatic reduction in the variance compared with basic MC. But it should be noted that it can also lead to a larger variance if the choices are not appropriate.

**Example 2.5.** Consider the following two dimensional integral

$$I = \int_0^1 \int_0^1 f(x,y)dxdy$$

where $f(x) = \frac{x^4y^4}{x^4+y^4+1}$.

We have applied the basic MC to estimate $I$. The results are given in Table 2.6.

| N | MC estimate | Estimated error |
|---|---|---|
| 100 | 0.013727 | $3.840166 \times 10^{-3}$ |
| 1,000 | 0.018787 | $1.348394 \times 10^{-3}$ |
| 10,000 | 0.019717 | $4.331535 \times 10^{-4}$ |
| 100,000 | 0.019717 | $1.391403 \times 10^{-4}$ |
| 1000,000 | 0.019737 | $4.389176 \times 10^{-5}$ |

Table 2.6: The basic MC estimate of the integral $I$.

Figure 2.3 shows the graph of $f(x,y)$, one can see that $f(x,y)$ is approximately zero on a relatively large parts of the domain of integration.

We divided the domain of integration into three subregions: $A$, $B$ and $C$ where $A$ is the square $A = \{(x,y);\ 0.35 \le x \le 1,\ 0.35 \le y \le 1\}$, $B$ is the rectangle $B = \{(x,y);\ 0 \le x \le 0.35,\ 0.35 \le y \le 1\}$ and $C$ is the rectangle $C = \{(x,y);\ 0 \le x \le 1,\ 0 \le y \le 0.35\}$; as in Figure 2.4.

The function is approximately zero on the regions $B$ and $C$. We have performed $(0.11375N)$ random points in region $B$ and $(0.175N)$ random points in region $C$ (which

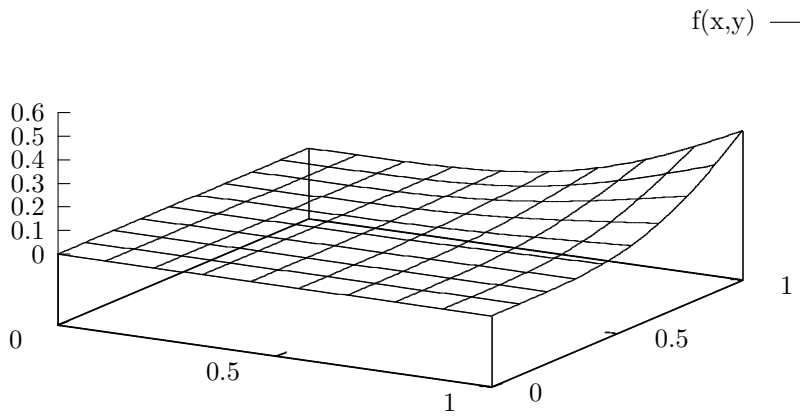


Figure 2.3: The graph of $f(x, y)$.

is the half of number of generated points in $C$ and $B$ using the basic MC without Stratified Sampling). Table 2.7 shows the results; it is clear that we got a better estimated errors than the estimated errors in Table 2.6

| N | MC estimate | Estimated error |
|---|---|---|
| 100 | 0.016037 | $2.414786 \times 10^{-3}$ |
| 1,000 | 0.018963 | $8.269671 \times 10^{-4}$ |
| 10,000 | 0.019677 | $2.718933 \times 10^{-4}$ |
| 100,000 | 0.019657 | $8.652684 \times 10^{-5}$ |
| 1000,000 | 0.019735 | $2.743859 \times 10^{-5}$ |

Table 2.7: The basic MC estimate of the integral $I$ using the Stratified Sampling technique.

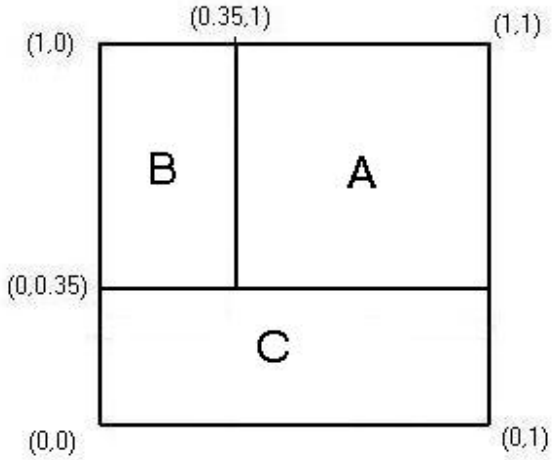


Figure 2.4: The proposed division of the integration domain.

# Chapter Three

# Adaptive Monte Carlo Integration

In this chapter, we consider the multidimensional integral given in equation (1.0.1), in which we do not have any priori information about $f(\mathbf{x})$. The value of the integrand can vary significantly in the domain of integration, and in some cases, there can be only small parts of the domain in which the integrand is non-zero (or non-constant). Since the basic MC method distributes the evaluation points as uniform as possible, it will waste calculations on regions that are not important, we need a procedure that allocates the resources to the important parts of the domain. If we have priori information about the location of the important (or non-important) parts, then we can use the Stratified Sampling technique, see Section (2.5.2). In many cases these information are not available certainly in multidimensional integrals, and for this case, we can use the *adaptive* techniques, which learn about the function as they proceed.

In the next section, we describe the general adaptive MC procedure that uses the idea of Stratified Sampling iteratively. Then, in Section (3.2), we give a deep description of the adaptive MC algorithm. We first describe the division of a hyper-rectangular region and propose an efficient way of storage and transition into the next iteration. Then we give a detailed adaptive MC algorithm that can be applied directly to estimate the solution of the multidimensional integral given in equation (1.0.1). Finally, in Section (3.3), we consider several test multidimensional integrals. We give a detailed outputs of the adaptive algorithm and make comparisons between adaptive MC algorithms with different parameters.

## 3.1 General adaptive MC algorithm

Adaptive algorithms aim to reduce the estimated (absolute) error(s) of the integration estimation (approximation). The main difference between the adaptive algorithms and non-adaptive algorithms (integration formulas) is that, the non-adaptive algorithms use a fixed set of abscissas and weights; so, all nodes of integration and weights can be generated before any evaluation of $f$ is performed. On the other hand, the adaptive algorithms use the results from previous integrand evaluations for generating the following abscissas and weights.

The key concept of the *adaptive Monte Carlo* (AMC) integration is to apply the basic MC method to smaller subregions of the original integration domain. This needs a subdivision strategy. There are two common types of subdivision, see Schürer [13].

**Local subdivision strategy:** The subregions in each iteration are classified as being either *active* or *inactive*, in order to decide whether or not they are eligible for further subdivision. This decision for each subregion to be either active or inactive is independent of the other regions.

**Global subdivision strategy:** The division is performed using information about all the current subregions. A popular subdivision strategy is to subdivide the region(s) with the largest estimated error(s), until some stopping criterion is satisfied.

The iterative MC adaptive algorithm discussed and tested here uses a global subdivision strategy, and can be described as follows: In the first iteration, the basic MC method is applied to the whole integration domain to estimate the integral and the standard error. The region that represents the integration domain is stored in a region collection. In the second iteration, the whole region is divided into subregions and the basic MC method is applied to each of these subregions. These subregions are stored in the region collection. Next, the region that has the largest estimated integral error is taken and divided into new subregions. The algorithm continues in this manner until a stopping criterion is satisfied. The estimated integral is the sum

of the estimated integrals at each subregion from the region collection, see Dahl [12] and Shürer [13, 14]. The following algorithm describes the general form of an AMC integration algorithm, see Schürer [14].

**Algorithm 1:**

- Put the whole domain of integration into region collection.
- **While** Stopping criterion is not satisfied **Do**
  Choose subregion with largest estimated error.
  Split this region.
  Apply the basic MC method to newly created regions.
  Store new regions in region collection.
- **Endwhile**.

One can use the stopping criterion to be a specific number of iterations to be terminated, or a specified estimated error to be reached. For example, until the total estimated integral error is less than $\varepsilon$ for some real number $\varepsilon > 0$.

## 3.2 A deep description of an adaptive MC integration algorithm

It is easy to understand the basic idea of Algorithm 1 given in the previous section, but if one decides to apply it, he must face several hard questions: How to make a split of the regions? How to store all these regions, the estimated integrals and the errors in each iteration in an efficient way?

In this section, we give a deep and simple description of Algorithm 1, in which we describe how to perform the $s$-division, and how to store all the needed variables in an efficient way. We propose a detailed AMC algorithm that can be applied directly to estimate the solution of the multidimensional integral given in equation (1.0.1).

To proceed, we need the following notations:

1. $\Omega_i$ : the region collection in iteration $i$.

2. $\Gamma_i(j)$ : the subregion that belongs to $\Omega_i$ with index $j$; where $\bigcup_j \Gamma_i(j) = \Omega_i$ and $\Gamma_i(j) \cap \Gamma_i(k) = \phi$ for $k \neq j$.

3. $\hat{I}_{ij}$ : the basic MC integration estimate over the region $\Gamma_i(j)$.

4. $\hat{I}_i$ : denotes the integral estimate over all the integral domain in iteration $i$. So,

$$\hat{I}_i = \sum_j \hat{I}_{ij}$$

5. $\hat{E}_{ij}$ : the standard estimated error of integration over the subregion of index $j$ in iteration $i$; $(\Gamma_i(j))$. So,

$$\hat{E}_{ij} = V_{ij} \frac{\hat{\sigma}_f|_{\Gamma_i(j)}}{\sqrt{N_{ij}}} \tag{3.2.1}$$

where $V_{ij}$ is the volume of the region $\Gamma_i(j)$, $N_{ij}$ the number of random points taken in $\Gamma_i(j)$ and $\hat{\sigma}_f|_{\Gamma_i(j)}$ is the estimated standard deviation of $f$ over the subregion $\Gamma_i(j)$.

6. $\hat{E}_i$ : the standard error estimate of integration over the whole integration domain. So,

$$\hat{E}_i = \sqrt{\sum_j \hat{E}_{ij}^2}$$

7. $\alpha_i$ : the promising index; the index of the region that has the maximum estimated error in iteration $i$; i.e.,

$$\alpha_i = \arg\max_j \hat{E}_{ij} \tag{3.2.2}$$

8. $\Gamma_i^*$ : the promising region; the region that has the maximum estimated error in iteration $i$, so $\Gamma_i^*$ is the region whose index is the promising index.

$$\Gamma_i^* = \Gamma_i(\alpha_i) \tag{3.2.3}$$

### 3.2.1 The division of a hyper-rectangle

The division of the hyper-rectangle $\Gamma = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_d, b_d] \subset \Re^d$ can be done by taking $s$ coordinates (where $s$ is a positive integer with $1 \leq s \leq d$), and divide

each coordinate into two equal (or non-equal) intervals (the $s$-division). To make a *one*-division of the region $\Gamma_i(j)$, the fundamental property of division of the Riemann integral in one dimension is generalized to $d$-dimensions as follows:

$$\int_{a_d}^{b_d}\dots\int_{a_k}^{b_k}\dots\int_{a_1}^{b_1} f(\mathbf{x})d\mathbf{x} = \int_{a_d}^{b_d}\dots\int_{a_k}^{c}\dots\int_{a_1}^{b_1} f(\mathbf{x})d\mathbf{x} + \int_{a_d}^{b_d}\dots\int_{c}^{b_k}\dots\int_{a_1}^{b_1} f(\mathbf{x})d\mathbf{x}$$

where $c \in (a_k, b_k)$, $1 \leq k \leq d$ and $\mathbf{x} \in \Re^d$.

And to make a *two*-division of the same region, the idea of division of the square into four subregions is used and generalized to get the following formula

$$\begin{aligned}\int_{a_d}^{b_d}\dots\int_{a_{k1}}^{b_{k2}}\dots\int_{a_{k2}}^{b_{k2}}\dots\int_{a_1}^{b_1} f(\mathbf{x})d\mathbf{x} = &\int_{a_d}^{b_d}\dots\int_{a_{k1}}^{c_1}\dots\int_{a_{k2}}^{c_2}\dots\int_{a_1}^{b_1} f(\mathbf{x})d\mathbf{x} \\ &+ \int_{a_d}^{b_d}\dots\int_{c_1}^{b_{k1}}\dots\int_{a_{k2}}^{c_2}\dots\int_{a_1}^{b_1} f(\mathbf{x})d\mathbf{x} \\ &+ \int_{a_d}^{b_d}\dots\int_{a_{k1}}^{c_1}\dots\int_{c_2}^{b_{k2}}\dots\int_{a_1}^{b_1} f(\mathbf{x})d\mathbf{x} \\ &+ \int_{a_d}^{b_d}\dots\int_{c_1}^{b_{k1}}\dots\int_{c_2}^{b_{k2}}\dots\int_{a_1}^{b_1} f(\mathbf{x})d\mathbf{x}\end{aligned}$$

with $c_j \in (a_{kj}, b_{kj})$, where $1 \leq kj \leq d$ for $j = 1, 2$, and $\mathbf{x} \in \Re^d$.

Note that, when using *one*-division, we get two subregions; when using the *two*-division, we get four subregions; and if we perform a *three*-division we will get eight subregions. In general, the $s$-division gives us $M = 2^s$ subregions.

In iteration 0, $\Omega_0$ contains only one region; namely, the whole integration domain. In iteration 1, we divide the whole integration domain into $M$ subregions, so $|\Omega_1| = M$. In iteration 2, we choose the worst region (which has the maximum estimated error) and divide it into $M$ subregions, so $|\Omega_2| = M - 1 + M = 2M - 1$. In iteration 3, $|\Omega_3| = (2M-1)-1+M = 3M-2$; and in general, $|\Omega_i| = iM-(i-1) = i(M-1)+1$. So,

$$\Omega_i = \{\Gamma_i(1), \Gamma_i(2), \dots, \Gamma_i(v_i)\} \tag{3.2.4}$$

where $v_i = i(M-1)+1$.

So, the total estimated integral in iteration $i$ is given by

$$\hat{I}_i = \sum_{j=1}^{v_i} \hat{I}_{ij}$$

and the total estimated error in iteration $i$; $(\hat{E}_i)$ is given by

$$\hat{E}_i = \sqrt{\sum_{j=1}^{v_i} \hat{E}_{ij}^2}$$

In each iteration $i$, we need to choose $s$ distinct coordinates to be divided namely; $r_{i1}, \ldots, r_{is}$, and we need to specify a reasonable values for $c_1, c_2, \ldots, c_s$ (where $c_j$ divides the axis $r_{ij}$, $j = 1, 2, \ldots, s$).

In our AMC algorithm, we choose the divided coordinates *randomly* and we use two different ways for selecting $c_j$, $j = 1, 2, \ldots, s$. The first way is $c_j = (a_{kj} + b_{kj})/2$ (case1), and hence, with each iteration $i$, $\Gamma_i$ is divided into $M = 2^s$ regions with equal volumes. The second way of selecting $c_j$, $(j = 1, \ldots, s)$ is $c_j = (b_{kj} - a_{kj})u + a_{kj}$ where $u \sim U(0, 1)$ (case2); and in this case, $c_j$ is taken randomly in the interval $(a_{kj}, b_{kj})$.

### 3.2.2 The storage approach

Recall that, in iteration $i$, we need to store $v_i = i(M - 1) + 1$ subregions $(\Gamma_i(j), j = 1, \ldots, v_i)$ in $\Omega_i$. Each subregion can be represented by $d$ lower bounds and $d$ upper bounds of the $d$-dimensional integration, namely; $(a_1, \ldots, a_d)$ and $(b_1, \ldots, b_d)$, respectively. To store all regions, we use two $v_i \times d$ arrays; $A^i$ and $B^i$, where $i$ denotes the $i$-th iteration.

The $k$-th row of $A^i$ and $B^i$ stores the lower bounds and upper bounds respectively, of the region $\Gamma_i(k)$. The $l$-th column of $A^i$ and $B^i$ contains the lower bounds and upper bounds respectively, of the $l$-th coordinate of all subregions $\Gamma_i(j), j = 1, 2, \ldots, v_i$.

For example, suppose the subregions in iteration $i$ are defined as follows

$$\Gamma_i(j) \equiv \{\mathbf{x}, \mathbf{x} \in [a_{j1}, b_{j1}] \times [a_{j2}, b_{j2}] \times \ldots \times [a_{jd}, b_{jd}]\} \quad , j = 1, 2 \ldots, v_i$$

then all these subregions are represented by the following two arrays.

$$A^i = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1d} \\ a_{21} & a_{22} & \ldots & a_{2d} \\ \vdots & & & \\ a_{v_i 1} & a_{v_i 2} & \ldots & a_{v_i d} \end{pmatrix}$$

and

$$B^i = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1d} \\ b_{21} & b_{22} & \dots & b_{2d} \\ \vdots & & & \\ b_{v_i 1} & b_{v_i 2} & \dots & b_{v_i d} \end{pmatrix}$$

Also, we need to store the estimated MC integrals ($\hat{I}_{ij}$) and the standard estimated errors ($\hat{E}_{ij}$) over each region $\Gamma_i(j)$, $j = 1, \dots, v_i$, from $\Omega_i$. To store the estimated MC integrals ($\hat{I}_{ij}$), we use the vector array $TM^i$,

$$TM^i = [\hat{I}_{i1}, \dots, \hat{I}_{iv_i}]$$

where

$$\hat{I}_{ij} = \frac{V_{ij}}{N_{ij}} \sum_{k=1}^{N_{ij}} f(\mathbf{x}_k), \quad j = 1, 2, \dots, v_i. \tag{3.2.5}$$

with $\mathbf{x}_k = (x_{k1}, x_{k2} \dots, x_{kd})$, $x_{kl} \sim U(A^i(j,l), B^i(j,l))$ for $l = 1, 2, \dots, d$; $N_{ij}$ is the number of random points generated in $\Gamma_i(j)$ and $V_{ij}$ is calculated as follows

$$V_{ij} = \prod_{k=1}^{d} (B^i(j,k) - A^i(j,k))$$

To store the standard error estimates ($\hat{E}_{ij}$) over all regions of $\Omega_i$, we use the vector array $TE^i$

$$TE^i = [\hat{E}_{i1}, \dots, \hat{E}_{iv_i}]$$

where $\hat{E}_{ij}$ is calculated using equation (3.2.1).

### 3.2.3 The transition from iteration $i$ to iteration $i+1$

Given $\Omega_i$ in iteration $i$, as in equation (3.2.4) then $\Gamma_i^* = \Gamma_i(\alpha_i)$ will be divided into $M$ subregions $\Gamma_{i+1}(\alpha_i), \Gamma_{i+1}(\alpha_i+1), \dots, \Gamma_{i+1}(\alpha_i+M-1)$. So, $\Omega_i$ is updated as follows

$$\Omega_{i+1} = \{\Gamma_i(1), \dots, \Gamma_i(\alpha_i - 1), \Gamma_{i+1}(\alpha_i), \Gamma_{i+1}(\alpha_i + 1), \dots, \Gamma_{i+1}(\alpha_i + M - 1), \\ \Gamma_i(\alpha_i + 1), \dots, \Gamma_i(v_{i+1})\}$$

Recall that, $A^{i+1}$ and $B^{i+1}$ represent $\Omega_{i+1}$; so, they contain $M$ new rows. The two arrays $A^i$ and $B^i$ are updated into $A^{i+1}$ and $B^{i+1}$ respectively, instead of the update of $\Omega_i$ into $\Omega_{i+1}$, as follows: The first $(\alpha_i - 1)$ rows of $A^{i+1}$ and $B^{i+1}$ are the same as

the first $(\alpha_i - 1)$ rows of $A^i$ and $B^i$, respectively. The last $(v_i - \alpha_i)$ rows of $A^{i+1}$ and $B^{i+1}$ are the same as the last $(v_i - \alpha_i)$ rows of $A^i$ and $B^i$, respectively. The remaining rows of $A^{i+1}$ and $B^{i+1}$ (which are the rows $\alpha_i, (\alpha_i+1), \ldots, (\alpha_i+M-1)$, that represent the new $M$ regions in $\Omega_{i+1}$) are a copy rows of the $\alpha_i$ row of $A^i$ and $B^i$ respectively, except for some values in their intersection with the $s$ columns $r_{i1}, r_{i2}, \ldots, r_{is}$, that correspond with the $s$ divided coordinates. This intersection will become the following array (the $MA_{M\times s}$ array) in $A^{i+1}$.

$$MA = \begin{pmatrix} * & * & \ldots & * & * \\ * & * & \ldots & * & c_s \\ * & * & \ldots & c_{s-1} & * \\ * & * & \ldots & c_{s-1} & c_s \\ & & \vdots & & \\ * & c_2 & \ldots & * & * \\ * & c_2 & \ldots & * & c_s \\ * & c_2 & \ldots & c_{s-1} & * \\ * & c_2 & \ldots & c_{s-1} & c_s \\ & & \vdots & & \\ c_1 & c_2 & \ldots & c_{s-1} & c_s \end{pmatrix}_{M\times s} \tag{3.2.6}$$

where $*$ (the star) denotes that the value in the corresponding location is unchanged (the original value in the $\alpha_i$ row of $A^i$). Note that $MA$ is not a portion array.

One can find the corresponding modified array in $B^{i+1}$ (the $MB_{M\times s}$ array) by replacing the unchanged values (the stars) in $MA$ by $c_j$'s in the $j$-th column, $j = 1, \ldots, s$, and replacing the $c_j$'s in the array (3.2.6) by stars (become unchanged values); or simply, by reversing the rows of the $MA$ array. So,

$$MB = \begin{pmatrix} c_1 & c_2 & \dots & c_{s-1} & c_s \\ c_1 & c_2 & \dots & c_{s-1} & * \\ c_1 & c_2 & \dots & * & c_s \\ c_1 & c_2 & \dots & * & * \\ & & \vdots & & \\ c_1 & * & \dots & c_{s-1} & c_s \\ c_1 & * & \dots & c_{s-1} & * \\ c_1 & * & \dots & * & c_s \\ c_1 & * & \dots & * & * \\ & & \vdots & & \\ * & * & \dots & * & * \end{pmatrix}_{M\times s}$$

For example, if we make a *one*-division, then

$$MA = \begin{pmatrix} * \\ c_1 \end{pmatrix}, \quad MB = \begin{pmatrix} c_1 \\ * \end{pmatrix}$$

And in this case, $A^i$ and $B^i$ are updated as follows

$$A^{i+1}(k,l) = \begin{cases} A^i(k,l), & \text{if } k \le \alpha_i,\ l = 1,\dots,d \\ A^i(k,l), & \text{if } k = \alpha_i + 1,\ l = 1\dots,d, l \ne r_{i1} \\ c_1, & \text{if } k = \alpha_i + 1, l = r_{i1} \\ A^i(k-1,l), & \text{if } \alpha_i + 2 \le k \le i+2,\ l = 1,\dots,d \end{cases} \tag{3.2.7}$$

and

$$B^{i+1}(k,l) = \begin{cases} B^i(k,l), & \text{if } k < \alpha_i,\ l = 1,\dots,d \\ B^i(k,l), & \text{if } k = \alpha_i,\ l = 1\dots,d, l \ne r_{i1} \\ c_1, & \text{if } k = \alpha_i, l = r_{i1} \\ B^i(k-1,l), & \text{if } \alpha_i + 1 \le k \le i+2,\ l = 1,\dots,d \end{cases} \tag{3.2.8}$$

And if we perform a *two*-division, then

$$MA = \begin{pmatrix} * & * \\ * & c_2 \\ c_1 & * \\ c_1 & c_2 \end{pmatrix}, \quad MB = \begin{pmatrix} c_1 & c_2 \\ c_1 & * \\ * & c_2 \\ * & * \end{pmatrix}$$

The value of all $c_j$'s, where $j = 1,\dots,s$, are one of the following two cases:

$$\begin{aligned} &\textbf{Case1} : \tfrac{A^i(\alpha_i,r_{ij}) + B^i(\alpha_i,r_{ij})}{2} \\ &\textbf{Case2} : (B^i(\alpha_i,r_{ij}) - A^i(\alpha_i,r_{ij}))u + A^i(\alpha_i,r_{ij}),\ \text{where } u \sim U(0,1) \end{aligned} \tag{3.2.9}$$

Note that, we use only one choice of $c$ (case1 or case2) in the adaptive algorithm.

After finding $\alpha_i$ and after updating $A^i$ and $B^i$, $TM^i(l)$ is updated as follows

$$TM^{i+1}(l) = \begin{cases} \hat{I}_{il}, & \text{if } l < \alpha_i \\ \hat{I}_{(i+1)l}, & \text{if } l = \alpha_i, \ldots, (\alpha_i + M - 1) \\ \hat{I}_{i(l-1)}, & \text{if } l > \alpha_i + M - 1 \end{cases} \tag{3.2.10}$$

So, we need to calculate $\hat{I}_{(i+1)\alpha_i}$, $\hat{I}_{(i+1)(\alpha_i+1)}$, ..., and $\hat{I}_{(i+1)(\alpha_i+M-1)}$, using equation (3.2.5).

The total estimated error in iteration $i$; $(TE^i)$ is updated in the same way as $TM^i$, and in iteration $(i+1)$, We need to calculate $\hat{E}_{(i+1)\alpha_i}$, $\hat{E}_{(i+1)(\alpha_i+1)}$, ..., and $\hat{E}_{(i+1)(\alpha_i+M-1)}$ using equation (3.2.1).

### 3.2.4 A detailed AMC algorithm

We set $N_{ij}$ to be a constant $N$ for all regions in all iterations. When using the *one*-division; then, in iteration 0, the adaptive algorithm uses $N$ random points; in iteration 1, the algorithm uses $2N$ random points; and in iteration $i$, the adaptive algorithm uses $N(i+1)$ random points. When using the *two*-division, in the second iteration (iteration 1), $4N$ random points are used; in iteration 2, the algorithm uses $4N - N + 4N = 7N$ random points; and in iteration $i$, $N(3i+1)$ random points are used.

In general, when using the $s$-division; then, in iteration $i$, the adaptive algorithm uses $N((2^s - 1)i + 1)$ random points.

We need to specify the values for $c_j$, $j = 1, 2, \ldots, s$, using equation (3.2.9), and throughout the adaptive algorithm, we use only one case choice of $c$ (either case1 or case2). We now present a detailed AMC integration algorithm.

**Algorithm 2**

**Parameters:** $T$ (the maximum number of iterations), $d$ (the dimension of the problem), $N$ (the generated random points in each new region), $s$ (the number of divided coordinates, where $0 \leq s \leq d$).

**Step 1:** $A^0(1,l) = a_l$, $B^0(1,l) = b_l$, $\forall\ l = 1, \ldots, d$.

**step 2:** Calculate $\hat{I}_{01}$ and $\hat{E}_{01}$ and set $TM^0 = [\hat{I}_{01}]$, $TE^0 = [\hat{E}_{01}]$, $\alpha_0 = 1$, $i = 0$.

**Step 3:** $*$ Find distinct random integers $r_{ij} \in [1, d]$, $j = 1, \ldots, s$.

$*$ Calculate $c_j$, $j = 1, 2, \ldots, s$, using equation (3.2.9).

$*$ Find $A^{i+1}$ and $B^{i+1}$ as in Section (3.2.3) (if the *one*-division is performed then, use equations (3.2.7) and (3.2.8) respectively)

$*$ Calculate $\hat{I}_{(i+1)\alpha_i}, \hat{I}_{(i+1)(\alpha_i+1)}, \ldots, \hat{I}_{(i+1)(\alpha_i+M-1)}$ using equation (3.2.5), and update $TM^i$ using equation (3.2.10).

$*$ Calculate $\hat{E}_{(i+1)\alpha_i}, \hat{E}_{(i+1)(\alpha_i+1)}, \ldots, \hat{E}_{(i+1)(\alpha_i+M-1)}$ using equation (3.2.1) and upadate $TE^i$.

$*$ Find $\alpha_{i+1}$ using equation (3.2.2).

$*$ $i = i + 1$.

**Step 4:** If $i < T$ Goto Step 3.

**Step 5:** The estimated integral is

$$\hat{I}_T = \sum_{j=1}^{v_T} TM^T(j)$$

and the estimated error is

$$\hat{E}_T = \sqrt{\sum_{j=1}^{v_T} (TE^T(j))^2}$$

## 3.3 Numerical examples

In this section, we consider the following three different multidimensional integrals, that have been considered by Karaivanova and Dimov [6].

$d = 4$

$$J_1 = \int_0^1 \int_0^1 \int_0^1 \int_0^1 \frac{4x_1x_3^2 \exp(2x_1x_3)}{(1 + x_2 + x_4)^2} dx_1dx_2dx_3dx_4$$

$d = 25$

$$J_2 = \int_0^1 \ldots \int_0^1 \frac{4x_1x_3^2 \exp(2x_1x_3)}{(1 + x_2 + x_4)^2} \exp(x_5 + \ldots + x_{20}) x_{21}x_{22} \ldots x_{25} dx_1 \ldots dx_{25}$$

$d = 30$

$$J_3 = \int_0^1 \dots \int_0^1 \frac{4x_1x_3^2 \exp(2x_1x_3)}{(1+x_2+x_4)^2} \exp(x_5 + \dots + x_{20}) x_{21}x_{22}\dots x_{30} dx_1 \dots dx_{30} \tag{3.3.1}$$

We give two detailed output of Algorithm 2 using case1 choice of $c$, see equation (3.2.9), for estimating $J_3$. We first use a *one*-division (case1-AMC1), and then a *two*-division (case1-AMC2). Then, we make comparisons between the basic MC, case1-AMC1 and case2-AMC1 algorithms for estimating solutions for the set of multi-dimensional integrals given in equation (3.3.1). Finally, we make comparisons between case1-AMC2 and case2-AMC2 in estimating $J_1$, $J_2$ and $J_3$.

### 3.3.1 A detailed output

We use case1-AMC1 algorithm in estimating $J_3$. We use $N = 5 \times 10^4$ random points. Table 3.1 shows the output of 4 iterations in details.

| Iteration $i$ | $\Gamma_i(j)$ | Estimated integral | Estimated error |
|---|---|---|---|
| iteration 0 | $\Gamma_0(1)$ | 3.351107 | 0.221240 |
| | $\Gamma$ | **3.351107** | **0.221240** |
| iteration 1 | $\Gamma_1(1)$ | 1.210097 | 0.062587 |
| | $\Gamma_1(2)$ | 2.289098 | 0.239714 |
| | $\Gamma$ | **3.499195** | **0.247750** |
| iteration 2 | $\Gamma_2(1)$ | 1.210097 | 0.062587 |
| | $\Gamma_2(2)$ | 1.253465 | 0.067066 |
| | $\Gamma_2(3)$ | 0.690014 | 0.040054 |
| | $\Gamma$ | **3.153576** | **0.100097** |
| iteration 3 | $\Gamma_3(1)$ | 1.210097 | 0.062587 |
| | $\Gamma_3(2)$ | 0.313236 | 0.020022 |
| | $\Gamma_3(3)$ | 0.935178 | 0.046537 |
| | $\Gamma_3(4)$ | 0.690014 | 0.040054 |
| | $\Gamma$ | **3.148525** | **0.08993** |

Table 3.1: A detailed 4 iterations output of case1-AMC1 algorithm using $N = 5 \times 10^4$ for estimating $J_3$.

In iteration 0, the estimated integrals is actually the basic MC integration over $\Gamma$ (the whole integration domain). In iteration 1, $\Gamma$ is divided into two subregions: $\Gamma_1(1)$ and $\Gamma_1(2)$, where the basic MC method is applied to each region. It is noted that the estimated error of the integral in iteration 1 is worse than the estimated error

in iteration 0. This maybe because of the bad division of $\Gamma_0(1)$, and this problem is considered and discussed in the next chapter. We can see that the promising region in each iteration is $\Gamma_0(1)$, $\Gamma_1(2)$, $\Gamma_2(2)$, $\Gamma_3(1)$ respectively.

Table 3.2 shows the estimated integral and errors and the absolute errors using case1-AMC1 algorithm for 10 iterations in estimating $J_3$. It is known that the exact value of $J_3$ is 3.244.

| $i$ | $\hat{I}_i$ | $\hat{E}_i$ | Absolute error |
|---|---|---|---|
| 0 | 3.351107 | 0.221240 | 0.107107 |
| 1 | 3.499195 | 0.247750 | 0.255195 |
| 2 | 3.153576 | 0.100097 | 0.090424 |
| 3 | 3.148525 | 0.089933 | 0.095475 |
| 4 | 3.256002 | 0.109971 | 0.012002 |
| 5 | 3.234978 | 0.082208 | 0.009022 |
| 6 | 3.272827 | 0.081168 | 0.028827 |
| 7 | 3.238134 | 0.074790 | 0.005866 |
| 8 | 3.348110 | 0.088251 | 0.104110 |
| 9 | 3.235390 | 0.066780 | 0.007610 |

Table 3.2: A 10 iterations output of Algorithm 2 (case1) using $N = 5 \times 10^4$ for estimating $J_3$.

In general, we can see that the estimated errors of the integral decrease slowly until reaching 0.06678 in iteration 9. The absolute errors are less than the estimated errors except in three places (in iterations 1, 3 and 8). It is noted that case1-AMC1 algorithm moves to a worse estimate (bigger estimated error) in iterations 1, 4 and 8.

We use case1-AMC2 algorithm in estimating $J_3$. We use $N = 15,000$ random points. Table 3.3 shows the output of 3 iterations in details.

We can see in Table 3.3 that in iteration 1, $\Gamma$ is divided into four subregions, the promising region is $\Gamma_1(4)$; so, we divide it into four subregions and we get seven regions in $\Omega_2$, the promising region is $\Gamma_2(7)$. If we perform a fourth iteration, then $\Gamma_2(7)$ will be divided into new four subregions.

Table 3.4 shows the estimated integral and errors using case1-AMC2 algorithm for 10 iterations with $N = 15,000$ in estimating $J_3$.

From Table 3.4, the absolute errors are less than the estimated errors except in two places (in iterations 6 and 9). We get an estimated error equals to 0.054800 in

| Iteration $i$ | $\Gamma_i(j)$ | Estimated integral | Estimated error |
|---|---|---|---|
| iteration 0 | $\Gamma_0(1)$ | 3.349646 | 0.354838 |
| | $\Gamma$ | **3.349646** | **0.354838** |
| iteration 1 | $\Gamma_1(1)$ | 0.060929 | 0.006080 |
| | $\Gamma_1(2)$ | 0.784820 | 0.059846 |
| | $\Gamma_1(3)$ | 0.176693 | 0.020387 |
| | $\Gamma_1(4)$ | 2.456285 | 0.227911 |
| | $\Gamma$ | **3.478727** | **0.236596** |
| iteration 2 | $\Gamma_2(1)$ | 0.060929 | 0.006080 |
| | $\Gamma_2(2)$ | 0.784820 | 0.059846 |
| | $\Gamma_2(3)$ | 0.176693 | 0.020387 |
| | $\Gamma_2(4)$ | 0.330183 | 0.027868 |
| | $\Gamma_2(5)$ | 0.456953 | 0.029239 |
| | $\Gamma_2(6)$ | 0.578463 | 0.051105 |
| | $\Gamma_2(7)$ | 0.961674 | 0.129045 |
| | $\Gamma$ | **3.349716** | **0.157892** |

Table 3.3: A detailed 3 iterations output of case1-AMC2 algorithm using $N = 15,000$ for estimating $J_3$.

| $i$ | $\hat{I}_i$ | $\hat{E}_i$ | Absolute error |
|---|---|---|---|
| 0 | 3.349646 | 0.354838 | 0.105646 |
| 1 | 3.478727 | 0.236596 | 0.234727 |
| 2 | 3.349716 | 0.157892 | 0.105716 |
| 3 | 3.290011 | 0.101583 | 0.046011 |
| 4 | 3.225158 | 0.087790 | 0.018842 |
| 5 | 3.176290 | 0.073511 | 0.067710 |
| 6 | 3.108422 | 0.063654 | 0.135578 |
| 7 | 3.198509 | 0.064926 | 0.045491 |
| 8 | 3.200352 | 0.060123 | 0.043648 |
| 9 | 3.178501 | 0.054800 | 0.065499 |

Table 3.4: A 10 iterations output of case1-AMC2 using $N = 15,000$ for estimating $J_3$.

iteration 9, which is better than the estimated error of case1-AMC1 in Table 3.2; where we got an estimated error equals to 0.06678. Note that, in Table 3.2, we used $N = 5 \times 10^4$; so, in iteration 9, case1-AMC1 algorithm uses $5 \times 10^4 \times (9+1) = 5 \times 10^5$ random points and during all iterations we use $(5 \times 10^4) \times 55(regions) = 2.75 \times 10^6$ samples. On the other hand, in Table 3.4, we use $N = 15,000$ only; so, in iteration 9, case1-AMC2 algorithm uses $15000 \times (3 \times 9 + 1) = 420000$ random points and during all iterations, we use $(1.5 \times 10^4) \times 145(regions) = 2.175 \times 10^6$. So, with a smaller number of generated points; case1-AMC2 gives better estimate than case1-AMC1 for estimating $J_3$ in 10 iterations.

### 3.3.2 A numerical study of the estimated errors

We consider the set of multidimensional integrals given is equation (3.3.1), and we apply the *one*-division and the *two*-division methods in Algorithm 2 for estimating the set of multidimensional integrals. So, we apply case1-AMC1, case2-AMC1, case1-AMC2 and case2-AMC2 algorithms for estimating $J_1$, $J_2$ and $J_3$.

We first apply case1-AMC1 and case2-AMC1 algorithms. Figures 3.1, 3.2 and 3.3 show the behavior of the estimated errors using case1-AMC1 and case2-AMC1 algorithms for estimating $J_1$, $J_2$ and $J_3$ in 20 iterations with $N = 5 \times 10^4$. Recall that, in iteration $k$, we use a total of $5(k+1) \times 10^4$ random points (this is the total number of generated points in iteration $k$, but we generate in each iteration $10^5$ random points only). The estimated errors of the basic MC integration of $J_1$, $J_2$ and $J_3$ are calculated using the same number of random points generated using Algorithm 2 (case1-AMC1 and case2-AMC1) in each iteration.

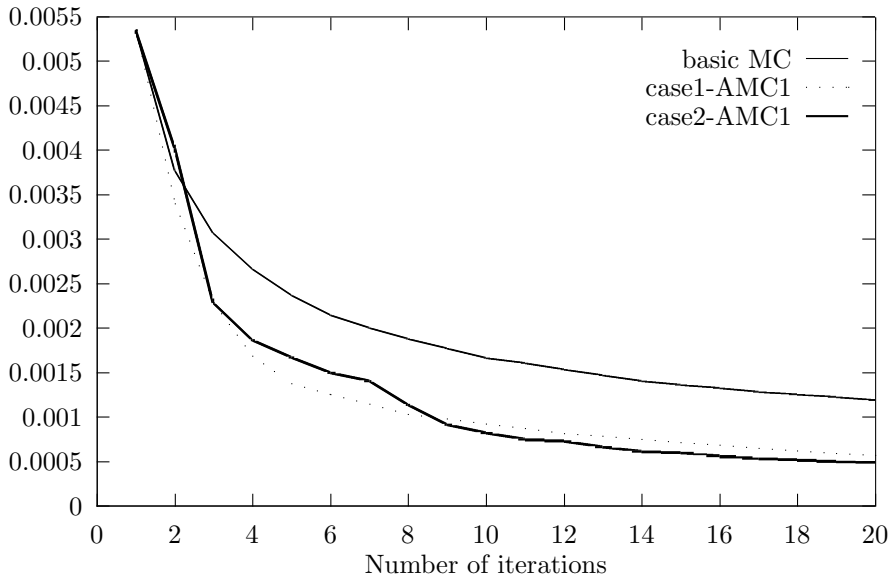


Figure 3.1: The estimated errors of the three algorithms applied to estimate $J_1$.

In Figure 3.1, it is clear that the two adaptive algorithms (case1-AMC1 and case2-AMC1) perform better than the basic MC. It is noted that after 9 iterations, case2-AMC1 algorithm performs a little better than case1-AMC1. The estimated errors of the three algorithms always go to a better estimate.

In Figures 3.2 and 3.3, it is noted that case1-AMC1 algorithm performs better than

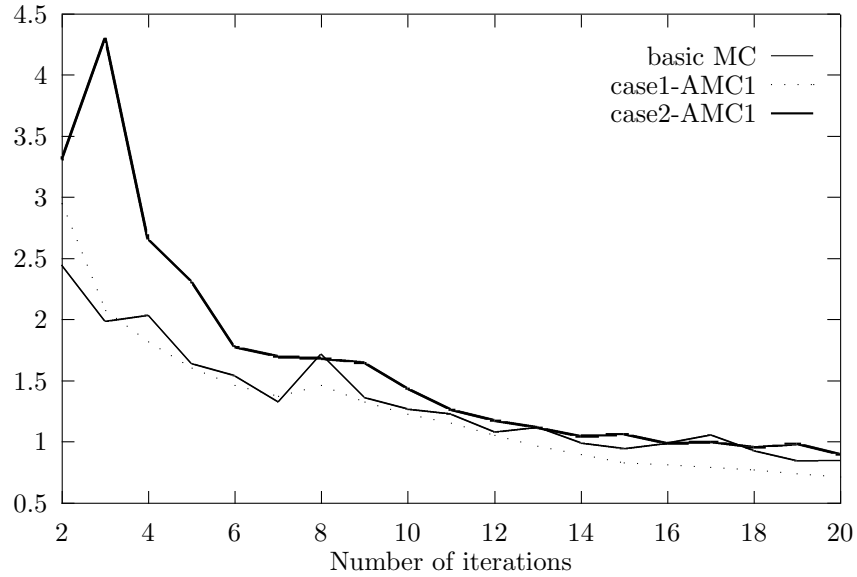


Figure 3.2: The estimated errors of the three algorithms applied to estimate $J_2$.

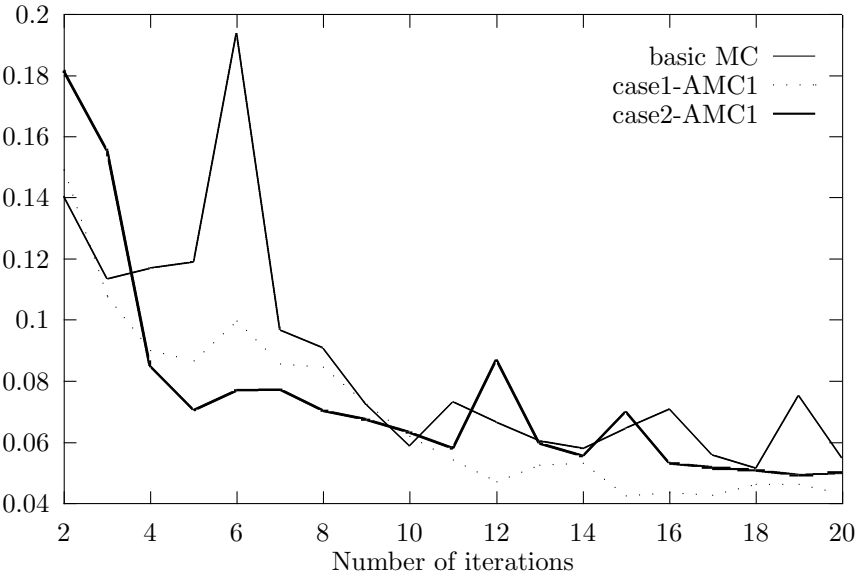


Figure 3.3: The estimated errors of the three algorithms applied to estimate $J_3$.

the two other algorithms. In Figure 3.2, case2-AMC1 algorithm is the worst algorithm, that is because it goes to a worse estimate of the integral (bigger estimated error) in the 3-rd iteration (an early iteration), so, the next iterations are affected by this bad estimate. The three algorithms move several times to worse estimates. In Figure 3.3, after 10 iterations, case2-AMC1 algorithm and the basic MC seem to have the same performance. Also, the three algorithms move several times to worse estimates.

Now, we use Algorithm 2 using the *two*-division method; so, we apply case1-AMC2 and case2-AMC2 algorithms for estimating $J_1$, $J_2$ and $J_3$ given in equation (3.3.1). We perform two output for each estimation, the first output uses $N = 9,000$ sample points for each subregion; and the second uses $N = 17,241$ sample points, so, in this case, after 20 iterations, the algorithm uses approximately the same number of sample points used by the *one*-division based algorithm in Figures 3.1, 3.2 and 3.3.

Figures 3.4 and 3.5 represent the estimated errors of case1-AMC2 and case2-AMC2 algorithms applied for estimating $J_1$. In Figure 3.4, we use $N = 9,000$ only, so, in iteration 19 (after 20 iterations), the algorithm uses $522,000$ random points; and as we see, after 20 iterations, case1-AMC2 algorithm reaches a better estimated error (smaller) than the estimated error of case1-AMC1 in Figure 3.1, which uses $N = 5 \times 10^5$, so, in iteration 19 (after 20 iterations), the algorithm uses $10^6$ random points. Also, it is noted that case2-AMC1 (in Figure 3.1) performs better than case2-AMC2 (in Figure 3.4). In Figure 3.5, We increase $N$ to $17,241$ sample points, so after 20 iterations, the *two*-division based algorithm (case1-AMC2 and case2-AMC2) uses approximately the same number of random points used by the *one*-division based adaptive algorithm in Figure 3.1. It is clear, that case1-AMC2 algorithm is better than case1-AMC1 algorithm; and case2-AMC2 algorithm remains worse than case2-AMC1 algorithm.

Figures 3.6 and 3.7 represent the estimated errors of case1-AMC2 and case2-AMC2 algorithms applied for estimating $J_2$. In Figure 3.6, the two algorithms are worse than the two corresponding algorithms in Figure 3.2. Case1-AMC2 algorithm has a stable decreasing graph of estimated errors, and case2-AMC2 algorithm moves two times to a worse estimated error. In Figure 3.7, the estimated errors of case1-AMC2 algorithm

are better than the estimated errors of case1-AMC1 algorithm in Figure 3.2, and case2-AMC2 algorithm remains worse than the corresponding algorithm in Figure 3.2.

Figures 3.8 and 3.9 represent the estimated errors of case1-AMC2 and case2-AMC2 algorithms applied for estimating $J_3$. In Figure 3.8, The two graphs are a little worse than their corresponding graphs in Figure 3.3. The two algorithms move several times to a worse estimated error. In Figure 3.9, The two algorithms reach a better estimated error than the estimated error reached by their corresponding adaptive algorithms in Figure 3.3. Also, case1-AMC2 algorithm moves to a worse estimated error in the 6-th iteration; and case2-AMC2 algorithm moves three times to a worse estimate.

One of the problems in high dimensional integration is the existence of a unique algorithm that is efficient for all types of integrands, see Schürer [13, 14]. For this reason, the estimated errors obtained in estimating $J_2$ is bigger than the estimated errors obtained in estimating $J_1$ and $J_3$.

One can see that case2-AMCi, $i = 1, 2$, algorithms move to a worse estimated errors in a number of iterations greater than the number of iterations in which case1-AMCi algorithms move to a worse estimated errors.

It is observed that, most of the time case1-AMCi, $i = 1, 2$ algorithms gave us better estimates than case2-AMCi algorithms. So, we conclude that, the global subdivision strategy that divides the promising region into subregions with *equal* volumes is better than the global subdivision strategy that divides the promising region into *random size* subregions.

Also, it is noted that, the case$i$-AMC2, $i = 1, 2$, algorithms move to a worse estimated errors in a fewer number of iterations than the case$i$-AMC1 algorithms. It is obvious that the number of iterations in which the adaptive algorithm moves to a worse estimate decreases as $s$ increases ($s$ is the number of divided coordinates).

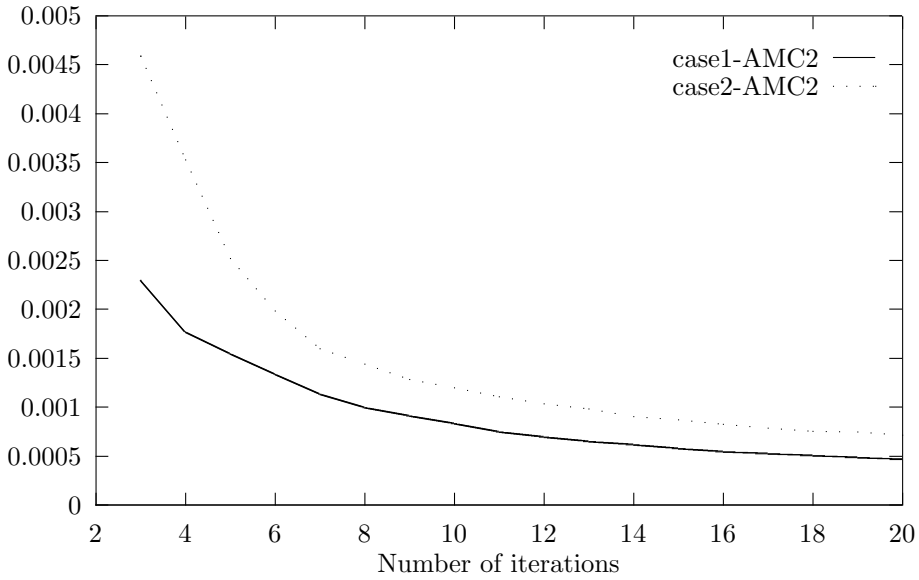


Figure 3.4: The estimated errors of case1-AMC2 and case2-AMC2 algorithms applied to estimate $J_1$ using $N = 9,000$.

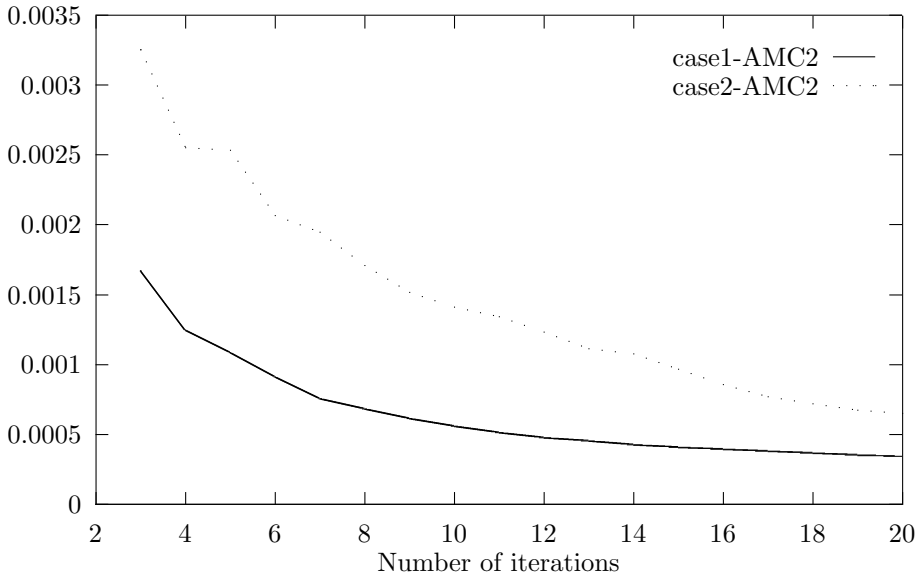


Figure 3.5: The estimated errors of case1-AMC2 and case2-AMC2 algorithms applied to estimate $J_1$ using $N = 17,241$.

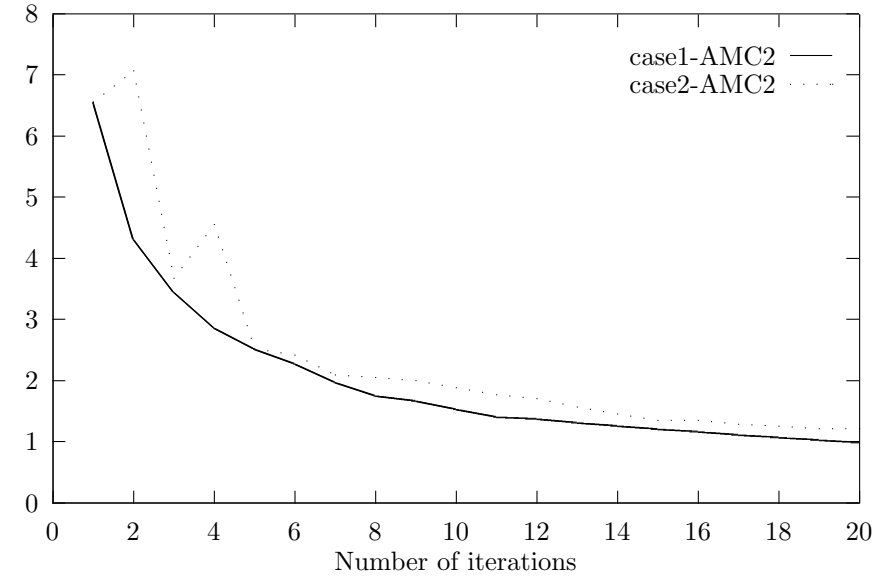


Figure 3.6: The estimated errors of case1-AMC2 and case2-AMC2 algorithms applied to estimate $J_2$ using $N = 9,000$.

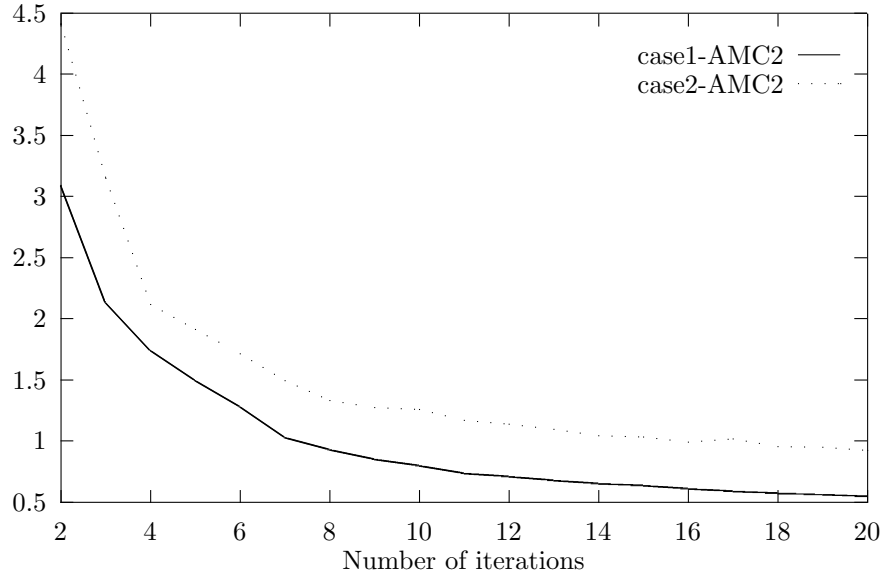


Figure 3.7: The estimated errors of case1-AMC2 and case2-AMC2 algorithms applied to estimate $J_2$ using $N = 17,241$.

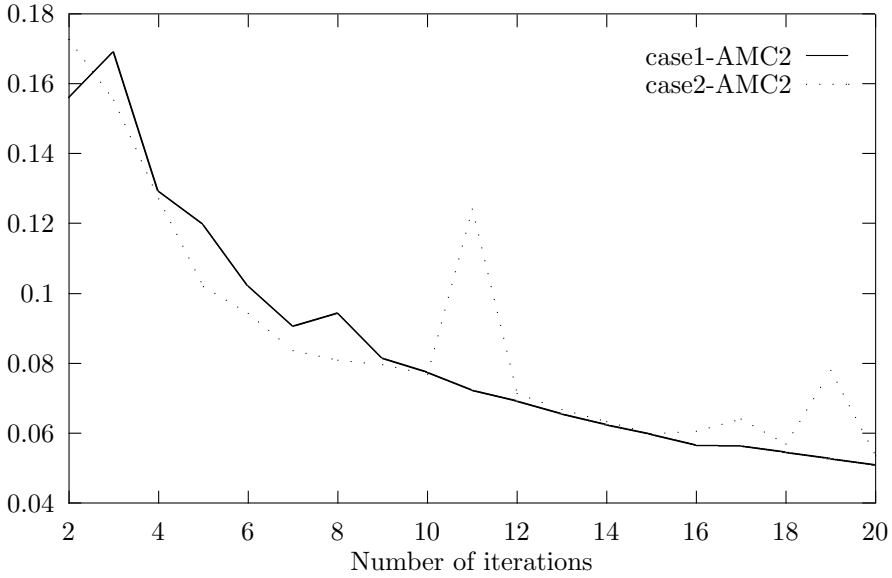


Figure 3.8: The estimated errors of case1-AMC2 and case2-AMC2 algorithms applied to estimate $J_3$ using $N = 9,000$.

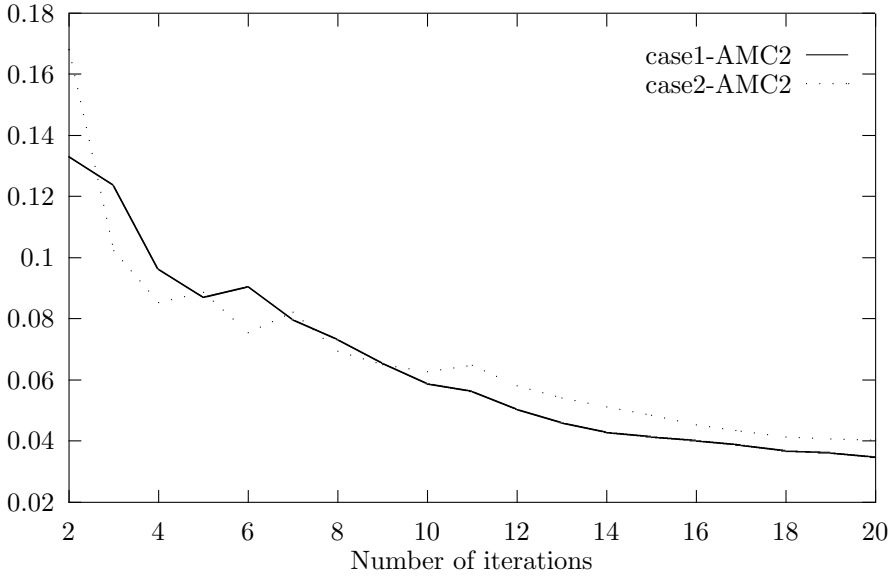


Figure 3.9: The estimated errors of case1-AMC2 and case2-AMC2 algorithms applied to estimate $J_3$ using $N = 17,241$.

# Chapter Four

# A Corrector for the Adaptive MC Algorithm

Consider the multidimensional integral $I$ given in equation (1.0.1). When Algorithm 2, given in Chapter 3 is applied to estimate $I$, an additional random points (abscissas) are generated and used in each iteration to estimate the integral. So, we expect that: Most of the time, Algorithm 2 moves to a better estimate (smaller estimated error). But as we have seen in Section (3.3), sometimes Algorithm 2 moves to a worse estimate of the integral. For example, in Table 3.1, the estimated error of the integral in iteration 0 is 0.22124 and in iteration 1 is 0.24775. In Table 3.2 we can see that the algorithm moves to worse estimates in 3 places (iterations 1, 3 and 8). In Table 3.4, the algorithm moves to a worse estimated error in iteration 7, and in Figures 3.2, 3.3, 3.6, 3.7, 3.8 and 3.9; it is clear that the adaptive algorithm (case1 and case2) moves several times to a worse estimate. This may happen because of the bad partition of the promising region.

In this chapter, we propose a corrector that prevents Algorithm 2 from moving to a worse estimate, so less time and less iterations might be needed to reach a specific estimated error. Moreover, the graph of the estimated error is always decreasing.

## 4.1 The detailed corrector algorithm

The corrector algorithm is summarized as follows: Suppose we are in iteration $i$, after finding the promising region in $\Omega_{i-1}$ (in iteration $(i-1)$), divide this region

using the $s$-division method into $M = 2^s$ subregions. The new regions are stored "temporarily" in the region collection, we calculate $\hat{E}_i$:

- If $\hat{E}_i \leq \hat{E}_{i-1}$ (the algorithm moves to a better estimate) then the promising region and index are found, and we go to the next iteration (as in Algorithm 2).
- If $\hat{E}_i > \hat{E}_{i-1}$ (the algorithm moves to a worse estimate, so the proposed division is not good), in this case, we go back to the previous iteration $(i-1)$, and make another division until we get $\hat{E}_i \leq \hat{E}_{i-1}$.

Recall that $\Omega_i$ is represented by $A^i$ and $B^i$. So, we need to store $A^i$, $B^i$, $TM^i$, $TE^i$, $\hat{E}_i$ and $\alpha_i$ in temporary locations in the memory. Then, we choose $s$ distinct coordinates to be divided. We update $A^i$ and $B^i$ into $A^{i+1}$ and $B^{i+1}$ respectively, as in Section (3.2.3). We calculate $\hat{E}_{(i+1)j}$, $j = 1, 2, \ldots, \alpha_i + M - 1$ using equation (3.2.1), and update $TE^i$ into $TE^{i+1}$; so, we can calculate

$$\hat{E}_{i+1} = \sqrt{\sum_{j=1}^{v_{i+1}} TE^{i+1}(j)} \tag{4.1.1}$$

Then check, if $\hat{E}_{i+1} \leq \hat{E}_i$, then find $TM^{i+1}$ and $\alpha_{i+1}$, and replace the stored arrays and values in the temporary locations by their updated values. Then we go to the next iteration as in Algorithm 2. On the other hand, if $\hat{E}_{i+1} > \hat{E}_i$, then restore the stored arrays and values, and repeat this iteration by a new values of $r_{ij}$, $j = 1, 2, \ldots, s$, and the corresponding values of $c_j$, $j = 1, 2, \ldots, s$, using equation (3.2.9) for case1 and case2, until we get a smaller estimated error.

The algorithm continues in this manner until a stopping criterion is satisfied.

We now present our *corrector adaptive MC* (CAMC) algorithm.

**Algorithm 3**

**Parameters:** $T$ (the maximum number of iterations), $d$ (the dimension of the problem), $N$ (the generated random points in each new region), $s$ (the number of divided coordinates, where $1 \leq s \leq d$).

**Step 1:** $A^0(1,l) = a_l$, $B^0(1,l) = b_l$, $\forall\ l = 1, \ldots, d$.

**step 2:** Calculate $\hat{I}_{01}$ and $\hat{E}_{01}$ and set $TM^0 = [\hat{I}_{01}]$, $TE^0 = [\hat{E}_{01}]$, $\alpha_0 = 1$, $\hat{E}_0 = \hat{E}_{01}$, $i = 0$.

**Step 3:**
- $*$ Store $A^i, B^i, TE^i, \hat{E}_i$ in temporary locations in the memory.
- $*$ Find a distinct random integers $r_{ij} \in [1, d]$, $j = 1, \ldots, s$.
- $*$ Calculate $c_j$, $j = 1, 2, \ldots, s$ using equation (3.2.9).
- $*$ Calculate $A^{i+1}$ and $B^{i+1}$ as in Section (3.2.3) (if the *one*-division is performed then, use equations (3.2.7) and (3.2.8) respectively).
- $*$ Calculate $\hat{E}_{(i+1)\alpha_i}, \hat{E}_{(i+1)(\alpha_i+1)}, \ldots, \hat{E}_{(i+1)(\alpha_i+M-1)}$, using equation (3.2.1).
- $*$ Update $TE^i$ to $TE^{i+1}$.
- $*$ Calculate $\hat{E}_{i+1}$ using equation (4.1.1).

**Step 4:**
- $*$ If $\hat{E}_{i+1} < \hat{E}_i$ then
  - Calculate $\hat{I}_{(i+1)\alpha_i}, \hat{I}_{(i+1)(\alpha_i+1)}, \ldots, \hat{I}_{(i+1)(\alpha_i+M-1)}$, using equation (3.2.5).
  - Update $TM^i$ to $TM^{i+1}$ using equation (3.2.10).
  - Find $\alpha_{i+1}$ using equation (3.2.2).
  - $i = i + 1$.
  - Go to Step 5.
- $*$ Else
  - Restore the stored arrays and vectors $A^i, B^i, TE^i, \hat{E}_i$.
  - Go to Step 3.

**Step 5:** If $i < T$, then Goto Step 3.

**Step 6:** The estimated integral is

$$\hat{I}_T = \sum_{j=1}^{v_T} TM^T(j)$$

and the estimated error is

$$\hat{E}_T = \sqrt{\sum_{j=1}^{v_T} (TE^T(j))^2}.$$

## 4.2 Numerical examples

In this section, we consider the set of multidimensional integrals $J_1$, $J_2$ and $J_3$ given in Section (3.3). We first discuss the performance of the basic MC, the CAMC algorithm with case1 choice of $c$, see equation (3.2.9), and using the *one*-division method (case1-CAMC1) and the CAMC algorithm with case2 choice of $c$ and using the *two*-division method (case2-CAMC1); in estimating $J_1$, $J_2$ and $J_3$. Then we make comparisons between the performance of case1-AMC1, case2-AMC1, case1-AMC2 and case2-AMC2 algorithms on one hand and case1-CAMC1, case2-CAMC1, case1-CAMC2 and case2-CAMC2 algorithms on the other hand in estimating $J_1$, $J_2$ and $J_3$. Finally, we give examples to show that the corrector algorithm works well.

### 4.2.1 A numerical study of the estimated error

We apply case1-CAMC1 and case2-CAMC1 algorithms into the set of multidimensional integrals given in equation (3.3.1), in which we perform 20 iterations and we use $N = 5 \times 10^4$. The estimated errors of case1-CAMC1 and case2-CAMC1 algorithms are calculated in each iteration. The estimated errors of the basic MC integration of the multidimensional integrals given in equation (3.3.1) are calculated using the same number of random points generated using the corrector algorithms in each iteration.

In Figure 3.1, it is clear that the two adaptive algorithms move always to better estimated error (the output needn't any correction). So, when applying case1-CAMC1 and case2-CAMC1 algorithms for estimating $J_1$, no differences occur on the performance between the CAMC algorithms and the AMC algorithms.

Figures 4.1 and 4.2 show the graphs of the estimated errors of the basic MC, case1-CAMC1 and case2-CAMC2 algorithms for estimating $J_2$ and $J_3$ respectively. We use the same initial seeds that we used in Figures 3.2 and 3.3 to put Algorithm 3 in the same conditions of Algorithm 2. It is clear that the corrector (case1 and case2) AMC moves in each iteration to a better estimate (smaller estimated error). The two corrector algorithms have always decreasing graph of estimated errors. Case1-CAMC1

algorithm performs better than the two other algorithms. In Figure 3.2, case2-AMC1 algorithm seems to have the same performance as the basic MC. On the other hand, in Figure 4.1, case2-CAMC1 algorithm seems to have the same performance as case1-CAMC1 algorithm.

Figures 4.3 and 4.4 show the graphs of the estimated errors of case1-CAMC2 and case2-CAMC2 in estimating $J_3$, using $N = 9,000$ and $N = 17,241$, respectively. To make a fair comparison, we use the same seeds used as in Figures 3.8 and 3.9, respectively. It is clear that the two corrector adaptive algorithms with the two cases of $c$ (case1-CAMC2 and case2-CAMC2), give a decreasing estimated errors.

Figures 4.5, 4.6, 4.7 and 4.8 show the differences between the estimated errors of the adaptive (case1 and case2) MC and the corrector adaptive (case1 and case2) MC algorithms in estimating $J_2$ and $J_3$, using the *one*-division.

Figures 4.9, 4.10, 4.11 and 4.12 show the differences between the estimated errors of the adaptive (case1 and case2) MC and the corrector adaptive (case1 and case2) MC algorithms in estimating $J_2$ and $J_3$, using the *two*-division.

It is clear that in all cases, the CAMC algorithms perform better than the AMC algorithms. The corrector algorithms (case1 and case2) always have decreasing graphs of estimated errors, and most of the time have better estimates than the AMC algorithms. For example, in Figure 4.5, the estimated errors of case1-AMC1 and case1-CAMC1 are the same until the 7-th iteration, where the first goes to a worse estimate and the second goes to a better estimate.

It is observed that, most of the time case1-CAMCi, $i = 1, 2$ algorithms gave better estimates than case2-CAMCi algorithms. So, we conclude that, the global subdivision strategy that divides the promising region into subregions with *equal* volumes is better than the global subdivision strategy that divides the promising region into *random size* subregions.

It is noted that, case$i$-CAMC2, $i = 1, 2$, algorithms move to a worse estimated error in a fewer number of iterations than case$i$-CAMC1 algorithms. It is also clear that the number of iterations in which the adaptive algorithm moves to a worse estimate

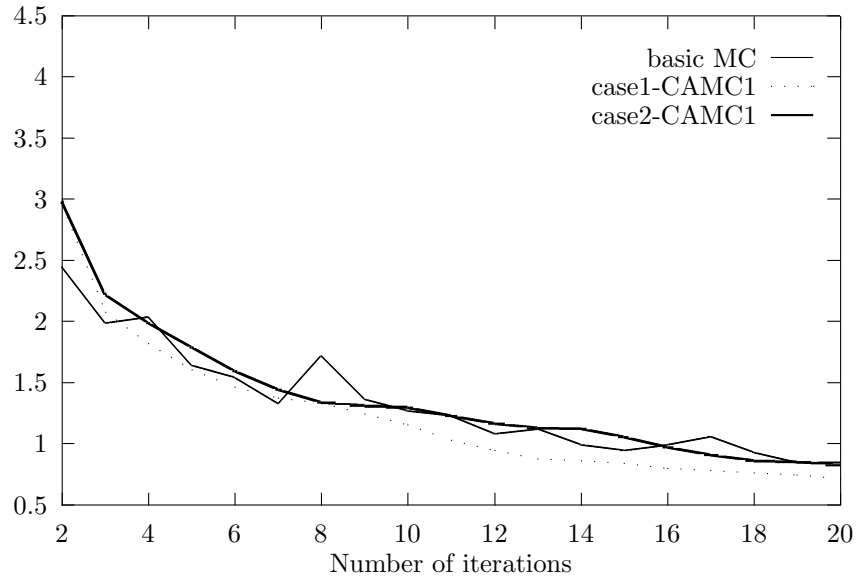


Figure 4.1: The estimated errors of the corrector algorithms (case1 and case2) and the basic MC algorithm for estimating $J_2$.

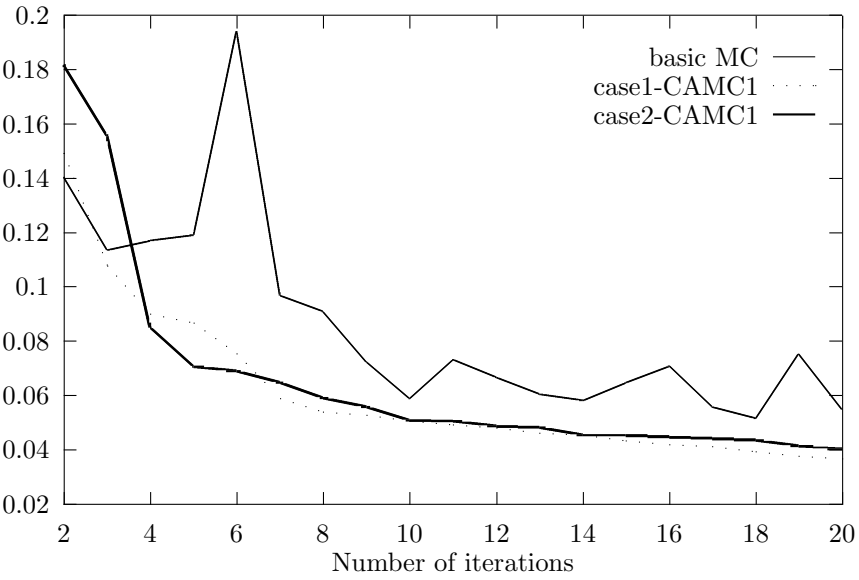


Figure 4.2: The estimated errors of the corrector algorithms (case1 and case2) and the basic MC algorithm for estimating $J_3$.

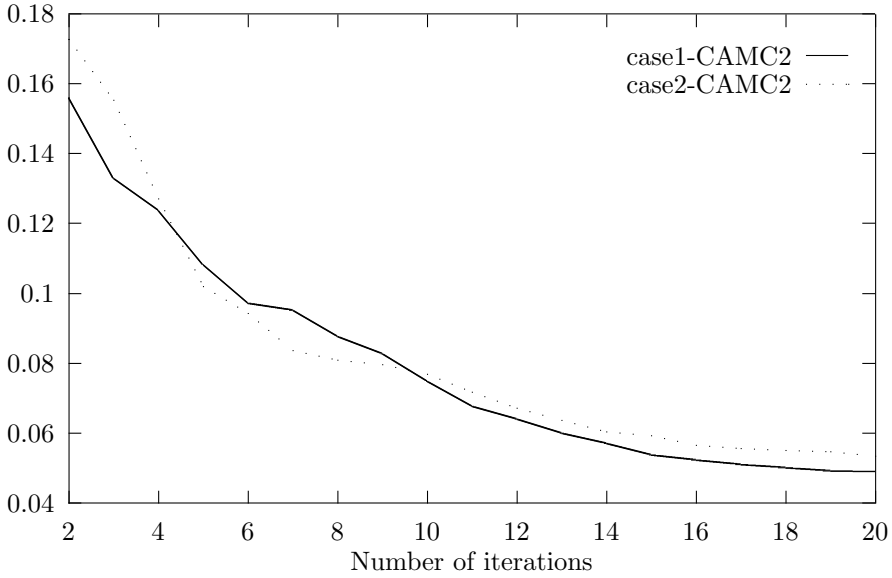


Figure 4.3: The estimated errors of case1-CAMC2 and case2-CAMC2 algorithms using $N = 9,000$ in estimating $J_3$.

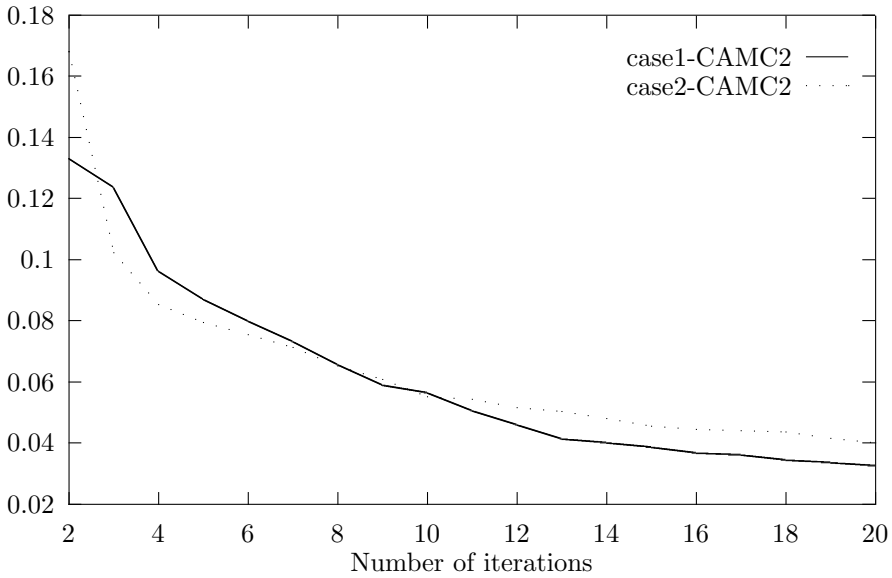


Figure 4.4: The estimated errors of case1-CAMC2 and case2-CAMC2 algorithms using $N = 17,241$ in estimating $J_3$.

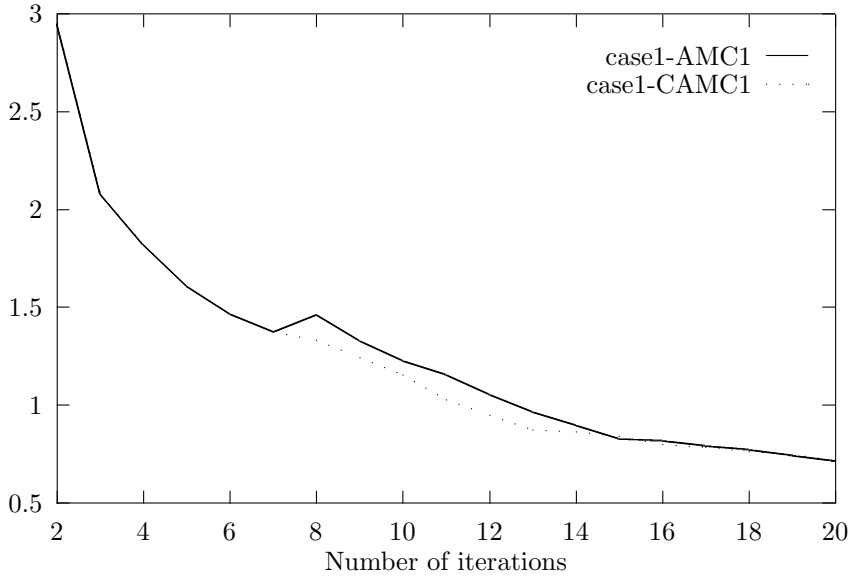


Figure 4.5: The estimated errors of case1-CAMC1 algorithm and case1-AMC1 algorithm for estimating $J_2$.

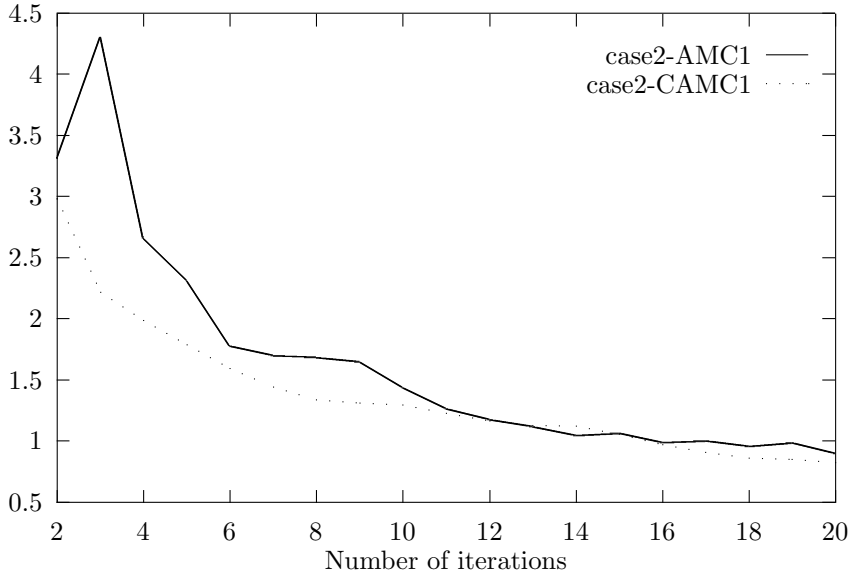


Figure 4.6: The estimated errors of case2-CAMC1 algorithm and case2-AMC1 algorithm for estimating $J_2$.

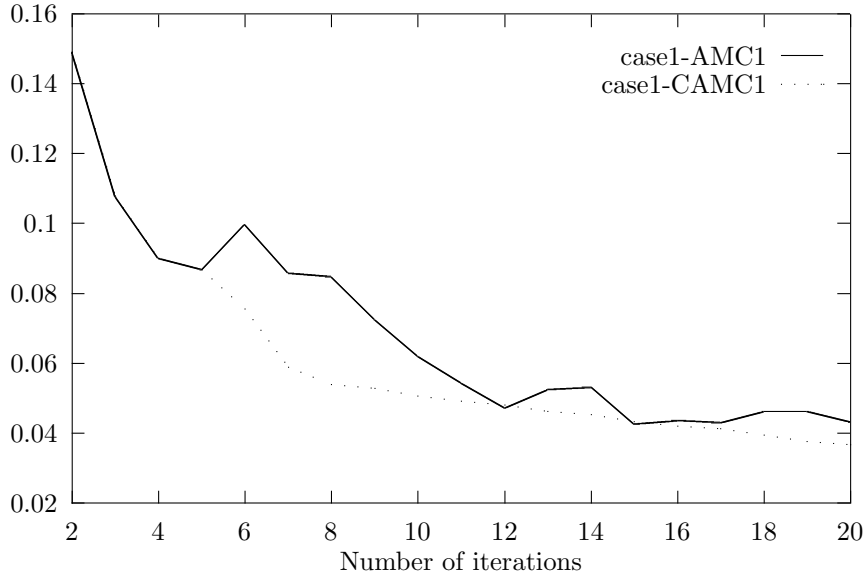


Figure 4.7: The estimated errors of case1-CAMC1 algorithm and case1-AMC1 algorithm for estimating $J_3$.

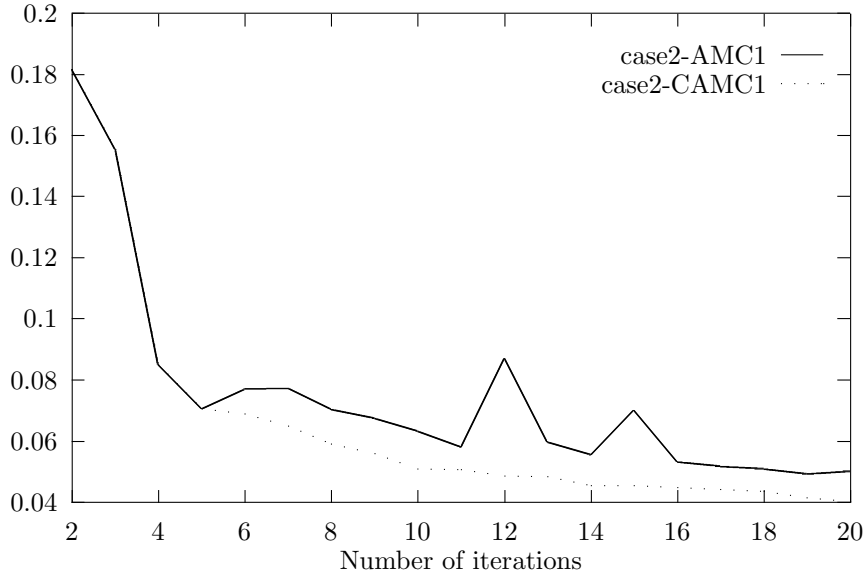


Figure 4.8: The estimated errors of case2-CAMC1 algorithm and case2-AMC1 algorithm for estimating $J_3$.

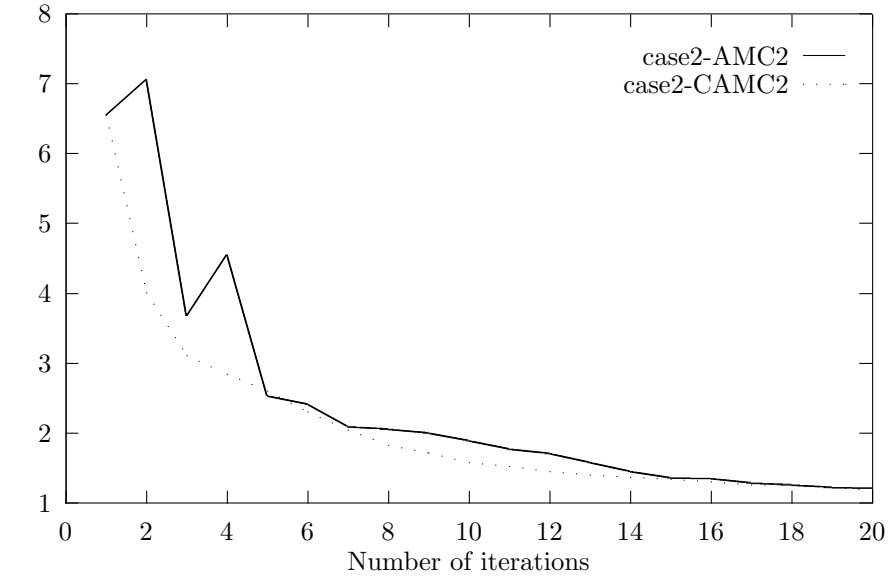


Figure 4.9: The estimated errors of case2-AMC2 and case2-CAMC2 algorithms using $N = 9,000$ in estimating $J_2$.

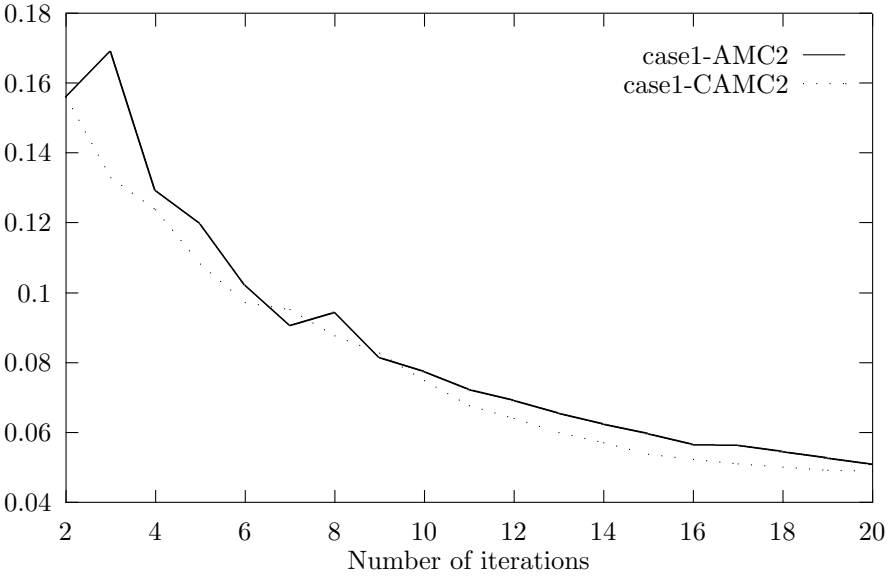


Figure 4.10: The estimated errors of case1-AMC2 and case1-CAMC2 algorithms using $N = 9,000$ in estimating $J_3$.

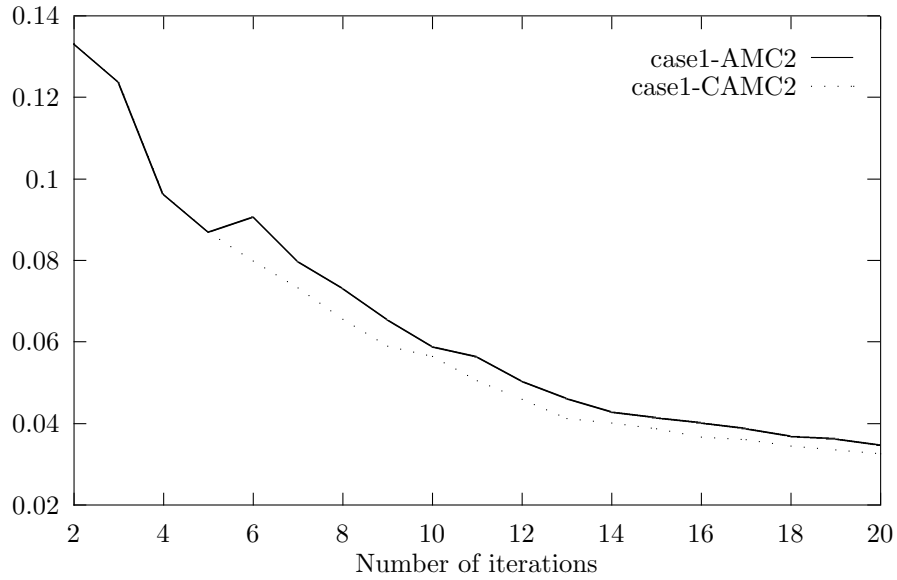


Figure 4.11: The estimated errors of case1-AMC2 and case1-CAMC2 algorithms using $N = 17,241$ in estimating $J_3$.

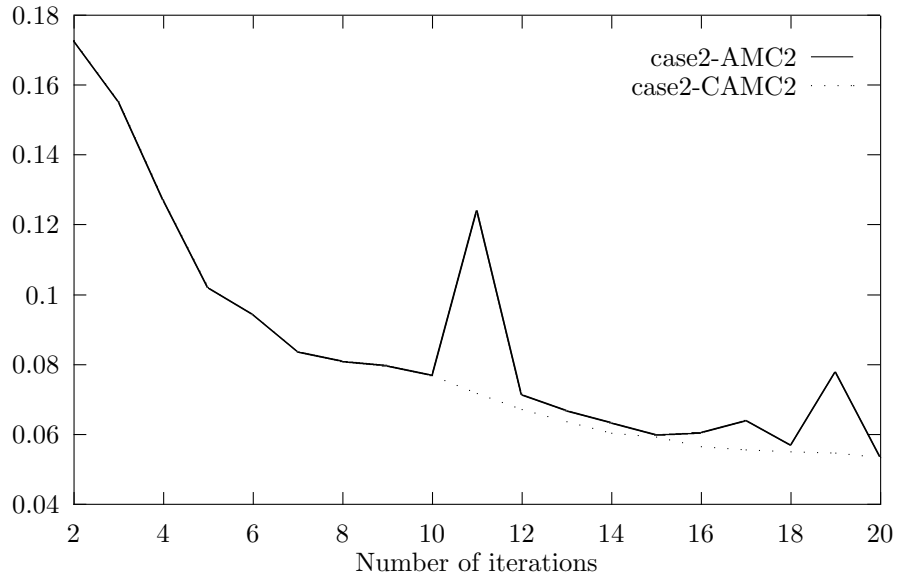


Figure 4.12: The estimated errors of case2-AMC2 and case2-CAMC2 algorithms using $N = 9,000$ in estimating $J_3$.

decreases as $s$ increases ($s$ is the number of divided coordinates). Therefore, CAMC is recommended for small values of $s$.

### 4.2.2 The efficiency of the corrector algorithm on the absolute error

Table 4.1 shows a three iterations from case1-AMC1 and case1-CAMC1 algorithms in estimating $J_2$ and $J_3$. We used $N = 5 \times 10^4$, the exact values for $J_2$ and $J_3$ are 103.8 and 3.244, respectively.

In estimating $J_2$, the three iterations represent a portion from Figure 4.5, where we see that case1-AMC1 algorithm moves to a worse estimate (bigger estimated error) in the 8-th iteration. When using case1-CAMC1 algorithm, the estimated error in the 8-th iteration becomes better with absolute error less than the absolute error of the usual adaptive algorithm.

In estimating $J_3$, the three iterations represent the first three iterations of Table 3.2 together with the first three iterations of case1-CAMC1. Also, it is observed that in the second iteration, not only the estimated error of case1-CAMC1 algorithm is better than the estimated error the usual adaptive algorithm; but also the absolute error of the corrector adaptive algorithm is better than the absolute error of the usual adaptive algorithm. This ensures that the division used in the usual adaptive algorithm in estimating $J_2$ and $J_3$ in iterations 7 and 1 respectively is not good; and that the new division performed using the proposed corrector algorithm is better.

| | | case1-AMC1 | | case1-CAMC1 | |
|---|---|---|---|---|---|
| $J$ | Iteration | $\hat{E}_i$ | Absolute error | $\hat{E}_i$ | Absolute error |
| $J_2$ | 6 | 1.372891 | 0.284175 | 1.372891 | 0.284175 |
| | **7** | **1.461784** | **2.441429** | **1.331482** | **0.944450** |
| | 8 | 1.326269 | 1.678988 | 1.242365 | 0.439903 |
| $J_3$ | 0 | 0.221240 | 0.107107 | 0.221240 | 0.107107 |
| | **1** | **0.247750** | **0.255195** | **0.133700** | **0.082885** |
| | 2 | 0.100097 | 0.090424 | 0.132100 | 0.020680 |

Table 4.1: The estimated and absolute errors of case1-AMC1 and case1-CAMC1 algorithms in three iterations.

# Chapter Five

# Conclusion

We have discussed the use of Monte Carlo methods in estimating solutions of multidimensional definite integrals over a hyper-rectangular region. The Mersenne Twister algorithm was used throughout this thesis for generating the pseudo-random numbers. This generator is believed to be one of the best pseudo-random number generators known today.

An adaptive Monte Carlo algorithm have been discussed in depth. This algorithm uses iteratively the basic idea of the Stratified Sampling using a global subdivision strategy. We have proposed a detailed algorithm which can be applied with an arbitrary $s$-division. This algorithm is implemented on a set of numerical multidimensional integrals, using a *one*-division and a *two*-division. Two types of division are used; in the first type, the region is divided into *equal* subregions, while in the second, it is divided into *random size* subregions. The adaptive Monte Carlo algorithm may move to worse estimates in few iterations, a corrector adaptive Monte Carlo integration algorithm has been proposed to prevent the adaptive algorithm from moving to worse estimates.

Extensive numerical examples are given for each type of Monte Carlo algorithms given in this thesis. From the numerical results, we conclude that the global subdivision strategy that is based on dividing the region into *equal* subregions is better than the strategy that is based on dividing the region into *random size* subregions. The *two*-division gave better results than the *one*-division in all cases. Moreover, the corrector adaptive Monte Carlo algorithm always gives better results when applied to the *one*-

division algorithm. However in the *two*-division, the corrector is not always necessary as the *two*-division algorithm behaves well. Therefore, the corrector adaptive algorithm is recommended for small values of $s$ (where $s$ is the number of divided coordinates). This is because the more division we use the better error estimates we get, so the corrector can give better results in a fewer subdivisions as the adaptive algorithm moves to worse estimates more.

# Bibliography


[1] Hammersley M. and D. Handscomb. Monte Carlo Methods. Chapman and Hall. London; 1965.

[2] Fishman G. Monte Carlo: Concepts, Algorithms and Applications. Springer. New York; 1996.

[3] Miguel A. Application to Risk Theory of a Monte Carlo Method. Mathematics and Economics 1998; 23: 71-83.

[4] Erich N. High Dimensional Integration. Advances in Computational Mathematics 2000; 12: 1-2.

[5] Woolfson M. and J. Pert. An Introduction to Computer Simulation. Oxford University press. New York; 1999.

[6] Karaivanova A. and I. Dimov. Error Analysis of an Adaptive Monte Carlo Method For Numerical Integration. Mathematics and Computers in Simulation 1998; 47: 201-213.

[7] Ross K. and J. Wang. Implementation of Monte Carlo Integration For the Analysis of Product-Form Queing Networks. Performance Evaluation 1997; 29: 273-292.

[8] Kahaner D., C. Moler and S. Nash. Numerical Methods and Software. Prentice-Hall. Englewood Cliffs; 1989.

[9] Rubinstein R. Simulation and the Monte Carlo Method. John Willy. New York; 1981.

[10] Lux I. and L. Koblinger. Monte Carlo Particle Transport Methods: Neutron and Photon Calculations. CRN Press. Florida; 1991

[11] Marokoff W. and R. Caflisch. Quasi-Monte Carlo Integration. Journal of Computational Physics 1995; 122: 218-230.

[12] Dahl L. An Adaptive Method For Evaluating Multidimensional Contingent Claims. Part I. International Journal of Theoretical and Applied Finance 2003; 6(4): 327-353.

[13] Schürer R. High-Dimensional Numerical Integration on Parallel Computers. Deploma thesis. University of Salzburg. Austria; 2001.

[14] Schürer R. A Comparison Between (Quasi-) Monte Carlo and Cubature Rule Based Methods For Solving High-dimensional Integration Problems. Mathematics and Computers in Simulation 2003; 62: 509-517.

[15] Kaneko T. and K. Tobimatsu. A New Monte Carlo Method of the Numerical Integration "Superposing Method". Nuclear Instruments and Methods in Physics Research 2003; A 502: 590-592.

[16] Matsumoto M. and T. Nishimura. Mersenne Twister: A 623-dimensionally Equidistributed Uniform Pseudorandom Number Generator. ACM Transactions on Modeling and Computer Simulation 1998; 8(1): 3-30.

[17] Lehmann E.L. and G. Casella. Theory of Point Estimation. Springer, New York; 1998.

[18] Press W., B. Flannevy , S. Tenkolsky and W. Vetterling. Numerical Recipes: The Art of Scientific Computing. Cambridge University Press. Cambridge; 1986

[19] Evans G. Practical Numerical Integration. John Wiley. England; 1993.

[20] Yakowitz S. Computational Probability and Simulation. Addison-Wesley. Massa; 1977.

**Website References:**

[21] Internet site: www.math.keio.ac.jp/∼matumoto/emt.html.

# Appendix A

# Generation of Random Variables

The MC procedure is based on developing random variables, which can be generated using uniform random numbers.

In this Appendix, we are concerned in generating random variables from different distributions. Firstly, we review two important methods for generating random numbers (uniform random numbers); namely, the "linear congruential" and the "Mersenne twister" generators. Then, we briefly describe some common techniques to transform the sequence of random numbers which are uniformly distributed in the interval [0,1], into a sequence distributed according to a specified probability density function. Finally, we describe the generalization of these methods into the multivariate case.

## A.1 Random number generators

One can define the MC method to be a technique that uses *random* or *pseudo-random numbers* for estimating solutions of a model, and the success of any MC application depends crucially on the quality of the PRN generator used. In this section, we are concerned with methods of generating random numbers on digital computers. *Random Numbers* (RN) are essentially independent random variables uniformly distributed over the unit interval $[0, 1]$, and the term "random number" is used instead of "uniform random number".

In the previous, manual methods were used for generating RNs, including such techniques as coin flipping, dice rolling and card shuffling; also physical devices were

used such as a roulette wheel, the random noise in an electronic circuit and the emission of $\gamma$-rays from radioactive nuclide. It was believed that only mechanical (or electronic) devices could yield "truly" RNs, but these methods were too slow for general use and the sequences generated by them could not be reproduced. Reproducible sequences are known as *Pseudo-Random Number* streams, and these are the ones that are almost exclusively used in simulation today. PRN streams are usually generated at run-time using a deterministic recurrence formula. We say that the PRNs generated by a specific method are "good" ones if they are uniformly distributed, statistically independent and reproducible, see Rubinstein [9] and Woolfson and Pert [5].

A good PRN generator should satisfy the following criteria:

- Good distribution. The points should be distributed according to what one would expect from a truly random distribution. Furthermore, a PRN generator should not introduce artificial correlations between successively generated points. There are several tests to measure the "good randomness" of the generated RNs such as the *die hard* test, the *spectral* test and the *k-distribution* test defined in Appendix (A.1.2), these tests are considered to be the strongest, see Matsumoto and Nishimura [16].

- Long period. Any PRN generator has a period, after which they begin to generate the same sequence of numbers over again.

- Repeatability. For testing and development, it may be necessary to repeat a calculation with exactly the same numbers as in the previous run. Most simulation experiments require the sequence of generated PRNs to be reproducible, in order to assist the debugging of computer programs.

- Portability. This means not only that the code should be portable (i.e., in a high level language like C, C++ or Fortran), but it should generate exactly the same sequence of numbers on different machines.

- Efficiency. The generation of the PRNs should not be too time-consuming. Almost all generators can be implemented in a reasonable efficient way.

Many generators of PRNs have been proposed, and we review here two methods, the Linear Congruential Generator, which is widely used for generating PRNs, and the Mersenne Twister generator, which is believed to be one of the best generators of PRNs known today.

### A.1.1 Linear congruential generators

In this section, we review the linear congruential method for generating PRNs that was designed specifically to satisfy as many of the requirements above as possible. The *Linear Congruential Generator* (LCG) is the most commonly used method for generating PRNs. LCG is based on a fundamental linear congruence relationship, and can deliver a sequence of nonnegative integers $\{X_i\}$, where

$$X_{i+1} = (aX_i + c) \mod m, \quad i = 1, 2, \ldots, n$$

where $m$ is a specified positive integer known as the *modulus*; $X_0$, $a$ and $c$ are specified nonnegative integers known as the *seed*, *multiplier*, and *increment*, respectively. The sequence $\{X_i\}$ takes values in the range $[0, m-1]$. The PRN is delivered as $R_i = X_i/m$ and so $R_i \in [0, 1)$. Clearly, such sequence will repeat itself in at most $m$ steps, for example, let $a = c = X_0 = 3, m = 5$ then the sequence obtained from the recursive formula:

$$X_{i+1} = (3X_i + 3) \mod 5$$

is $X_i = 3, 2, 4, 0, 3, 2, 4 \ldots$

As we know, the period of this generator is the smallest positive integer $\lambda$ for which $X_\lambda = X_0$. The sequence repeats itself after $\lambda$ numbers, so it is interesting to make the period as large as possible. The period cannot exceed $m$, so we want to choose $m$ as large as possible to ensure a sufficiently large sequence of distinct numbers. When the period $\lambda$ equals its maximum (i.e., $\lambda = m$), we say that the random number generator has a full period. Hull and Dobell (1962) showed that a full period can be obtained if and only if

1. $c$ is relatively prime to $m$.

2. $a \equiv (1 \mod g)$, $((a-1)$ is a multiple of $g)$ for every prime factor, $g$ of $m$.

3. $a \equiv (1 \mod 4)$, $((a-1)$ is a multiple of 4) if $m$ is a multiple of 4.

Good choices of the constants would be $a = 69069$, $c = 0$, and $m = 2^{32}$; or $a = 1812433253$, $c = 0$ and $m = 2^{32}$. IBM used in the sixties and seventies the values $a = 65539$, $c = 0$ and $m = 2^{31}$ as well as $a = 16807$, $c = 0$ and $m = 2^{31} - 1$.
See Rubinstein [9] and Woolfson and Pert [5].

### A.1.2 Mersenne twister generator

Matsumoto and Nishimura [16] have proposed a PRN generator called the *Mersenne Twister*. The algorithm provides a far longer period than any previously existing generator, where it has a period of $2^{19937} - 1$. The MT19937 is widely believed to be one of the best PRN generators known today, it has passed several stringent tests, including the *Die hard*, the *load* and the *k-distribution* tests.

Consider the pseudo-random real number $x_i$ in [0,1]-interval. Put $P$ points in the $k$-dimensional unit cube with coordinate $(x_i, x_{i+1}, \ldots, x_{i+k-1})$, $i = 0, 1, \ldots, P-1$. Then divide each [0,1] axis into $2^v$ pieces. Thus, the unit cube has been partitioned into $2^{kv}$ small cubes. Then the sequence is *k-distributed to v-bit accuracy* (or *k-dimensionally equidistributed to v-bit of accuracy*) if each cube contains the same number of points except for the cube at the origin which contains one less.

The MT19937 attains a far longer $k$-distribution than any previously existing generators, where it has 623-dimensionally equidistributed up to 32-bit accuracy.

A comparison between the LCG "rand" (the standard random number generator of the C-library) and the MT19937 is given in Table A.1, see Matsumoto and Nishimura [16].

MT19937 is believed to be ideal for MC simulation, see Matsumoto and Nishimura [16].

The MT19937 algorithm generates a sequence of *word vectors* (a $w$-dimensional row vectors over the two element field $\mathbb{F}_2 = \{0, 1\}$), which are considered to be uniform

| | rand | MT19937 |
|---|---|---|
| CPU-time (sec.) | 9.64 | 10.18 |
| working area | 1 | 624 |
| period | $\sim 2^{31}$ | $2^{19937}-1$ |

Table A.1: CPU-time for $10^7$ generations and the working area (using a Sun Workstation).

pseudo-random integers between 0 and $2^w - 1$. Dividing by $2^w - 1$, then each word vector is transformed into a real number in [0,1].

The algorithm is based on the following cleverly selected linear recurrence:

$$\mathbf{x}_{k+n} = \mathbf{x}_{k+m} \oplus (\mathbf{x}_k^u | \mathbf{x}_{k+1}^l)A, \quad k = 0, 1, \ldots \tag{A.1.1}$$

(the bold letters denote word vectors), $n$ is the degree of the recurrence, $m$ is an integer with $1 \le m \le n$, $A$ is a $w \times w$ matrix with entries in $\mathbb{F}_2$, $\mathbf{x}_k^u$ means "the upper $w - r$ bits" of $\mathbf{x}_k$, and $\mathbf{x}_{k+1}^l$ "the lower $r$ bits" of $\mathbf{x}_{k+1}$ where $r$ is an integer with $0 \le r \le w-1$. Thus, if $\mathbf{x} = (x_{w-1}, x_{w-2}, \ldots, x_0)$, then $\mathbf{x}^u$ is the $w - r$ bits vector $(x_{w-1}, \ldots, x_r)$ and $\mathbf{x}^l$ is the $r$ bits vector $(x_{r-1}, \ldots, x_0)$. $(\mathbf{x}_k^u | \mathbf{x}_{k+1}^l)$ is the concatenation; namely, this is a word vector obtained by concatenating the upper $w-r$ bits of $\mathbf{x}_k$ and the lower $r$ bits of $\mathbf{x}_{k+1}$ in this order. Finally, the $\oplus$ is the bitwise addition modulo two.

$\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{n-1}$ are given as initial seeds. Then the generator generates $\mathbf{x}_n$ by the recurrence equation (A.1.1) with $k = 0$. By putting $k = 1, 2, \ldots$, the generator determines $\mathbf{x}_{n+1}, \mathbf{x}_{n+2}, \ldots$.

The matrix $A$ is chosen so that the multiplication by $A$ is very fast,

$$A = \begin{pmatrix} & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \\ a_{w-1} & a_{w-2} & \ldots & \ldots & a_0 \end{pmatrix}$$

then the calculation of $\mathbf{x}A$ can be done using only bit operations:

$$\mathbf{x}A = \begin{cases} \text{shiftright}(\mathbf{x}) & \text{if } x_0 = 0 \\ \text{shiftright}(\mathbf{x}) \oplus \mathbf{a} & \text{if } x_0 = 1 \end{cases}$$

where $\mathbf{a} = (a_{w-1}, a_{w-2}, \ldots, a_0)$, $\mathbf{x} = (x_{w-1}, x_{w-2}, \ldots, x_0)$. Also, $\mathbf{x}_k^u$ and $\mathbf{x}_{k+1}^l$ on the recurrence equation (A.1.1) can be calculated with bitwise AND operation. Thus,

the calculation of the recurrence equation (A.1.1) is realized with bitshift, bitwise *exclusive*-OR, bitwise OR, and bitwise AND operations.

For improving the $k$-distribution to $v$-bit accuracy, the following transformation is done

$$\begin{aligned}\mathbf{y} &= \mathbf{x} \oplus (\mathbf{x} >> u)\\ \mathbf{y} &= \mathbf{y} \oplus ((\mathbf{y} << s) \text{ AND } \mathbf{b})\\ \mathbf{y} &= \mathbf{y} \oplus ((\mathbf{y} << t) \text{ AND } \mathbf{c})\\ \mathbf{z} &= \mathbf{y} \oplus (\mathbf{y} >> l)\end{aligned}$$

where $l, s, t,$ and $u$ are integers, $\mathbf{b}$ and $\mathbf{c}$ are suitable bitmasks of word size, and $(\mathbf{x} >> u)$ denotes the $u$-bit shiftright ($(\mathbf{x} << u)$ the $u$-bit shiftleft).

We now state the MT19937 algorithm, see Matsumoto and Nishimura [16].

**MT19937 algorithm:**

Let $\mathbf{x}[0 : n-1]$ be an array of $n$ unsigned integers of word size, $i$ be an integer variable, and $\mathbf{u}$, $\mathbf{ll}$, $\mathbf{a}$ be unsigned constant integers of word size.

**Parameters:** $w$ (word size), $n$ (degree of recursion), $m$ (middle term), $r$ (separation point of one word); $u, s, t, l$ (integers); $\mathbf{b}$, $\mathbf{c}$ (vector parameters); $\mathbf{u}$, $\mathbf{ll}$, $\mathbf{a}$ (unsigned constants)

**Step 0:** $\mathbf{u} \leftarrow \underbrace{1\ldots1}_{w-r}\underbrace{0\ldots0}_{r}$ ;(bitmask for upper $w-r$ bits)

$\mathbf{ll} \leftarrow \underbrace{0\ldots0}_{w-r}\underbrace{1\ldots1}_{r}$ ;(bitmask for lower $r$ bits)

$\mathbf{a} \leftarrow a_{w-1}a_{w-2}\ldots a_1a_0$ ;(the last row of the matrix $A$)

**Step 1:** $i \leftarrow 0$

$\mathbf{x}[0], \mathbf{x}[1], \ldots, \mathbf{x}[n-1] \leftarrow$"any non-zero initial values"

**Step 2:** $\mathbf{y} \leftarrow$ ($\mathbf{x}$[i] AND $\mathbf{u}$) OR ($\mathbf{x}$[(i+1) mod n] AND $\mathbf{ll}$) ;(computing $(\mathbf{x}_i^u|\mathbf{x}_{i+1}^l)$)

**Step 3:** $\mathbf{x}[i] \leftarrow \mathbf{x}[(i+m) \mod n]$ XOR $(\mathbf{y} >> 1)$

$$\text{XOR}\begin{cases}\mathbf{0} & \text{if the least significant bit of } \mathbf{y}{=}0\\ \mathbf{a} & \text{if the least significant bit of } \mathbf{y}{=}1\end{cases}$$ ;(multiplying $A$)

**Step 4:** $\mathbf{y} \leftarrow \mathbf{x}[i]$
$\mathbf{y} \leftarrow \mathbf{y}$ XOR $(\mathbf{y} >> u)$ ;(shiftright $\mathbf{y}$ by $u$ bits and add to $\mathbf{y}$)
$\mathbf{y} \leftarrow \mathbf{y}$ XOR $((\mathbf{y} << s)$ AND $\mathbf{b})$
$\mathbf{y} \leftarrow \mathbf{y}$ XOR $((\mathbf{y} << t)$ AND $\mathbf{c})$
$\mathbf{y} \leftarrow \mathbf{y}$ XOR $(\mathbf{y} >> l)$
output $\mathbf{y}$

**Step 5:** $i \leftarrow (i+1) \mod n$

**Step 6:** Goto Step 2.

Matsumoto and Nishimura [16] have proposed the following values for the parameters in the previous algorithm: $w = 32$, $n = 624$, $m = 397$, $r = 31$, $u = 11$, $s = 7$, $t = 15$, $l = 18$, $\mathbf{a}$ =9908B0DF, $\mathbf{b}$ =9D2C5680, $\mathbf{c}$ =EFC60000. They also gave a portable C-code for their generator, MT19937 is a reproducible PRN generator, we saw that it has a very large period, and it has 623-dimensionally equidistributed up to 32-bit accuracy. Therefore, the MT19937 is widely believed to be one of the best generators of PRNs known today.

## A.2 The generation of general continuous random variables

A MC method is a procedure that involves the use of statistical sampling procedures to estimate the solution of a mathematical or physical problem, so we need to generate random variables with a particular density function.

In this section, we consider some procedures for generating random variables from different distributions. We review the *inverse transformation* and the *acceptance rejection* methods for generating random variables from a specified p.d.f. in one variate case. Then we describe a generalization of the inverse transformation method into the multivariate case.

### A.2.1 The inverse transformation method

Let $X$ be a random variable with *cumulative probability distribution function* (CDF) $F_X(x)$. Since $F_X(x)$ is nondecreasing function, so the inverse function $F_X^{-1}(y)$ may be defined for any value of $y$ between 0 and 1 as the following definition

$$F_X^{-1}(y) = \inf\{x : F_X(x) \geq y\}, \quad 0 \leq y \leq 1$$

It can be shown that $X = F_X^{-1}(U)$ where $U \sim U(0,1)$, has CDF $F_X(x)$.
The proof is straight forward:

$$\begin{aligned} p(X \leq x) &= p(F_X^{-1}(U) \leq x) \\ &= p(U \leq F_X(x)) \\ &= F_X(x) \end{aligned}$$

The following algorithm describes the inverse transformation method

**Algorithm**

**Step 1:** Generate $U \sim U(0,1)$.

**Step 2:** Return $X = F_X^{-1}(U)$.

For example, suppose we need to generate a uniform random number in the interval $[a, b]$ using the inverse transformation method. Then, the CDF of the uniform p.d.f. is given by

$$F(x) = \frac{x-a}{b-a}, \quad \text{for } a \leq x \leq b.$$

And

$$u = F(x) \Rightarrow x = (b-a)u + a$$

So, we generate $u \sim U(0,1)$, then $x = (b-a)u + a$ is uniformly distributed in the interval $[a, b]$.

### A.2.2 The acceptance rejection method

Suppose the CDF ($F_X(x)$) is *not continuous* or it is *not one to one*, then it is not easy to use the computer for generating random variables from the p.d.f. $f_X(x)$ using the inverse transformation method.

In this section, we give a simple illustration of the principle of the acceptance rejection method which has been developed by Von Neumann (1951), see Woolfson and Pert [5]. The proof of the following theorem is given in Rubinstein [9].

**Theorem A.2.1.** *let $X$ be a random variate distributed with the p.d.f. $f_X(x)$ represented as*

$$f_X(x) = Ch(x)g(x)$$

*where $C \geq 1$, $h(x)$ is a p.d.f. and $0 \leq g(x) \leq 1$. Let $U$ and $Y$ be distributed $U(0,1)$ and $h(y)$, respectively; then*

$$f_Y(x/U \leq g(Y)) = f_X(x)$$

So, to generate a random variable distributed with p.d.f. $f_X(x) = Ch(x)g(x)$; firstly, we generate $Y$ according to the distribution $h(y)$ (a p.d.f.). Then, we generate a RN $U$, which is uniformly distributed in [0,1] and check $U \leq g(Y)$ $(0 \leq g(x) \leq 1)$. If this is the case, then we accept $Y$ as a random variable from $f_X(x)$. Otherwise, we reject and we start again. The following algorithm describes the acceptance rejection algorithm.

**Acceptance rejection algorithm 1**

**Step 0:** Generate $U \sim U(0,1)$.

**Step 1:** Generate $Y$ from the p.d.f. $h(y)$.

**Step 2:** If $U \leq g(Y)$ Return $Y$ (as a variate generated from $f_X(x)$) Else Goto Step 0.

There exist an infinite number of ways to choose $h(x)$, and the simplest way is to choose $h(x) = \frac{1}{b-a}$ (the uniform distribution) and $g(x) = \frac{f_X(x)}{M} \quad \forall\ a \leq x \leq b$,

$C = M(b-a)$, where $M = \max\{f(x), x \in [a,b]\}$. This algorithm can be described as follows:

**Acceptance rejection algorithm 2**

**Step 0:** Generate $U_i \sim U(0,1)$, $i = 1, 2$.

**Step 1:** Calculate $Y = (b-a)U_1 + a$.

**Step 2:** If $MU_2 \leq f_X(Y)$ then deliver $Y$ as a variate generated from $f_X(x)$ Else Goto Step 0.

## A.3 Multivariate case

A point of power for the MC methods in multidimensional problems is the use of random variables which are *independent*. So, in this section we are concerned in generating a random vector $X = (X_1, X_2, \ldots, X_n)$ from a given CDF $F_X(x)$, where $X_1, X_2, \ldots, X_n$ are independent random variable.

### A.3.1 Inverse transformation method

We consider the joint p.d.f.

$$f_{X_1,X_2,\ldots,X_n}(x_1, \ldots, x_n) = \prod_{i=1}^{n} f_i(x_i)$$

where $f_i(x_i)$ is the marginal p.d.f. of the random variable $X_i$. To generate the random vector $X = (X_1, X_2, \ldots, X_n)$ from the CDF $F_X(x)$, we can apply the inverse transformation method,

$$X_i = F_i^{-1}(U_i), \quad U_i \sim U(0,1), i = 1, 2, \ldots, n$$

for each variable separately.

**Example A.1.** Consider $X_1, X_2, \ldots, X_n$ as i.i.d. from the uniform distribution on $[0, 1]$, then the p.d.f. for each variable $X_i$ is

$$f_i(x_i) = \begin{cases} \frac{1}{b_i - a_i} & a_i \leq x_i \leq b_i, \quad i = 1, \ldots, n \\ 0 & \text{otherwise} \end{cases}$$

To generate the random vector $X = (X_1, X_2, \ldots, X_n)$ with the p.d.f.

$$f_{X_1,\ldots,X_n}(x_1, \ldots, x_n) = \begin{cases} \frac{1}{\prod_{i=1}^{n}(b_i - a_i)} & (x_1, \ldots, x_n) \in D \\ 0 & \text{otherwise} \end{cases}$$

where $D = \{(x_1, \ldots, x_n), a_i \leq x_i \leq b_i, i = 1, \ldots, n\}$; we apply the Inverse Transformation formula,

$$X_i = a_i + (b_i - a_i)u_i, \quad i = 1, 2, \ldots, n$$

where $u_i \sim U(a_i, b_i) \quad i = 1, \ldots, n$.