# Applying Markov Chains to Monte Carlo Integration

Leonardo Ferreira Guilhoto

**Abstract**

This paper surveys the Markov Chain Monte Carlo (MCMC) method of numeric integration, with special focus on importance sampling. In the first part of the paper, the Monte Carlo Integration method is explained, along with a brief discussion about variance and how to reduce it. In the second part, Markov Chains are introduced along with some basic definitions and theorems. The third part links these two concepts using the Metropolis-Hastings algorithm. The fourth and final part implements this method to evaluate the integral $\int_0^1 \int_0^1 e^{sin(xy)} dxdy$ in a C code, and examines the efficiency of MCMC integration. This section also tests and confirms that the Ergodic Theorem for Markov Chains is useful for the purpose of numeric integration.

## Contents

# 1 Monte Carlo Integration

Monte Carlo is a group of techniques used to numerically approximate the value of computations that are either very difficult or impossible to be solved analytically. The idea behind this method is to sample randomly according to some probability distribution and then use this information to estimate the value of the desired calculation. These techniques rely on the Law of Large Numbers to confirm that a random sampling indeed makes it so that the approximation converges almost surely to the actual value desired. This section will examine Monte Carlo Integration first by using a uniform probability distribution, and then any probability distribution, and discuss which ones make it so that the estimation converges more quickly.

## 1.1 The Monte Carlo Estimator

The average value of a function, $f$, over a multidimensional region, $V \subseteq \mathbb{R}^d$, can be easily obtained by:

$$\langle f \rangle = \frac{1}{vol(V)} \int_V f$$

Therefore, it follows that, if the average value of the function is known, the integral can be calculated by:

$$\int_V f = vol(V)\langle f \rangle$$

This is precisely the idea behind Monte Carlo Integration, which uses random sampling over the integration domain in order to estimate the average value of the function over that region.

**Definition 1.1.** the Monte Carlo Estimator of a function, using $N$ data points, is given by:

$$\langle F^N \rangle = \frac{vol(V)}{N} \sum_{i=1}^{N} f(x_i) \tag{1}$$

Where each $x_i$ is an independent data point sampled from a uniform distribution.

The Strong Law of Large Numbers then tells us that:

$$P(\lim_{N\to\infty} \langle F^N \rangle = \int_V f) = 1 \tag{2}$$

Which confirms that this method works for computing integrals as the number of points used in the sample increases.

## 1.2 Expected Value and Variance

As seen above, the Law of Large Numbers tells us that our Estimator converges to the right value as $N$ tends to infinity. However, in real life it is impossible to run a program with an infinite number of computations, so in order to show that the method is actually useful we must study how fast the Estimator converges to the value of the integral.

The first concept to help us understand convergence is the **expected value** of a function over some region. As the name suggests, this is a number that indicates, what one should *expect* to obtain as the mean of their data points as the number of points collected increases.

**Definition 1.2.** The expected value of a function $f : D \to \mathbb{R}$ with respect to a measure given by a probability density function, $\lambda : D \to \mathbb{R}_{\geq 0}$, is:

$$E_\lambda(f) = \int_D f\lambda \tag{3}$$

Or, if $D$ is finite:

$$E_\lambda(f) = \sum_{x\in D} f(x)\lambda(x) \tag{4}$$

This definition, allied with the Law of Large Numbers, provides us with exactly what we hope to get from this definition, that, if we sample according to $\lambda$:

$$P(\lim_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n} f(x_i) = E_\lambda(f)) = 1$$

While the expected value tells us that the estimator converges to the right value, the concept that actually helps us understand *how fast* the sample mean converges to expected value is the **variance**, which describes how far away our individual data points are from the expected mean.

**Definition 1.3.** the variance of a function $f : D \to \mathbb{R}$ with respect to a measure given by a probability density function, $\lambda : D \to \mathbb{R}_{\geq 0}$, is the expected value of $(f - E_\lambda(f))^2$, *i.e.*:

$$\mathrm{Var}_\lambda(f) = E_\lambda([f - E_\lambda(f)]^2) \tag{5}$$

The square in this definition guarantees that this quantity will always be non-negative, and so the only way for the variance to be 0 is if $f = E_\lambda(f)$ with probability 1. Otherwise, the variance will always be of positive value.

The variance can also be defined for a collection of random points, $(x_1, ...x_n)$:

$$\mathrm{Var}(x_1, ...x_n) = \frac{1}{n-1}\sum_{i=1}^{n}[x_i - \frac{1}{n}\sum_{j=1}^{n} x_j]^2 \tag{6}$$

In this case, the sum is divided by $n-1$ instead of $n$, since we are taking the variance over a sample without knowing the actual mean of the entire population. This change (called "Bessel's correction") is employed because the value used for the expected value, $(\frac{1}{n}\sum_{i=1}^{n} x_i)$, is only an estimate of the population mean, and is biased with relation to the $x_i$'s sampled. This bias makes it so that dividing the sum by $n$ gets us a value that is off by a factor of $\frac{n-1}{n}$. Therefore, dividing the sum by $n-1$ instead makes up for this bias, and is called the unbiased sample variance.

The important consequence of knowing the concept of variance is that it tells us, on average, how fast the mean of our data points will converge to the expected value. Take for example the extreme case in which all data points are the same: in this case the variance will be 0, which agrees with the fact that the mean is exactly the same, no matter where you cut the sequence off. However, if the variance is big, the point at which we stop collecting samples will have a greater impact on how close the estimator is to the actual expected value. Of course, as we take more and more points the estimator should converge to the expected value regardless, but if we want to make our method efficient, decreasing the variance is a very valid choice, as this enables us to get a more precise estimate while collecting the same number of points.

## 1.3 Importance Sampling

In section **1.1** the Monte Carlo Estimator was described using a uniform probability distribution over the integration domain (*i.e.*: all points have the same likelihood of being sampled). However, it is possible to generalize this method to work with any probability distribution. Since integration usually happens over an infinite domain, and we want to be able to sample from all points within it, the actual probability of drawing any specific point is, in practice, 0. In oder to work with probabilities in such situations, we use a probability density function (pdf).

**Definition 1.4.** a probability density function (pdf) over a domain $D \subseteq \mathbb{R}^d$ is a function $\lambda : D \to \mathbb{R}_{\geq 0}$ st $\int_D \lambda = 1$

Such functions have the property that the probability of drawing a point within some region $C \subseteq D$ is $\int_C \lambda$. This is why we must require that $\int_D \lambda = 1$; because the probability of drawing a point within the entire domain is 1.

Our Monte Carlo estimator, then, can also be used while sampling from a non-uniform probability density function, $\lambda$, with a few alterations (this is, in fact, a more general version of the previous definition):

$$\langle F_\lambda^N \rangle = \frac{1}{N}\sum_{i=1}^{N} \frac{f(x_i)}{\lambda(x_i)} \tag{7}$$

Where all $x_i$'s are independent sample points, obtained according to $\lambda$.

**Lemma 1.1.** *The expected value of the above estimator is* $\int_D f$.

*Proof.*

$$\begin{aligned} E_\lambda(\langle F_\lambda^N \rangle) &= E_\lambda(\frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{\lambda(x_i)}) \\ &= \frac{1}{N}\sum_{i=1}^{N} E_\lambda(\frac{f(x_i)}{\lambda(x_i)}) \\ &= \frac{N}{N}\int_D [\frac{f(x_i)}{\lambda(x_i)} * \lambda(x_i)]dx_i \\ &= \int_D f \end{aligned}$$

□

The variance of the estimator can then be calculated by:

$$\mathrm{Var}(\langle F_\lambda^N \rangle) = \mathrm{Var}(\frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{\lambda(x_i)}) = \frac{1}{N^2}\sum_{i=1}^{N}\mathrm{Var}(\frac{f(x_i)}{\lambda(x_i)}) = \frac{N}{N^2}\mathrm{Var}(\frac{f}{\lambda}) = \frac{1}{N}\mathrm{Var}(\frac{f}{\lambda})$$

A very important fact about the math above is that at no moment did we specify the number of dimensions involved in the integration. This is probably the main advantage of Monte Carlo Integration over most other methods of numerical integration: its variance is proportional to $\frac{1}{N}$ regardless of the dimension of the space we are integrating over, while many other methods have its variance growing exponentially with respect to dimension.

This result can then be exploited in order to decrease the variance of the estimator by choosing a pdf $\lambda$ that minimizes the constant $\mathrm{Var}(\frac{f}{\lambda})$; this is precisely the idea behind importance sampling. A reasonable conclusion to jump to, then, is to have $\lambda \propto f$, since this would mean that $(\frac{f}{\lambda})$ is constant and, therefore, that the variance is 0. The problem with this method, however, is that it requires that you know the value of the integral of $f$ over $D$ beforehand, thus defeating the purpose of a numerical integration. This is because of the requirement that $\int_D \lambda$ be equal to 1. Therefore, in order for $\lambda$ to be proportional to $f$, you would need to have $\lambda = \frac{f}{\int_D f}$.

The method of importance sampling, however, is still valid in many situations! The trick that makes this method useful is choosing a pdf that *resembles* $f$ and is easy to be integrated analytically. This way, one can get the advantage of the reduced variance without needing to previously know the value of the integral.

# 2 Markov Chains

The method described in the previous section, despite being very useful, faces a difficult problem relating to the way in which computers sample from non-uniform probability distributions. To put it bluntly: computers are generally good at sampling from uniform distributions, not so much for complicated ones. There are plenty of efficient ways to create uniform distributions in computers, but simulating non-uniform distributions from these quick functions requires a lot of computational power. This section will describe a mathematical object called "Markov Chain", which will help us to efficiently sample from any distribution we desire.

## 2.1 Definitions

Before we start using Markov Chains for sampling, we must understand what a Markov Chain *is*, and some consequences of these initial definitions.

### 2.1.1 Markov Chain

**Definition 2.1.** a Markov Chain with finite state space $\Omega = \{X_1, ...X_{|\Omega|}\}$ and with transition matrix $P \in M_{|\Omega|}([0,1])$ is a sequence of random variables $(x_1, x_2, ...)$ such that for all $i \in \mathbb{N}$, $x_i \in \Omega$. This

model also implies that the probability that a given entry $x_i$ is at state $X_j$ (*i.e.*: $x_i = X_j$) depends only on the state of the previous entry in the sequence (*i.e.*: only on the $X_{j'}$ such that $x_{i-1} = X_{j'}$), and is equal to $P(X_{j'}, X_j)$[1] $= P_{j',j}$. The property that the next element in the sequence depends only on the one that immediately precedes it is called the Markov Property.

A classic example of a Markov Chain is a jumping frog: suppose a frog jumps between two rocks in a pond; take one of them as rock number one ($R_1$), and the other as rock number 2 ($R_2$). Whenever the frog is at rock number 1, there is a 50% chance that it will jump to rock number two, and 50% chance that it will decide to stay at rock number 1 for a little while longer (*i.e.*: $P(R_1, R_1) = 0.5$ and $P(R_1, R_2) = 0.5$). When the frog is at rock number two, however, it might notice a possible threat nearby, so it will jump to rock number 1 with 70% chance, and stay there for some more time with 30% chance (*i.e.*: $P(R_2, R_1) = 0.7$ and $P(R_2, R_2) = 0.3$). This system describes a Markov Chain, as the positions of the frog are determined only by the rock it is at the moment, and ignore any past memory (the frog even forgets about the possible threat at rock number 2!). In this example, the state space is $\Omega = \{R_1, R_2\}$, and the transition matrix is:

$$P = \begin{bmatrix} 0.5 & 0.5 \\ 0.7 & 0.3 \end{bmatrix}$$

As it can be noted, not only must P be a square matrix, but all entries in P must be non-negative and the sum in each row must be equal to 1; these three requirements define what is called a stochastic matrix.

### 2.1.2 Irreducible Chains

Saying that a chain is irreducible means that any state in $\Omega$ is accessible from any other state in $\Omega$. The word "irreducible" is used to mean that no state can be removed from $\Omega$ if one wishes to accurately depict the phenomena described by the rules of transition of the chain. The concept can be formalized in the following manner:

**Definition 2.2.** a Markov Chain is called irreducible if for all $x \in \Omega$ and for all $y \in \Omega$ there exists some $t \in \mathbb{N}$ such that $P^t(x, y) > 0$.

The frog chain described in the previous section is an irreducible chain. However, if a third rock was included to the state space, one which is too far to be reached from rocks 1 and 2, then the chain would not be irreducible, as having the frog on either one of the original rocks would mean that rock number 3 would never be accessible.

### 2.1.3 Periodic and Aperiodic Chains

Let $x_0$ be the initial state of a Markov Chain, and let $x_t$ be the state obtained by applying $t$ steps of the Markov Chain to $x_0$. The period, $N$, of a state $X \in \Omega$ is a number that indicates that, if $X$ is the initial state of the chain (*i.e.*: if $x_0 = X$) and $x_n$ is such that $x_n = X$ (*i.e.*: $x_0 = x_n = X$), then $n$ is a multiple of $N$. This can be formally expressed by the following definition:

**Definition 2.3.** the period of a state $x \in \Omega$ is the greatest common divisor of the set $\tau(x) := \{t \geq 1 | P^t(x, x) > 0\}$ *i.e.*: $\text{period}(x) = \gcd(\tau(x))$

**Lemma 2.1.** *If a Markov Chain is irreducible and $x, y \in \Omega$, then period(x) = period(y).*

*Proof.* Since the chain is irreducible, for all $x, y \in \Omega$, there exist $a, b \in \mathbb{N}$ such that $P^a(x, y) > 0$ and $P^b(y, x) > 0$. Take $c = a + b$, then $P^c(x, x) = P^{a+b}(x, x) \geq P^a(x, y) * P^b(y, x) > 0 \Rightarrow c \in \tau(x)$. By symmetry, we can also conclude that $c \in \tau(y)$. Therefore, $c \in [\tau(x) \cap \tau(y)]$.

Now notice that for all $n \in \tau(x)$ we have that $(n + c) \in \tau(y)$, since $P^{n+c}(y, y) = P^{b+n+a}(y, y) \geq P^b(y, x) * P^n(x, x) * P^a(x, y) > 0$. This means that $\tau(x) + c \subset \tau(y) \Rightarrow \tau(x) \subset \tau(y) - c \Rightarrow \gcd(\tau(y)) \leq \gcd(\tau(x))$.

Again, by symmetry, we may also conclude that $\gcd(x) \leq \gcd(y)$. Therefore, $\gcd(\tau(x)) = \gcd(\tau(y))$ □

[1] This notation can be expanded in the following manner: $\forall x, y \in \Omega$ and $\forall t \in \mathbb{N}$, $P^t(x, y)$ denotes the probability that $y$ is the $(i + t)^{th}$ term in the sequence, given that $x$ is the $i^{th}$ term.

As a consequence of this lemma, we may classify any irreducible Markov Chain by the period which is common to all elements in its state space.

**Definition 2.4.** for an irreducible Markov Chain, we say it is periodic of period $T$ if its elements have period $T \geq 2$. Alternatively, if all elements have period 1, we say the chain is aperiodic.

In summary, suppose a Markov Chain is initiated at state $X \in \Omega$, and returns to that same state after $N$ steps (*i.e.*: $x_0 = x_N = X$). Then if the chain is aperiodic, no new information is given to us about this number $N$; if the chain is periodic, however, we may deduce that $N$ is a multiple of the chain's period.

### 2.1.4 Stationary Distribution

Perhaps one of the most useful property of Markov Chains and transition matrices is that they can be used as more than just probability tables for reference. If instead of knowing for sure in which state a chain currently is, you have a suspicion about each state, described by a probability distribution, you may multiply this distribution by the transition matrix and obtain the probability distribution of the chain after one transition. In order to understand this property, we use the definition:

**Definition 2.5.** a discrete probability distribution on a state space $\Omega$ is a vector $\lambda = (\lambda_1, ..., \lambda_{|\Omega|}) \in \mathbb{R}^{|\Omega|}$ such that for all $i$ we have $\lambda_i \geq 0$, and $\sum_{i=1}^{|\Omega|} \lambda_i = 1$.

Therefore, suppose your suspicions about the chain are sumarized by the probability distribution $\lambda$, then the probability distribution, $\mu$, of the chain after one transition is:

$$\mu = \lambda P \tag{8}$$

Take for example the frog presented in section **2.1.1**, and suppose you think there is a 40% chance that the frog is at rock number 1, and 60% chance at rock number 2; that is, suppose you have the probability distribution $\lambda = (0.6, 0.4)$. Then, if one transition of the chain happens, your suspicions will be updated to the probability distribution $\mu$ given by:

$$\mu = \lambda P = \begin{bmatrix} 0.6 & 0.4 \end{bmatrix} \begin{bmatrix} 0.5 & 0.5 \\ 0.7 & 0.3 \end{bmatrix} = \begin{bmatrix} 0.58 & 0.42 \end{bmatrix}$$

Now that this crucial fact has been stated, an important type of distribution will be studied: stationary distributions, which are distributions which do not change after the transition matrix is applied to it.

**Definition 2.6.** a probability distribution, $\pi$, is called stationary in relation to a transition matrix, P, if $\pi = \pi P$. i.e.: if $\pi$ is a left eigenvector of P, with eigenvalue 1.

Taking advantage of the same example as before, one could check that the distribution $\pi = (\frac{7}{12}, \frac{5}{12})$ is stationary in relation to the frog Markov Chain.

$$\pi P = \begin{bmatrix} \frac{7}{12} & \frac{5}{12} \end{bmatrix} \begin{bmatrix} 0.5 & 0.5 \\ 0.7 & 0.3 \end{bmatrix} = \begin{bmatrix} \frac{7}{12} & \frac{5}{12} \end{bmatrix} = \pi$$

## 2.2 Detailed Balance Equations

A very important theorem for MCMC is one involving what are called "detailed balance equations". The algorithm explored in section **3.1** heavily relies on the following result.

**Theorem 2.1.** *Given a state space $\Omega$, and a transition matrix $P$, a probability distribution $\pi$ on $\Omega$ is a stationary distribution for $P$, if it satisfies the detailed balance equations:*

$$\pi(x)P(x,y) = \pi(y)P(y,x) \quad \forall x, y \in \Omega \tag{9}$$

*Proof.* What we want to show is that $\pi = \pi P$. For ease of notation, take $|\Omega| = n$.

Since $\pi P = (\pi \cdot P(\cdot, x_1), ..., \pi \cdot P(\cdot, x_n))$, want to show: $\pi \cdot P(\cdot, x_i) = \pi(x_i)\ \forall i$

$$\begin{aligned}\pi \cdot P(\cdot, x_i) &= \sum_{j=1}^{n} \pi(x_j) P(x_j, x_i) \\ &= \sum_{j=1}^{n} \pi(x_i) P(x_i, x_j) \\ &= \pi(x_i) \sum_{j=1}^{n} P(x_i, x_j) \\ &= \pi(x_i)\end{aligned}$$

□

## 2.3 Convergence Theorem

The Convergence Theorem tells us that as long as our Markov Chain is irreducible and aperiodic, regardless of what our initial state $x_0$ is, as $t$ grows arbitrarily large the probability distribution $P^t(x_0, \cdot)$ converges to the stationary distribution, $\pi$, according to the metric established below.

**Definition 2.7.** the Total Variation Distance between two probability distributions $\lambda_1, \lambda_2$ over the state space $\Omega$ is:

$$||\lambda_1 - \lambda_2||_{TV} := \max_{A \subseteq \Omega} |\lambda_1(A) - \lambda_2(A)| \tag{10}$$

That is, the total variation distance is the largest difference between the two probabilities assigned to a single event $A \subseteq \Omega$. An equivalent (and very useful) form of this definition is also displayed below:

$$||\lambda_1 - \lambda_2||_{TV} := \frac{1}{2} \sum_{x \in \Omega} |\lambda_1(x) - \lambda_2(x)| \tag{11}$$

Before stating and proving the Convergence Theorem, we must first prove a lemma about irreducible and aperiodic chains which requires the following fact about subsets of the natural numbers (a complete proof can be found in [LPW16] pp. 19):

*Remark.* Let the set $A \subseteq \mathbb{N}$ be closed under addition and have $\gcd(A) = 1$. Then $A$ contains all but finitely many natural numbers.

**Lemma 2.2.** *Let $P$ be an aperiodic and irreducible transition matrix over a state space $\Omega$. Then there exists some $t_0 \in \mathbb{N}$ such that if $t \geq t_0$, then $P^t(x, y) > 0$ for all $x, y \in \Omega$.*

*Proof.* Since $P$ is aperiodic, $\gcd(\tau(x)) = 1$ for all $x \in \Omega$. Also note that for all $a, b \in \tau(x)$, $(a + b) \in \tau(x)$, as $P^{a+b}(x, x) \geq P^a(x, x) * P^b(x, x) > 0$. Therefore, the remark above applies to $\tau(x)$ and, therefore, it contains all but finitely many natural numbers. Take $t_0(x)$ as the largest integer not in $\tau(x)$.

Since $P$ is irreducible, $\forall y \in \Omega$, there exists $r(x, y)$ such that $P^{r(x,y)}(x, y) > 0$. Therefore, if $t > t_0'(x, y) := t_0(x) + r(x, y)$ we have $P^t(x, y) \geq P^{t-r(x,y)}(x, x) * P^{r(x,y)} > 0$.

To finalize the proof, take $t_0 = \max_{x,y \in \Omega} t_0'(x, y)$. □

**Theorem 2.2** (Convergence Theorem)**.** *Let $P$ be an irreducible and aperiodic transition matrix over $\Omega$ with stationary distribution $\pi$, then there exist $C > 0$ and $\alpha \in (0, 1)$ st:*

$$\max_{x \in \Omega} ||P^t(x, \cdot) - \pi||_{TV} \leq C\alpha^t \tag{12}$$

*Proof.* Since $P$ is irreducible and aperiodic, by the previous Lemma there exists an integer $r$ such that $P^r$ only has positive entries. It follows that $\forall x, y \in \Omega$ there exists $\delta(x, y) > 0$ such that $P^r(x, y) > \delta(x, y)\pi(y)$. Take $\delta = \min_{x,y \in \Omega} \delta(x, y)$.

Let $|\Omega| = n$ and $\Pi$ be an $n$ by $n$ matrix such that its rows are the stationary distribution $\pi$. It follows that $\Pi P = \Pi$ and $A\Pi = \Pi$ for any stochastic matrix (including $P$).

Now let $\theta := (1 - \delta)$ and define the stochastic matrix $Q$ by the equation:

$$P^r = (1 - \theta)\Pi + \theta Q$$

**Claim**: $P^{kr} = (1-\theta^k)\Pi + \theta^k Q^k$
*Proof of Claim.* By induction.
Base case ($k = 1$): true by definition of $Q$. ✓
Inductive case: assume true for some $n \in \mathbb{N}$. Want to show: true for $n+1$.

$$\begin{aligned} P^{(n+1)r} &= P^{rn}P^r \\ &= [(1-\theta^n)\Pi + \theta^n Q^n]P^r \\ &= (1-\theta^n)\Pi + (\theta^n Q^n)[(1-\theta)\Pi + \theta Q] \\ &= (1-\theta^n)\Pi + \theta^n Q^n(1-\theta)\Pi + \theta^{n+1}Q^{n+1} \\ &= (1-\theta^n)\Pi + (\theta^n - \theta^{n+1})Q^n\Pi + \theta^{n+1}Q^{n+1} \\ &= (1-\theta^{n+1})\Pi + \theta^{n+1}Q^{n+1} \ \checkmark \ \square \end{aligned}$$

Now multiplying both sides of the equation in the claim by $P^{t_0}$ we get

$$P^{kr+t_0} = (1-\theta^k)\Pi + \theta^k Q^k P^{t_0}$$

Which means that

$$P^{kr+t_0} - \Pi = \theta^k(-\Pi + Q^k P^{t_0})$$

Therefore, by our second definition of total variation distance (TVD), we may just sum the absolute values of all entries in any given $x_0^{th}$ row on both sides and divide by two. This sum for the term in parenthesis on the second side of the equation is *at most* 1, which is the largest possible value for any TVD. Therefore, we have that:

$$||P^{kr+t_0}(x_0, \cdot) - \pi||_{TV} \leq \theta^k$$

At last, choose $\alpha = \theta^{\frac{1}{r}}$ and $C = \frac{1}{\theta}$.

$\square$

### 2.4 Ergodic Theorem

The Ergodic Theorem tells us something very important about stationary distributions and the expected value of functions whose domain is the state space of the chain. The theorem presented in this section will be very important when Markov Chains are finally applied to Monte Carlo Integration in section **3.2**. The proof for this theorem will not given, but confirmation of its functionality will be tested in section **4.2.1**. A formal proof can be found in pp. 390 [LPW16].

**Theorem 2.3** (Ergodic Theorem)**.** *Given an irreducible Markov Chain with stationary distribution, $\pi$, over state space, $\Omega$, and a function $f : \Omega \to \mathbb{R}$, we have that $\forall\lambda$ a starting distribution over $\Omega$:*

$$P_\lambda(\lim_{t\to\infty} \frac{1}{t}\sum_{i=0}^{t-1} f(x_i) = E_\pi(f)) = 1 \tag{13}$$

## 3 Markov Chain Monte Carlo (MCMC)

The idea behind Markov Chain Monte Carlo (MCMC) is to use Markov Chains in order to sample according to non-uniform distributions. This method, although computationally expansive in lower dimensions, is extremely efficient in higher dimensions, as its convergence rate is independent of the dimension of the space.

### 3.1 Metropolis-Hastings Algorithm

The aim of the Metropolis-Hastings Algorithm is to do the following: given a probability distribution $\pi$ over the state space $\Omega$, find a transition matrix, $P$, that has $\pi$ as its stationary distribution. In order to do this, the theorem exposed in section **2.2** will be extremely useful. The algorithm is expressed as follows:

- **Step 1.** Pick any irreducible transition matrix, $\Psi$, over the desired state space, $\Omega$. If possible, choose a matrix that has a stationary distribution close to $\pi$.
- **Step 2.** Choose any initial state $x \in \Omega$.
- **Step 3.** Run the Markov Chain and generate a candidate, $y \in \Omega$, according to $\Psi(x, \cdot)$.
- **Step 4.** Instead of advancing to state $y$ immediately, do so with an *acceptance* probability $a(x, y) = \min(1, \frac{\pi(y)\Psi(y,x)}{\pi(x)\Psi(x,y)})$.[2]
- **Step 5.** If the candidate, $y$, is accepted, take it as the next value in the sequence. Otherwise, take the original state $x$ as the next one in the sequence.

This process results in a transition matrix, $P$, with probabilities:

$$P(x, y) = \begin{cases} \Psi(x, y) * \min(1, \frac{\pi(y)\Psi(y,x)}{\pi(x)\Psi(x,y)}) & \text{if } x \neq y \\ 1 - \sum_{z \in \Omega | z \neq x} P(x, z) & \text{if } x = y \end{cases} \tag{14}$$

**Lemma 3.1.** *the transition matrix, $P$, described by the Metropolis-Hastings algorithm has $\pi$ as its stationary distribution.*

*Proof.* In order to prove this claim, we will show that $\pi$ satisfies the detailed balance equations described in section **3.2**, proving that it is, indeed, a stationary distribution for $P$. Note that we only need to examine the case when $x \neq y$, as the detailed balance equation for $x = y$ is trivially true for any distribution and any transition matrix.

Without loss of generality, take: $\frac{\pi(y)\Psi(y,x)}{\pi(x)\Psi(x,y)} \geq 1$. Consequentially, $\frac{\pi(x)\Psi(x,y)}{\pi(y)\Psi(y,x)} \leq 1$.

Want to show: $\pi(x)P(x, y) = \pi(y)P(y, x)$

Left Hand Side $= \pi(x)P(x, y) = \pi(x)\Psi(x, y)$

Right Hand Side $= \pi(y)P(y, x) = \pi(y)\Psi(y, x)\frac{\pi(x)\Psi(x,y)}{\pi(y)\Psi(y,x)} = \pi(x)\Psi(x, y)$ □

### 3.2 The Importance of the Ergodic Theorem to MCMC Integration

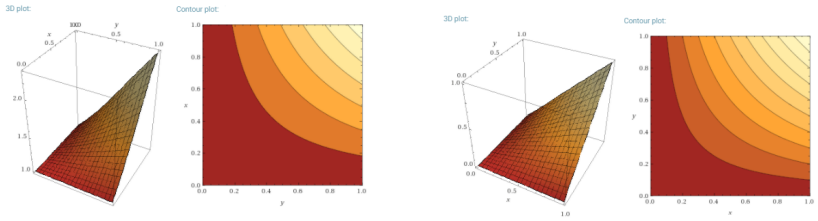
Some problems involving MCMC require completely independent sampling from the desired distribution. Therefore, a technique employed is to run a Markov Chain enough times so that the probability of drawing any state is that of the desired distribution, take note of that state, and then run the chain many more times before recording the next data point, so that the new sample is practically independent from the previous one and with probability according to the stationary distribution. However, when using MCMC for integration, the Ergodic Theorem tells us that we can record every data point (even if they are all directly related to the previous data point, and therefore not independent), and the overall mean will still equal the expectation of the function according to the desired distribution, with probability 1. Therefore, it is not needed to use expansive computational resources while not collecting data, thus making the computation of the integral much faster. This result will be employed in the example code explained in the next section.

## 4 Implementing The Method

In this section, the method described previously will be implemented using a program written in C, with some concepts taken from chapter 15.8 of [Pre07]. For the purposes of this example, the function integrated is $f(x, y) = e^{\sin(xy)}$, within the square $x, y \in [0, 1]$ which cannot be solved analytically. The code can be found in **Apendix A**.

The pdf chosen to resemble this function is $pdf(x, y) = kxy$, which is an easy one to be normalized (just use $k = 4$). Below is a 3D plot of both functions, along with their contour plots, which can be used to qualitatively verify that their shape is, indeed, similar.

[2] Note that the acceptance probability is well defined, as its denominator will never be 0, since $y$ being selected as a candidate after $x$ means that $\Psi(x, y) > 0$ and we will never choose $\pi$ with an entry $\pi(x) = 0$, as this would mean that we do not wish to sample $x$ at all, and so we may just exclude it from the state space $\Omega$ and make our transition matrix smaller and more efficient.



(a) $f(x, y) = e^{\sin(xy)}$ (b) $f(x, y) = xy$

Figure 1: The graphic representation of both functions. In the left, the 3D plot; in the right, the contour plot. Images created using WolframAlpha's "3D Plot" widget.

## 4.1 A Problem Relating to Memory

Markov Chain Monte Carlo Integration is only advantageous over other numerical methods on higher dimensions. However, if one chooses to divide a multidimensional region into a grid with each dimension partitioned into the same number of divisions, the amount of memory required to store the transition matrix increases incredibly drastically. Take for instance our two-dimensional example: if we want to divide each dimension into, say 500 divisions, this would result in a state space of size $500^2 = 250000$, and a transition matrix with a total of $(500^2)^2 = 12500000000$ entries. Assuming each entry is stored using 4 bytes of memory (an underwhelming estimate), this would mean that the entire matrix would require 250 GB of memory; an amount definitely accessible for specialized computers, but a bit too far fetched for personal computers as of now. In fact, it is not difficult to see that for $n$ dimensions, each divided into $m$ partitions, the amount of memory required would be $O(m^{2n})$. There are several ways in which to divide the region of integration as to lower this bound[3], but they fall out of the scope of this paper. In order to solve this problem, the implementation used in this paper calculated any specific transition probability each time it was required, rather than storing all values in memory. This, of course, slows down the program, but allows for an implementation achievable on a personal computer and is good enough for expository purposes.

## 4.2 Results

All data was collected using a personal computer, and can be found in **Appendix B**. The remaining of this section is analysis of the results obtained. The first part tests the hypothesis presented in section **3.2**, and the second addresses the accuracy of this implementation.

### 4.2.1 Testing Convergence With the Ergodic Theorem

Section **3.2** argued that it is possible to collect data from every state in the sequence, rather than skipping several iterations between any two states recorded. However, we do not know if this method also makes the estimator converge slower than if only every $t^{th}$ value were recorded. This would make sense, as the original process makes the probability distribution $P^t(x_0, \cdot)$, on average, closer to the stationary distribution we wish to sample from when data is collected. On the other hand, in order to obtain the uncertainty, $\delta$, for our estimator, we use the formula:

$$\delta = \frac{\text{stdev(points collected)}}{\sqrt{\text{total \# of points collected}}} \tag{15}$$

Which means that with more points collected, our estimator will be closer to the value of the integral. A fact that is reflected on the smaller value for the uncertainty.

In order to test this hypothesis, the program was run 140 times (10 for each case) with varying number of steps between points collected, and fixed 100K iterations of the Metropolis-Hastings algorithm (*i.e.*: 100K states in the entire sequence). This comparison is made as the same number of iterations leads to a roughly equivalent time for computation. Therefore, with this test we

[3] A particularly simple and yet very efficient way relies in using a Closed Network Model and having a separate Markov Chain to select each dimension. The only complication with this method is choosing marginal distributions that when jointed result in the multidimensional distribution you want to sample from. Chapter 1.15 of [Ser09] from Richard Serfozo addresses this subject in a very nice manner.

may determine which method leads to better results (smaller uncertainties) while requiring similar computational cost. The results for the average uncertainties in each case are shown below.

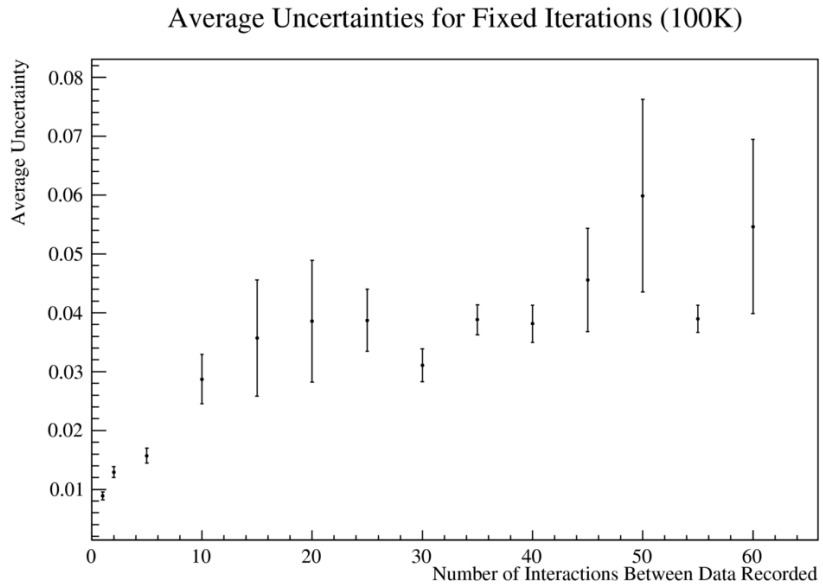



Figure 2: Uncertainties for trials with 500 division in each dimension and 100K iterations.

As it can be noted, with less points used to calculate the mean, not only did the average uncertainties got bigger, but they also varied way more, which makes sense, as the lower number of points leads to a higher chance of reaching results that are far from the expected value. It should also be noted here that the error bars used in the graph above are the standard deviation of the 10 uncertainties in each case, and that all the 140 numerical integrals were calculated using 500 divisions in each dimension, starting at state 150 (out of 250000), and that 1000 points were burned before any data was collected in each trial, as to not have the initial state interfere with the results.

As for the actual estimator calculated in each case, the results with less points recorded indeed deviated more from the reference value of the integral, which is 1.29885. Still, the results overall agree with this value, as the cases with more steps between points recorded also had larger error bars. More on accuracy is discussed in the next section.

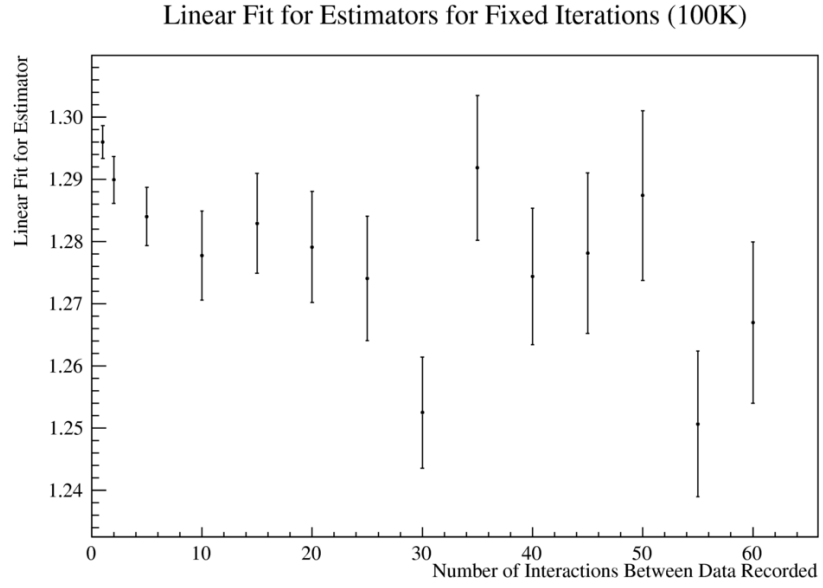



Figure 3: Results for trials with 500 division in each dimension and 100K iterations.

### 4.2.2 Testing Accuracy of the Model

In order to test the model's accuracy, the program was executed 40 times with 1000 divisions in each dimension and 250K iterations, all of which were used for the calculation of the mean (that is, taking advantage of the Ergodic Theorem). The results for these trials are shown below:

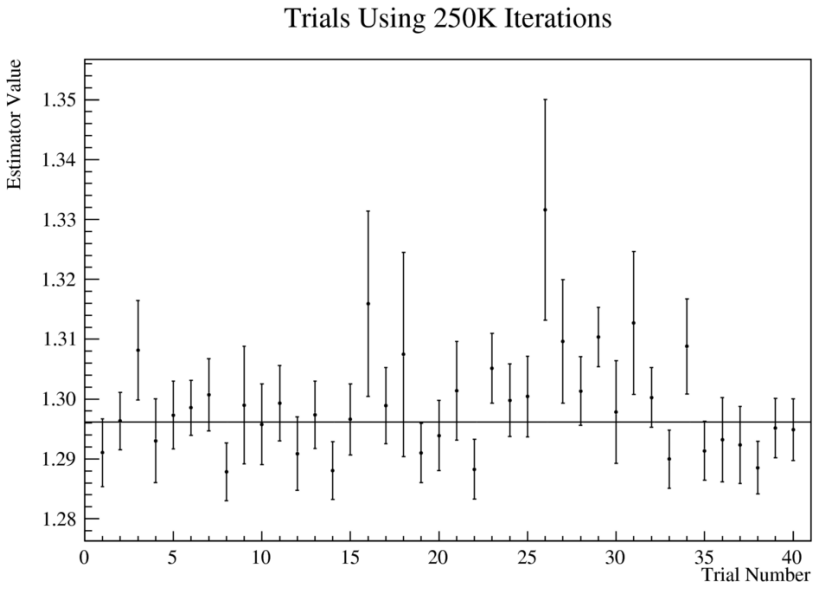



Figure 4: Results for trials with 1000 division in each dimension and 250K iterations.

The linear fit of all these data points (*i.e.*: using standard linear regression) results on $1.2962 \pm 0.0010$, while Mathematica's *NIntegrate* function evaluates the integral to 1.29885. These two values are within 3 uncertainties of each other, proving that the method is, indeed, valid. The source for the discrepancy among these two results may be due to the fact that we are dividing each dimension into only 1000 parts, whereas a specialized computer would be able to do these calculations with much more refinement. Another reason might even be that they are off by statistical chance, and that 1000 divisions is enough for a good measurement, as it is expected that roughly 9% of results fall within the range of 2-3 uncertainties from the reference value.

However, taking into account the results in Figure 3, where less divisions lead to an even smaller result ($1.2868 \pm 0.0016$), it can be deemed more likely that this difference is, indeed, caused by the number of divisions in each dimension rather than just chance.

## Acknowledgements

I would like to thank my mentor Shiva Chidambaram who guided me through the entire process of writing this paper and learning about the material covered in it, as well as all the people involved in the organization and implementation of the University of Chicago's Mathematics Research Experience for Undergraduates, in particular Prof. Laszlo Babai, who lectured with great mastery and sympathy in the Apprentice Program, and Prof. Peter May, who coordinated the program and reviewed all final papers.

## References

[LPW16] David A. Levin, Yuval Peres, and Elizabeth L. Wilmer. *Markov Chains and Mixing Times 2nd edition.* 2016.

[Pre07] William H Press. *Numerical recipes 3rd edition: The art of scientific computing.* Cambridge university press, 2007.

[Ser09] Richard Serfozo. *Basics of applied stochastic processes.* Springer Science & Business Media, 2009.

# Apendix A The Code

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

typedef struct _data {
	double result;		// the estimate of the integral
	double uncertainty;	// the uncertainty of that result
} data;

typedef unsigned long int state;

// Returns a random double within [0,1]
double generate_probability() {
	return (double) rand() / RAND_MAX;
}
// Given an inital state and a transition matrix
// Returns a state after one transition
state chain_step(state x_0, double (*trans_prob)(state, state, unsigned int), unsigned int divisions) {
	double rand_num = generate_probability();
	state x;
	for (x = 0; rand_num > 0; x++) {
		rand_num -= (*trans_prob)(x_0, x, divisions);
	}
	return x - 1;
}
//Calculates the acceptance rate described by the Metropolis-Hastings algorithim
double calculate_a_rate(state x_0, state x_can, double (*trans_prob)(state, state, unsigned int), double (*pi)(state, unsigned int), unsigned int divisions) {
	double nominator = (*pi)(x_can, divisions) * (*trans_prob)(x_can, x_0, divisions);
	double denominator = (*pi)(x_0, divisions) * (*trans_prob)(x_0, x_can, divisions);
	double aux = nominator / denominator;
	if (1 < aux)
		return 1;
	else
		return aux;
}
//Uses the Metropolis-Hastings algorithim to calculate the next state in the sequence
state met_hast(state x_0, double (*pi)(state, unsigned int), double (*trans_prob)(state, state, unsigned int), unsigned int divisions) {
	state x_can = chain_step(x_0, trans_prob, divisions);
	double rand_num = generate_probability();
	double a_rate = calculate_a_rate(x_0, x_can, trans_prob, pi, divisions);
	if (rand_num < a_rate)
		return x_can;
	else
		return x_0;
}
// The funtion to be integrated: f(x,y)=e^(sin(|x*y|))
double func(double x, double y) {
	return exp(sin(x*y));
}
// The function above divided by the pdf: f/pdf =
// the pdf used was pdf(x,y) = pdf(x)*pdf(y) = 4*x*y
double func_div_pdf(double x, double y) {
```

```
        return exp(sin(x * y))/(4*x*y);
}
// Given a initial state, a probability distribution, a transition matrix
// and the number of transitions between two sampled points
// Returns the resulting state after n transitions of the Metropolis-Hastings algorithim
// When testing the Ergodic Theorem, use n = 1.
state mc_sample(state x_0, double (*pi)(state, unsigned int), double (*P)(state, state, unsigned
int), unsigned int divisions, unsigned int n) {
        int i;
        for (i = 0; i < n; i++)
                x_0 = met_hast(x_0, pi, P, divisions);
        return x_0;
}
// Given a state, extracts the number x associated with it
// the 'division' in which x is found is the floor of (st / divisions)
// here, x is the midpoint of a division, so (1 / 2*divisions) is added to find x
double get_x(state st, unsigned int divisions) {
        return ((double) (st / divisions) / divisions) + (1 / ((double) 2 * divisions));
}
// Given a state, extracts the number y associated with it
// the 'division' in which y is found is the remainder of st when divided by divisions
// here, y is the midpoint of a division, so (1 / 2*divisions) is added to find y
double get_y(state st, unsigned int divisions) {
        return ((double) (st % divisions) / divisions) + (1 / ((double) 2 * divisions));
}
// Given a state and how many divisions each dimension has been divided into
// returns the probability of that state, according to the distribution we wish to sample from
// here, pi(x,y) is proportional to x*y, and the normalization constant depends on the number of
divisions
double pi(state st, unsigned int divisions) {
        double x = get_x(st, divisions);
        double y = get_y(st, divisions);
        double norm_const = pow(divisions, 2) / 4;
        return x * y / norm_const;
}
// Given two states and the number of divisions
// returns the probability of transition between these two states.
// This function corresponds to a transition matrix, without storing the actual values,
// but rather calculating them each time.
// This matrix will only serve as basis for the Metropolis-Hastings algorithm.
double trans_prob(state x_0, state x_f, unsigned int divisions) {
        return (double) 1 / pow((double) divisions, 2);
}
// This is the function that puts everything together and actually performs the Monte Carlo inte-
gration
data mc_importance_int(unsigned int divisions, unsigned long int st_0, unsigned int initial_burn,
unsigned long int points_to_collect, unsigned int steps_between_points) {
        int i;
        double results[points_to_collect];
        state st = st_0;
        double sum = 0;
        double quad_sum = 0;
        double x, y;
        // The following line makes it so that the initial states do not interfere too much in the first
data points collected.
        // a.k.a.: it performs the initial burn while not collecting any data.
        st = mc_sample(st, pi, trans_prob, divisions, initial_burn);
        // The following loop changes the state, then records data.
```

```
        for (i = 0; i < points_to_collect; i++) {
                st = mc_sample(st, pi, trans_prob, divisions, steps_between_points);
                x = get_x(st, divisions);
                y = get_y(st, divisions);
                results[i] = func_div_pdf(x, y);
                sum += results[i];
        }
        data dt;
        dt.result = sum / points_to_collect;
        for (i = 0; i < points_to_collect; i++)
                quad_sum += pow((results[i] - dt.result), 2);
        dt.uncertainty = sqrt(quad_sum / (points_to_collect * (points_to_collect - 1))); // The
uncertainty of the mean is given by: stdev/sqrt(points collected)
        fprintf(stderr, "\nIntegral = %lf Uncertainty = %lf\n", dt.result, dt.uncertainty);
        return dt;
}
```

# Apendix B Data

For the sake of completeness, the data collected using the code above and analyzed in section **4.2** is presented below. "Iterations Between Data" is abbreviated to "IBD"; "Error" is used as synonym to "Uncertainty".

Table 1: Data for Figure 4

| Estimator | Error | Estimator | Error |
|---|---|---|---|
| 1.291047 | 0.005673 | 1.301356 | 0.008242 |
| 1.296335 | 0.004771 | 1.288282 | 0.005004 |
| 1.308140 | 0.008290 | 1.305147 | 0.005838 |
| 1.293028 | 0.006991 | 1.299798 | 0.006066 |
| 1.297326 | 0.005658 | 1.300436 | 0.006730 |
| 1.298537 | 0.004601 | 1.331606 | 0.018444 |
| 1.300729 | 0.006014 | 1.309625 | 0.010306 |
| 1.287840 | 0.004815 | 1.301348 | 0.005728 |
| 1.298991 | 0.009828 | 1.310365 | 0.004974 |
| 1.295772 | 0.006742 | 1.297810 | 0.008586 |
| 1.299306 | 0.006286 | 1.312704 | 0.011950 |
| 1.290890 | 0.006133 | 1.300260 | 0.004995 |
| 1.297378 | 0.005626 | 1.289968 | 0.004866 |
| 1.288040 | 0.004830 | 1.308791 | 0.007916 |
| 1.296602 | 0.005920 | 1.291352 | 0.004908 |
| 1.315929 | 0.015461 | 1.293204 | 0.007059 |
| 1.298912 | 0.006370 | 1.292336 | 0.006405 |
| 1.307456 | 0.017058 | 1.288548 | 0.004361 |
| 1.290997 | 0.004927 | 1.295135 | 0.004974 |
| 1.293881 | 0.005866 | 1.294890 | 0.005176 |

Table 2: Data for Figures 2 & 3

| IBD | Estimator | Error |
|---|---|---|
| 1 | 1.290929 | 0.008019 |
| 1 | 1.296857 | 0.008670 |
| 1 | 1.296748 | 0.007197 |
| 1 | 1.301157 | 0.008133 |
| 1 | 1.299523 | 0.008853 |
| 1 | 1.286224 | 0.006321 |
| 1 | 1.290506 | 0.00737 |
| 1 | 1.315208 | 0.012806 |
| 1 | 1.307577 | 0.012656 |
| 1 | 1.302706 | 0.009005 |
| 2 | 1.309163 | 0.012369 |
| 2 | 1.286460 | 0.009733 |
| 2 | 1.307068 | 0.016303 |
| 2 | 1.273522 | 0.008246 |
| 2 | 1.296031 | 0.013499 |
| 2 | 1.278060 | 0.011789 |
| 2 | 1.320048 | 0.015393 |
| 2 | 1.301518 | 0.017720 |
| 2 | 1.287650 | 0.010633 |
| 2 | 1.288785 | 0.013694 |
| 5 | 1.298057 | 0.025807 |
| 5 | 1.299620 | 0.014984 |
| 5 | 1.301388 | 0.019021 |
| 5 | 1.281872 | 0.013264 |
| 5 | 1.285556 | 0.014451 |
| 5 | 1.292166 | 0.015599 |
| 5 | 1.271308 | 0.014285 |
| 5 | 1.288481 | 0.014850 |
| 5 | 1.280210 | 0.012696 |
| 5 | 1.268441 | 0.012327 |
| 10 | 1.308701 | 0.048929 |
| 10 | 1.278137 | 0.019207 |
| 10 | 1.239360 | 0.013927 |
| 10 | 1.285225 | 0.021654 |
| 10 | 1.302829 | 0.050472 |
| 10 | 1.328326 | 0.041371 |
| 10 | 1.274076 | 0.020637 |
| 10 | 1.291483 | 0.019323 |
| 10 | 1.320703 | 0.024126 |
| 10 | 1.297384 | 0.027445 |
| 15 | 1.284145 | 0.027000 |
| 15 | 1.268285 | 0.021235 |
| 15 | 1.286477 | 0.021379 |
| 15 | 1.446919 | 0.123275 |
| 15 | 1.304352 | 0.024138 |
| 15 | 1.334539 | 0.034236 |
| 15 | 1.291937 | 0.033764 |
| 15 | 1.342796 | 0.030144 |
| 15 | 1.236896 | 0.016735 |
| 15 | 1.296557 | 0.025071 |

| IBD | Estimator | Error |
|---|---|---|
| 20 | 1.253582 | 0.023574 |
| 20 | 1.267655 | 0.022880 |
| 20 | 1.294008 | 0.041893 |
| 20 | 1.276851 | 0.026396 |
| 20 | 1.316459 | 0.033659 |
| 20 | 1.324062 | 0.031909 |
| 20 | 1.256775 | 0.023626 |
| 20 | 1.282889 | 0.025624 |
| 20 | 1.285850 | 0.025707 |
| 20 | 1.441115 | 0.130275 |
| 25 | 1.265387 | 0.027777 |
| 25 | 1.290665 | 0.029474 |
| 25 | 1.328587 | 0.071192 |
| 25 | 1.310774 | 0.052727 |
| 25 | 1.285388 | 0.032066 |
| 25 | 1.254066 | 0.026122 |
| 25 | 1.267300 | 0.023066 |
| 25 | 1.321407 | 0.045082 |
| 25 | 1.344191 | 0.056961 |
| 25 | 1.250178 | 0.022852 |
| 30 | 1.291535 | 0.031800 |
| 30 | 1.281864 | 0.040783 |
| 30 | 1.315255 | 0.049902 |
| 30 | 1.223696 | 0.026955 |
| 30 | 1.250096 | 0.024852 |
| 30 | 1.256222 | 0.025587 |
| 30 | 1.278058 | 0.026978 |
| 30 | 1.308358 | 0.029684 |
| 30 | 1.280614 | 0.034873 |
| 30 | 1.189945 | 0.019440 |
| 35 | 1.286233 | 0.039139 |
| 35 | 1.256439 | 0.033748 |
| 35 | 1.252924 | 0.026864 |
| 35 | 1.385037 | 0.051481 |
| 35 | 1.325862 | 0.053583 |
| 35 | 1.259701 | 0.039542 |
| 35 | 1.335518 | 0.037728 |
| 35 | 1.308229 | 0.034481 |
| 35 | 1.316709 | 0.036922 |
| 35 | 1.291200 | 0.034729 |
| 40 | 1.317290 | 0.041661 |
| 40 | 1.344274 | 0.048443 |
| 40 | 1.361817 | 0.058878 |
| 40 | 1.281283 | 0.035878 |
| 40 | 1.245850 | 0.032698 |
| 40 | 1.327671 | 0.035975 |
| 40 | 1.214322 | 0.028885 |
| 40 | 1.311994 | 0.038252 |
| 40 | 1.306249 | 0.038065 |
| 40 | 1.235998 | 0.022592 |

| IBD | Estimator | Error |
|---|---|---|
| 45 | 1.319755 | 0.044367 |
| 45 | 1.276443 | 0.038576 |
| 45 | 1.295783 | 0.033105 |
| 45 | 1.269602 | 0.035613 |
| 45 | 1.285357 | 0.040183 |
| 45 | 1.378785 | 0.123965 |
| 45 | 1.252043 | 0.032298 |
| 45 | 1.278861 | 0.035884 |
| 45 | 1.260343 | 0.036237 |
| 45 | 1.274623 | 0.035816 |
| 50 | 1.525030 | 0.205191 |
| 50 | 1.296873 | 0.051928 |
| 50 | 1.286339 | 0.038780 |
| 50 | 1.269320 | 0.034092 |
| 50 | 1.307308 | 0.059571 |
| 50 | 1.256557 | 0.030199 |
| 50 | 1.270389 | 0.038789 |
| 50 | 1.343172 | 0.047018 |
| 50 | 1.344055 | 0.042417 |
| 50 | 1.260642 | 0.051023 |
| 55 | 1.218218 | 0.027496 |
| 55 | 1.285586 | 0.040366 |
| 55 | 1.308811 | 0.042726 |
| 55 | 1.251368 | 0.033031 |
| 55 | 1.276893 | 0.039001 |
| 55 | 1.219624 | 0.046151 |
| 55 | 1.236317 | 0.036487 |
| 55 | 1.285527 | 0.047144 |
| 55 | 1.253679 | 0.047852 |
| 55 | 1.233291 | 0.029480 |
| 60 | 1.273013 | 0.038445 |
| 60 | 1.297873 | 0.040407 |
| 60 | 1.236854 | 0.038934 |
| 60 | 1.298637 | 0.050119 |
| 60 | 1.458488 | 0.186980 |
| 60 | 1.250983 | 0.033871 |
| 60 | 1.296471 | 0.039079 |
| 60 | 1.259524 | 0.037188 |
| 60 | 1.234112 | 0.033641 |
| 60 | 1.284632 | 0.047753 |