# *NE 582*
# *Monte Carlo Analysis*

Instructor
Dr. Ronald E. Pevey

# Preface

These notes were written to accompany NE582, which is a graduate level course in Monte Carlo Analysis. This course is aimed primarily at nuclear engineering graduate students and follows a course (NE583) in deterministic neutral particle transport calculation methods.

Since the present course comes second (for most students) the actual derivation of the Boltzmann transport equation occurs in the first course and is not repeated here.

But a stronger effect of this being a follow-up course is the actual approach taken. Toward the Monte Carlo method itself. In the first decades of my career (at the Savannah River Plant and Laboratory) I worked more with deterministic methods of neutral particle transport (diffusion theory, discrete ordinates, integral transport methods) than I did with Monte Carlo. So, when I arrived at the university and it fell to me to teach a Monte Carlo course, I was never quite satisfied with how the method is presented (or even understood by most practitioners). I was more used to studying and coding numerical methods that began with a clear statement of the functional approximation that is occurring (e.g., expanding a desired function in a truncated Legendre polynomial series) and then solving for the coefficients. It bothered me that Monte Carlo methods do not do this, even though they are obviously approximating a continuous function, the particle flux, with a finite representation, just like the "deterministic" methods. In my zeal to correct this shortcoming I began teaching my (somewhat bewildered) students the function substitution approach, which is now relegated (in its full treatment) to the last chapters of this text, and wrote a paper explaining the approach. The paper was rejected and the students remained bewildered, so I eventually bowed to the reality that most developers and students regard Monte Carlo as a statistical simulation and not a method of solving an equation, and I returned to the more conventional order of presenting the material that you see here.

Although my heart is still with the function-substitution approach, I have relegated it to the later chapters and focused this course on the more immediately usable material in Chapters 1-4. Even when I tentatively introduce the Dirac substitution in Chapters 5 and 6, I lay low on the fuller development until the advanced chapters beginning with Chapter 7.

So, the structure of the material is such that:

- Chapters 1 & 2 cover the basic mathematical tools needed for Monte Carlo sampling and scoring.
- Chapters 3 & 4 apply these tools to an event-based neutral particle transport approach
- Chapter 5 introduces the Dirac delta function approximation method with general integral equation application
- Chapter 6 applies the Chapter 5 approximation to the neutral particle transport equation, both in forward and in adjoint form.
- Chapter 7 and beyond explore higher order Monte Carlo approximations beyond the Dirac delta.

# Chapter 1  Basic Monte Carlo Concepts

## 1.1 Introduction

Monte Carlo is a branch of mathematics that involves both modeling of stochastic event-based problems and the **stochastic** solution of equations.  In a sense, it is (and certainly feels like, when you do it) an **experimental** approach to solving a problem. It is like playing a game, hence the name (and probably hence the reason I like it).

When the analyst is trying to use a Monte Carlo approach to estimate a desired physical or mathematical value, the approach breaks itself down into two steps:

1. Devise a numerical experiment whose expected result will correspond to the desired value, $\overline{x}$ .
2. Run the problem to determine an estimate to this expected result.  We call the estimate $\hat{x}$ .

   *[NOTE: Throughout the course, the caret on top of a variable will denote a random variable. If it also has a subscript, then it denotes a sampled value of that variable.]*

The first step can either be very simple or very complicated, based on the situation.  If the mathematical or physical situation is itself stochastic, the experimental design step is very simple:  We simply let the Monte Carlo simulation mirror the stochastic "rules" of the situation. This is called an <u>analog</u> simulation, since the calculation is a perfect analog to the mathematical or physical situation.

Luckily for us, the physical situation we are focusing on—the interaction of neutral particles with material—is a stochastic situation.  All we have to do to get an estimate of a measurable effect from a transport situation is to simulate the decisions that nature makes in neutral particle transport: the probabilities involved in particle birth, particle travel through material media, particle interaction with the material, and particle contribution to the desired measurable value (also known as the "effect of interest" or "tally").

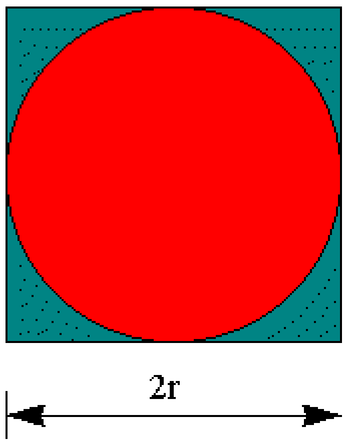
For processes that are not inherently stochastic, the experimental design is more complex and generally requires that the analyst:

1. Derive an equation (e.g., heat transfer equation, Boltzmann transport equation) from whose solution an estimate of the effect of interest can be obtained.
2. Develop a Monte Carlo algorithm to solve the equation.

Even more luckily for us, the transport of neutral particles falls into this second category as well: We have an equation to attack, and we can do so without any thought to the fact that the physical situation is itself stochastic.

In this course we will take advantage of the fact that the neutral particle transport equation falls into both categories, examining transport first as an event-based analog simulation, and then as the solution of a mathematical equation. I hope that this dual-view approach will give you particular insight into the Monte Carlo methods commonly used in Monte Carlo transport codes.

The rest of this section follows the traditional first example of Monte Carlo: a numerical estimation of $\pi$, based on use of a "dart board" approach. We know that the ratio of the area of circle to the area of the square that (just barely) superscribes it



is:

$$\frac{\pi r^2}{2r^{\ 2}} = \frac{\pi}{4} \tag{1-1}$$

Knowing this, we can design an experiment that will deliver an expected value of $\pi$. Let's set the origin at the center of the circle and the radius at 1; this gives us a circle area of exactly $\pi$. The experiment will then be:

1. Choose a point at random inside the square (assumed to span -1 to 1 in both x and y) by:
   - Choosing a random number between -1 and 1 for the x coordinate, and
   - Choosing a random number between -1 and 1 for the y coordinate.
2. Score the result of a the trial: Consider a "hit" (score = 4) to be the situation when the chosen point is inside the circle, i.e., $x^2 + y^2 < 1$ , with a "miss" scoring 0.

---

*Note: This may seem a little strange to you, to have an experiment with an "expected value" of $\pi$, when the only possible real results are 0 and 4. We professors are frequently accused of expecting the impossible. By the way, why is the score for a "hit" 4 instead of 1? A single trial of the area of something inside a square of area 4 can only result in one of two results: Either the inside object* *isn't there (area=0) or it fills the box (area=4). These "scores" are nothing more than very crude estimates of the circle's area.*

---

3. Run the experiment a large number ($N$) of times, with the final estimate of the circle's area being an average of the results:

$$\hat{\pi}_N = \frac{\sum_{i=1}^{N} s_i}{N}$$

where $s_i$ is the score for trial $i$.

A Java computer code to play this game is given by:

```
import java.util.Scanner;
class Pi
{
  public static void main(String[] args)
  {
    double pireal=Math.PI;
    while(true)
    {
      System.out.println("Input n?");
      Scanner sc=new Scanner(System.in);
      int ntry=sc.nextInt();

      if(ntry < 1)System.exit(0);
      int n=ntry;
      double pi=0.;
//*********************************************************************
//                                                                    *
//      For each history:                                             *
//                                                                    *
//*********************************************************************
      for(int ihistory=0;ihistory<n;ihistory++)
      {
        double x=2.*Math.random()-1.;
        double y=2.*Math.random()-1.;
        double score=0.;
        if(x*x+y*y < 1.)score=4.;
        pi+=score;
      }
      pi/=n;
      System.out.println(" After "+n+" trials, pi is estimated to be "+pi);
//*********************************************************************
//                                                                    *
//     Go back and see if user wants to run another problem           *
//                                                                    *
//*********************************************************************
    }
  }
}
```

*[NOTE: For those of you interested, Appendix A, contains some instructions for getting the Java language (which is free) and a brief tutorial on the subset of Java used in this course.]*

The results from running the program with various numbers of histories (with error) is:

```
After        10 trials, pi is estimated to be 3.2
After       100 trials, pi is estimated to be 3.08
After      1000 trials, pi is estimated to be 3.124
After     10000 trials, pi is estimated to be 3.1772
After    100000 trials, pi is estimated to be 3.14408
After   1000000 trials, pi is estimated to be 3.141196
After  10000000 trials, pi is estimated to be 3.1418224
After 100000000 trials, pi is estimated to be 3.14160504
```

By the way, the 10 million history case only took about a second on my laptop (which, I know, will seem ridiculously slow to future students).

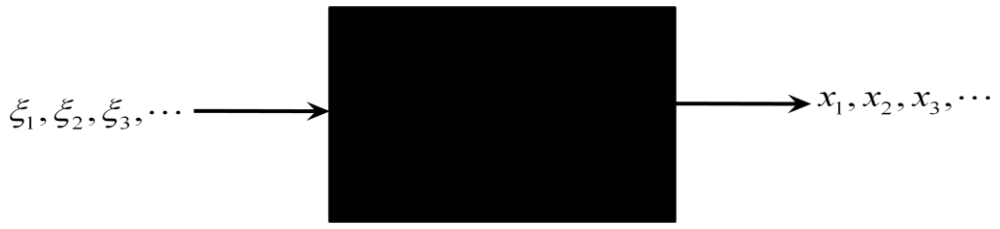
## 1.2 Statistical tools

There are three statistical formulae that we will be using over and over in this course:

1. Our estimate of the mean.
2. Our estimate of the standard deviation of the sample.
3. Our estimate of the standard deviation of the mean.

### *Monte Carlo as a stream of estimates*

Our basic view of a Monte Carlo process (properly designed to deliver an unbiased estimate of a desired physical or mathematical value of interest) is a black box that has a stream of random numbers as input and a stream of estimates of the effect of interest as outputs:



As we saw in our previous Java example (estimation of $\pi$ ), sometimes the estimates can be quite approximate, but with a long enough stream of *x*'s, we can get a good estimate from the average. The three formulae that we will develop here will help us gather important information from the estimates.

### *Estimate of the expected value, $\hat{x}$*

The first, easiest, and most important, deals with how we gather the best possible estimate of the expected value from the stream of estimates. The resulting formula for $\hat{x}$ is:

$$\hat{x} = \frac{\sum_{i=1}^{N} \hat{x}_i}{N} \tag{1-2}$$

Thus, our overall best estimate is the unweighted average of the individual estimates, $\hat{x}$, from each of the Monte Carlo histories

Let's compare this with two other situations:

1. Choosing from a continuous probability density function, $\pi\left(x\right)$, over a domain *(a,b)* (i.e., *x=a to x=b*).
2. Choosing from a discrete distribution, where you can only choose from M values, each of which has a probability of $\pi_i$ (e.g., throwing a die with 6 sides).

   *[NOTE: I know it is confusing to use $\pi\left(x\right)$ as the distribution when we just ran a problem to estimate π, but I like using pi as the PDF, so you will just have to live with it.]*

## *Review of PDFs and CDFs*

Before we go any further, let's get on the same page about PDF's and CDF's. A probability density function (PDF) is a general measure of the probability of choosing given individual element from the set of all possible choices. A cumulative distribution function (CDF) is the probability of choosing an element EQUAL TO OR LESS than a given individual element; the CDF is the integral of the PDF, so is—in general—smoother

*[NOTE: The "D" is different in the two abbreviations.]*

PDFs come in two flavors for our purposes:

1. Situations in which there are a finite number of discrete choices ($x_i$), each of which has a given probability, $\pi_i$, of being chosen.
2. Situations in which there is a continuous (infinitely dense) range of values that can be chosen, with the probabilities expressed in terms of probabilities of choosing within differentially sized ranges of values, using:

$$\Pr\{x \text{ chosen in } dx\} = \pi(x)dx$$

*[NOTE: It is extremely interesting to read the statistical literature, in which they deal with the logical inconsistency of choosing a single value from among an uncountably infinite number of choices. The logical inconsistency is, of course, that you end up choosing an element even though the probability of picking each element has to be zero (which is NOT because they are infinite but because they are UNCOUNTABLY infinite). They end up discriminating between "countably" infinite and "uncountably" infinite values, etc. It is interesting (as I said) but ultimately not useful for those of us who work with computers: Since we will always (I assume) work with real numbers with a fixed number of digits, we are never really choosing from an infinite number of choices when we pick a number*

*between 0 and 1. So, I am not going to get you wrapped around that axle in this course.]*

Each flavor of PDF has two rules: one involving non-negativity and one requiring that the sum of the probabilities of all possible choices equals one. For continuous, these rules amount to:

1. $\pi(x) \geq 0$ in the domain $(-\infty, \infty)$ and
2. $\int_{-\infty}^{\infty} \pi(x)\, dx = 1$

For discrete distributions, the corresponding rules are:

1. $\pi_i \geq 0$ for *i=1,2,..M*
2. $\sum_{i=1}^{M} \pi_i = 1$

*[NOTE: In problems that I give you—either in this writeup or orally—I reserve the right to state PDFs in unnormalized form. It is your responsibility to make sure that PDFs are normalized to add or integrate to 1. For example, I will often speak of a "uniform (or flat) distribution between a and b." I am saying that the PDF is a constant, but I am NOT telling you what the constant IS; that is your responsibility.y Okay. Just this once. The normalized PDF is* $\pi(x) = \dfrac{1}{b-a}$ *.]*

CDFs are denoted in this course with the CAPITAL pi and are defined by:

$$\Pi_j = \sum_{i=1}^{j} \pi_i \tag{1-3}$$

for discrete distributions and by:

$$\Pi(x) = \int_{-\infty}^{x} \pi(x')dx' \tag{1-4}$$

for continuous distributions.

## *Estimate of the mean,* $\bar{x}$

For a continuous distribution, the true mean, $\bar{x}$ , is found from:

$$\bar{x} = E(x) \equiv \int_{-\infty}^{\infty} x\, \pi(x)\, dx \tag{1-5}$$

For the discrete distribution, the corresponding definition for $\bar{x}$ is:

$$\bar{x} = E(x) \equiv \sum_{i=1}^{M} \pi_i x_i \tag{1-6}$$

The E(…) notation is read as “the expected value of x”. If I refer to the “expected value of” something else, then that something else would replace “x” in the above equations. For example, the expected value of $x^2$ would be written:

$$E(x^2) = \int_{-\infty}^{\infty} x^2 \, \pi(x) \, dx \tag{1-7}$$

or

$$E(x^2) = \sum_{i=1}^{M} \pi_i x_i^2 \tag{1-8}$$

As previously illustrated, we approximate the mean with Monte Carlo samples using an unbiased average of the sample results:

$$\bar{x} \approx \hat{x} = \frac{\sum_{i=1}^{N} x_i}{N} \tag{1-9}$$

---

**Example:** For our example of finding π, we were dealing with a binomial distribution (i.e., two possible outcomes):

Outcome 1 = Hit the circle: $x_1 = 4; \pi_1 = \pi / 4 = 0.7854...$

Outcome 2 = Miss the circle: $x_2 = 0; \pi_2 = 1 - \pi_1 = 0.2146...$

Therefore, the expected value is:

$$\sum_{i=1}^{2} \pi_i x_i = (0.7854...) \times (4) + (0.2146...) \times (0) = \pi$$

Included in the text (Equations 7.65 through 7.68) is a simple derivation that shows that Equation (1-9) is an unbiased estimate of $\bar{x}$, given that the individual estimates themselves are unbiased.

---

## *Estimate of the variance of the sample, $\sigma^2$*

As you know from statistics, the variance of a sample is the expected value of the squared error. It is a measure of the amount of variation we should expect from samples. The formulae, for the various distributions we have considered, are:

$$\text{For continuous distribution: } \sigma^2 = E((x-\bar{x})^2) \equiv \int_a^b \pi(x)(x-\bar{x})^2\,dx \qquad (1\text{-}10)$$

$$\text{For a discrete distribution: } \sigma^2 = E((x-\bar{x})^2) \equiv \sum_{i=1}^{M} \pi_i (x_i-\bar{x})^2 \qquad (1\text{-}11)$$

Analogous to our estimate of the mean, we have an estimate of the variance from Monte Carlo samples that is given by a straight average of each sample's error squared:

$$\sigma^2 \approx \frac{\sum_{n=1}^{N}(\hat{x}_n-\bar{x})^2}{N} \approx \frac{\sum_{n=1}^{N}(\hat{x}_n-\hat{x})^2}{N-1} \qquad (1\text{-}12)$$

where the samples $\hat{x}_n$ are chosen using the PDF $\pi(\hat{x})$.

This equation has some features that need explaining:

1. The second equation is an estimate of the first; the difference is that we square the difference from $\hat{x}$ instead of the difference from the true mean $\bar{x}$. This is because we usually do not know $\bar{x}$, just our estimate for it.

2. The second equation divides by *N-1* instead of *N*. The reduction to *N-1* compensates for the loss of a degree of freedom when we use $\hat{x}$ for $\bar{x}$.

   *[NOTE: If you KNOW the true mean, then you can use it and divide by N.]*

An interesting derivation from the book is the one from the text's equations (7-94) through (7-99); it shows that this is an unbiased estimate of the true variance of the sample.

## *Simplified calculational formula*

The problem with using the above equation for the estimate of the variance is that you cannot begin to use it until you know $\hat{x}$, which is not known until after all the samples have been drawn and averaged; therefore, to use this equation would mean that you would have to save all of the estimates, $\hat{x}_n$, until the end of the problem.

Fortunately, the equation can be reduced to a simpler form by simply expanding the square (like we learned in Algebra I):

$$
\begin{aligned}
\sigma^2 &\approx \frac{\sum_{n=1}^{N} (\hat{x}_n - \hat{x})^2}{N-1} \\
&= \frac{\sum_{n=1}^{N} \hat{x}_n^2 - 2\hat{x}_n\hat{x} + \hat{x}^2}{N-1} \\
&= \frac{N}{N-1}\left( \frac{\sum_{n=1}^{N} \hat{x}_n^2}{N} - 2\hat{x}\frac{\sum_{n=1}^{N} \hat{x}_n}{N} + \hat{x}^2 \frac{\sum_{n=1}^{N} 1}{N} \right) \\
&= \frac{N}{N-1}\left( \frac{\sum_{n=1}^{N} x_n^2}{N} - 2\hat{x}\hat{x} + \hat{x}^2 \frac{N}{N} \right) \\
&= \frac{N}{N-1}\left( \frac{\sum_{n=1}^{N} \hat{x}_n^2}{N} - \hat{x}^2 \right) \\
&= \frac{N}{N-1}\left( \frac{\sum_{n=1}^{N} \hat{x}_n^2}{N} - \left( \frac{\sum_{n=1}^{N} \hat{x}_n}{N} \right)^2 \right)
\end{aligned}
\tag{1-13}
$$

The advantage of this last equation is that each individual sample, $\hat{x}_n$, can be added to the two running summations (i.e., the numerators—one the sum of the samples and the other the sum of the samples squared) and then be discarded, eliminating the need to save them for later use.

### *Standard Deviation*

As you will remember, the square root of the variance is the **standard deviation**, which gives a measure of the variation expected in the individual sample $\hat{x}_n$.

---

**Example:** Back to our example of finding $\pi$ using the probabilities from the previous example,

Outcome 1 = Hit the circle: $x_1 = 4; \pi_1 = \pi / 4 = 0.7854$

Outcome 2 = Miss the circle: $x_2 = 0; \pi_2 = 1 - \pi_1 = 0.2146$

The associated variance of the sample would be:

$$\begin{aligned}\sigma^2 &= \sum_{i=1}^{2} \pi_i (x_i - \pi)^2 \\ &= (0.7842)\times(4-\pi)^2 + (0.2146)\times(0-\pi)^2 \\ &= 2.697\end{aligned}$$

$$\sigma = \sqrt{2.697} = 1.642$$

## *Estimate of the standard deviation of the mean*

The final formula is for the variance of the mean, $\sigma_{\hat{x}}^2$ and the corresponding standard deviation of the mean after *N* estimates of the mean have been obtained. (You should be careful to avoid confusing this with the variance of the sample itself.) The variance of the mean refers to the expected amount of variation we should expect from various estimates, $\hat{x}$, we might make of $\bar{x}$.

We should be careful here. Since it is our practice to run a Monte Carlo calculation, consisting of *N* samples, only once, we need to realize that what we are talking about here involves the variation that we would expect among many estimates of the mean (each involving *N* samples). If we run a series of Monte Carlo calculations, each of which involve *N* estimates, and each of which give us an estimate $\hat{x}$ of $\bar{x}$, then $\sigma_{\hat{x}}^2$ involves the variation that we would expect in these series of estimates, $\hat{x}$. Be sure you understand the difference between $\sigma_{\hat{x}}^2$ and $\sigma^2$.

There is a compact derivation in the text's Eq. 7-82 to Eq. 7-92, which shows that:

$$\sigma_{\hat{x}}^2 = \frac{\sigma^2}{N} \tag{1-14}$$

*[NOTE: In statistical literature, our estimate of the standard deviation is referred to as the "standard error", with the symbol S replacing the sigma we are more used to. I am going to stick with the sigma and trust that you will know from context whether it is an estimate or a true standard deviation.]*

The square root of this variance is, again, the standard deviation, this time the **standard deviation of the mean**:

$$\sigma_{\hat{x}} = \sqrt{\sigma_{\hat{x}}^2} = \frac{\sigma}{\sqrt{N}} \tag{1-15}$$

You should appreciate the power of this relation: It allows us to accurately estimate from one set of *N* samples (which would give us only one estimate of the mean) how much subsequent means of *N* samples each would be expected to vary. It saves us a boatload of computing.

This value $\sigma_{\hat{x}}$ is the second most important output result of a Monte Carlo calculation (the most important being our estimate of the mean itself, $\hat{x}$), giving us a measure of the confidence we can have in our estimate, $\hat{x}$. The way that Monte Carlo results are generally reported (in terms of our notation) is:

$$\hat{x} \pm \sigma_{\hat{x}}$$

Or, in the case of MCNP, in terms of fractional standard deviation (FSD):

$$\hat{x} \pm \frac{\sigma_{\hat{x}}}{\hat{x}}$$

---

**Example:** Back to our example of finding $\pi$, using the probabilities from the previous example, the standard deviation of the mean for a sample of N=10,000 would be:

$$\sigma_{\hat{x}} = \sqrt{\sigma^2_{\hat{x}}} = \frac{\sigma}{\sqrt{N}} \approx \frac{1.642}{\sqrt{10000}} = 0.01642$$

(where 1.642 was found from a previous example).

---

We can add this to our previous coding to get a measure of the expected accuracy of the answer:

```
import java.util.Scanner;
class Pi1
{
  public static void main(String[] args)
  {
    while(true)
    {
      System.out.println("Input n?");
      Scanner sc=new Scanner(System.in);
      int ntry=sc.nextInt();

      if(ntry < 1)System.exit(0);
      int n=ntry;
      double pi=0.;
      double piSquared=0;
//*********************************************************************
//                                                                    *
//      For each history:                                             *
//                                                                    *
//*********************************************************************
      for(int ihistory=0;ihistory<n;ihistory++)
      {
        double x=2.*Math.random()-1.;
        double y=2.*Math.random()-1.;
        double score=0.;
        if(x*x+y*y < 1.)score=4.;
        pi+=score;
        piSquared+=score*score;
      }
      pi/=n;
      piSquared/=n;
      double varDist=n/(n-1.)*(piSquared-pi*pi);
      double sdDist=Math.sqrt(varDist);
      double varMean=varDist/n;
      double sdMean=Math.sqrt(varMean);
      System.out.println(" After "+n+" trials, pi is estimated to be "+pi+
      " +/- "+sdMean);
      System.out.println("   The SD of the distribution is ~"+sdDist);
//*****************************************************************
//                                                                *
//     Go back and see if user wants to run another problem       *
//                                                                *
//*****************************************************************
    }
  }
}
```

With this new coding, the same selection of n values adds estimates of precision:

```
After 10 trials, pi is estimated to be 2.8 +/- 0.6110100926607788
   The SD of the distribution is ~1.932183566158592

After 100 trials, pi is estimated to be 3.2 +/- 0.1608060504414739
   The SD of the distribution is ~1.608060504414739

After 1000 trials, pi is estimated to be 3.264 +/- 0.04903782936375457
   The SD of the distribution is ~1.5507123230015005

After 10000 trials, pi is estimated to be 3.1564 +/- 0.016318717291913764
   The SD of the distribution is ~1.6318717291913765

After 100000 trials, pi is estimated to be 3.13784 +/- 0.005201295211648612
   The SD of the distribution is ~1.6447939651737167

After 1000000 trials, pi is estimated to be 3.139344 +/- 0.0016437450993271284
   The SD of the distribution is ~1.6437450993271285

After 10000000 trials, pi is estimated to be 3.1408428 +/- 5.194687647526035E-4
   The SD of the distribution is ~1.6427044699324211

After 100000000 trials, pi is estimated to be 3.14172616 +/- 1.6420905585674223E-4
   The SD of the distribution is ~1.6420905585674224
```

## 1.3 Markov and Chebyshev Inequalities

Most engineering students come away from the obligatory undergraduate statistics class thinking that about 68% of samples fall within one standard deviation of the mean, failing to properly recognize that this is a property of the normal distribution only (and that most distributions are not normal). When asked about the corresponding properties for other distributions, though, the students end up with nothing to say (although many of them take quite awhile to say it).

This is not too bad, actually, because most mathematicians don't have much to say either. But, it is useful to have at least a vague understanding of how non-normal distributions are distributed;

the main tool for this is the Chebyshev inequality, which comes from the Markov inequality. So, let's take a look at these two. First the Markov inequality.

If I tell you that I have a group of people whose average weight is 100 pounds, and I ask you how many of the group weigh more than 200 pounds, what would you say? (That is, after saying, "How could I possibly know, you idiot? You just made up the problem!") Can you say anything at all about it?

After thinking a minute, you should be able to say, "Well, it can't be more than half. If more than half weigh more than 200, then the average would have to be higher than 100." (Since weight has to be positive.)

This logic is expressed in the <u>Markov</u> inequality. It states that if all samples are non-negative, then the fraction that is greater than $n\bar{x}$ has to be less than $1/n$. The formal proof is a string of inequalities that you need to be able to reproduce on a test (<u>with</u> the explanation):

$$\begin{aligned}
E(x) \equiv \bar{x} &= \int_0^\infty x\,\pi(x)\,dx \\
&\geq \int_{n\bar{x}}^\infty x\,\pi(x)\,dx \quad \text{(Because the range of integration is smaller)} \\
&\geq \int_{n\bar{x}}^\infty (n\bar{x})\,\pi(x)\,dx \quad \text{(Because } n\bar{x} \text{ is the lower limit of x)} \\
&\geq n\bar{x}\int_{n\bar{x}}^\infty \pi(x)\,dx \quad \text{(Because } n\bar{x} \text{ is a constant)} \\
&\geq n\bar{x}\,\Pr\{x \geq n\bar{x}\} \quad \text{(Because the integral defines the probability)} \\
\therefore \quad & \Pr\{x \geq n\bar{x}\} \leq \frac{1}{n}
\end{aligned} \tag{1-16}$$

The Chebyshev inequality builds on this and is not quite as intuitive (at least to me). Its derivation begins with the observation that since the squared error of a distribution has to be positive, then the distribution of squared errors satisfies the requirements of the Markov inequality. Therefore (and this is a little tricky), we can substitute $(x-\bar{x})^2$ for $x$ (along with its expectation, the variance) in the previous equation:

$$\begin{aligned}
&E((x-\bar{x})^2) \equiv \sigma^2 \geq n\sigma^2 \Pr\{(x-\bar{x})^2 \geq n\sigma^2\} \\
&\therefore \quad \Pr\{(x-\bar{x})^2 \geq n\sigma^2\} \leq \frac{1}{n}
\end{aligned} \tag{1-17}$$

We then modify the above slightly by replacing $n$ with $n^2$ (i.e., simply replacing one positive constant with another one) and recognizing that the same probability must apply to the square root of the values, (i.e., the probability that *x* squared is greater than 4 must be the same as the probability that the absolute value of *x* is greater than 2):

$$\Pr\{(x-\bar{x})^2 \ge n^2\sigma^2\} \le \frac{1}{n^2}$$
$$\therefore \Pr\{|x-\bar{x}| \ge n\sigma\} \le \frac{1}{n^2} \tag{1-18}$$

This final result says that the probability that we choose a value more than *n* standard deviations away from the mean of a distribution has to be less than or equal to $1/n^2$. This is analogous to our knowledge of the normal distribution, but much less precise. For example, if we know that a variable is distributed normally, then we can say that 68.27% of the variables should fall within on standard deviation; if we have an unknown distribution, then the best we can say is that at least 0% are within one standard deviation.

ZERO per cent? What use is that? That doesn't tell us anything at all! But at least it gets a little better as we increase n: We can say that at least 75% are within two standard deviations—versus 95.45% for normally distributed variables. That is not much, but at least it is something. (For mathematicians, what really matters is that the probability is guaranteed to approach one as n increases. This is useful to mathematicians developing proofs.)

## 1.4 The Law of Large Numbers

Now for something a little more useful to us. The principal theoretical basis of Monte Carlo is the Law of Large Numbers (LLN), which can be over-simplified as:

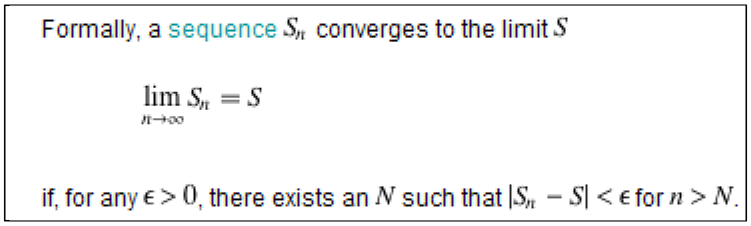
$$\bar{f} = \int_a^b f(x)\pi(x)\,dx = \lim_{N\to\infty} \frac{\sum_{i=1}^{N} f(\hat{x}_i)}{N} \tag{1-19}$$

where the samples $\hat{x}_i$ are chosen using the PDF $\pi(x)$. That is, a straight average of samples of a function (each of which is chosen using the same PDF) is guaranteed to approach the PDF-weighted mean of the function as you take more and more samples.

The usefulness of this is the Rosetta-Stone-like way in which the result of an integration is equated to the expected long-term result of a Monte Carlo process. This gives us the basis for applying the first "step" to the integration of functions, i.e., that a Monte Carlo game consisting of choosing samples of x according to the PDF and "scoring" the value of the function at those points will have an expected value equal to the indicated integral. All function-based Monte Carlo simulations are grounded in this Law.

We engineers will usually just write the LLN as above (i.e., with the limit as *N* "approaches infinity"), and then proceed to use it. However, you need to be aware of the fact that this formulation of the LLN is not quite rigorous.

The reason it is not goes back to our first encounter with the formal definition of the limit in our first calculus class:



Formally, a sequence $S_n$ converges to the limit $S$

$$\lim_{n\to\infty} S_n = S$$

if, for any $\epsilon > 0$, there exists an $N$ such that $|S_n - S| < \epsilon$ for $n > N$.

Or, as my calculus teacher used to say, "Give me an epsilon and I will give you an *N*."

Applying this rigorous definition of the limit to our oversimplified definition of the LLN, we would have to say that if we define our Monte Carlo estimate of $\overline{f}$ based on *N* samples as:

$$\overline{f}_N \equiv \frac{\sum_{i=1}^{N} f(\hat{x}_i)}{N} \qquad (1\text{-}20)$$

then our formulation of the LLN reduces to an assertion that the sequence $\overline{f}_1, \overline{f}_2, \overline{f}_3, \ldots$ converges to $\overline{f}$ in the limit as *N* increase. This is not really true.

Our situation falls short of the definition of a limit because no matter what epsilon I give the calculus teacher, for every *N*—no matter how large— $\overline{f}_N$ has some probability of falling outside epsilon because it is governed by a distribution that approaches (but does not reach) zero on the tails. Ultimately, our oversimplified definition of the LLN breaks down because we do not have a defined value for each of the elements $S_N$ of our sequence, but only a distribution of possible values for each element.

Thus, although we know that these distributions "tighten up" around the expected value as *N* increases, we cannot say that the value has to fall within ε of the mean. So convergence to the limit cannot really be claimed.

Although this situation is one that is unlikely to keep us up at night, as I said, I want you generally knowledgeable about two ways that this shortcoming is handled by formal mathematicians.

Both of the approaches switch our attention away from the sequence of estimates $\overline{f}_N$ (which will be different for each Monte Carlo simulation run), and instead concentrate on the unchanging probability distributions obeyed by the $\overline{f}_N$ for each value of *N*. (That is, we cannot say exactly what the value of $\overline{f}_N$ will be after *N* samples are averaged, but we can be confident that it will be distributed according to a fixed, predictable probability distribution.)

The first of these approaches is called the "weak form" of the LLN. It drops the requirement that I give you a specific *N* for each epsilon that you give me, replacing it with the weaker requirement that, if you give me an epsilon I will show you that, as *N* grows, the probability that a chosen *x* is farther away than epsilon from the mean has a limit of zero. (That is, the integral over the "tails" of the distribution more than epsilon away from the mean will get smaller as *N* increases.)

This is formally proven in Bernoulli's theorem, and it has one more point that you should know: It requires that the sequence be based on samples that have the (so-called) "i.i.d" property: They are independent and identically distributed. (For us, this simply translates into the requirement that the rules of the game do not change while the game is being played.)

The second form that I want you to know about is the "strong form" of the law. Instead of dropping the requirement that I have to give you a specific *N*, it adds to my burden. It says that, if you give me both an epsilon and a positive non-zero fraction delta (no matter how small), then I can give you an *N* such that, for any element of the sequence greater than *N*, the probability that the chosen x falls further than epsilon away from the mean is less than delta.

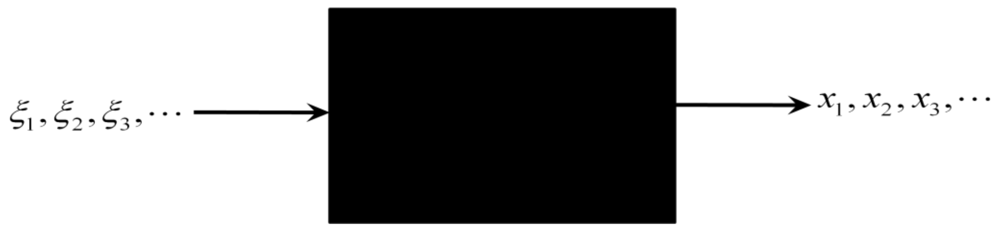
So, it deals directly with the fact that $\overline{f}_N$ has a distribution rather than a fixed value, by making the requirement depend on the (shrinking) distribution itself.

Interestingly, the proof of the strong form does NOT require that the samples be independent or that the rules of the game stay the same. All it requires is that the mean stays constant and that the variance of the game remains finite as the game is played. It opens up the possibility that we can change the rules of the game as we go (if we want to) based on what we have observed so far.

## 1.5 Central Limit Theorem

The second most useful theoretical basis of Monte Carlo is the Central Limit theorem, which states that if you sum independent variables chosen from the same distribution (no matter what the distribution is), this sum will tend to be distributed as a normal distribution (in the limit as the number of variables being summed increases without bound).

In terms of our stream of estimates:



this means that if you repeatedly average *N* values of x (e.g., you average the first 100, then the second 100, then the third 100, etc., and compare these averages), then IF *N* is large enough, these averages will be distributed in a normal distribution, no matter how the x's themselves are distributed. This is great news for Monte Carlo scorers because averaging large numbers of estimates of a desired value (with no preknowledge of the distribution of the population) is exactly what a Monte Carlo simulation does.

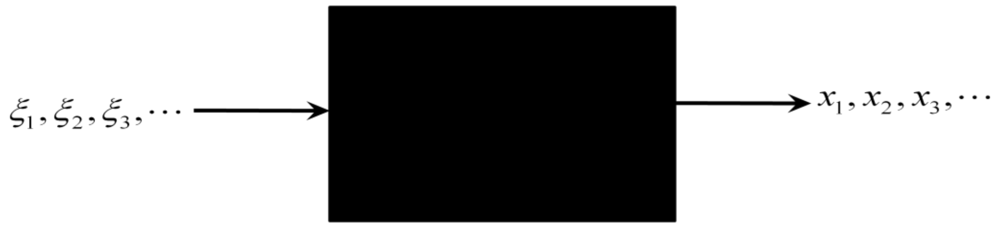
We can then use known properties of the normal distribution in characterizing these sums (e.g., that 68.3% of values are expected to be within one standard deviation of the mean). Of course, for this to be true, *N* has to be "large enough"; but in practice the thousands-to-millions of histories we use in our Monte Carlo simulations are large enough. (And, if we have doubts, we can always apply known "normality tests" to the *N* estimates.)

A necessary result of the Central Limit Theorem is that the sum of two (or more) normally distributed variables will itself be normally distributed.

This is as much as I am going to say about the Central Limit Theorem, but you have a homework problem that will allow you to dig a little deeper.

## 1.6 Pseudo-random Numbers

This section switches our focus to the input side of the "black box" first introduced in an earlier section:



That is, we now consider the sequences of uniformly distributed random numbers between 0 and 1 that we denote with the Greek letter xsi ($\xi$), which I will refer to as "squiggle" in the lectures. We generally refer to these computer-supplied numbers between 0 and 1 as "uniform deviates", although the algorithms that deliver them are still referred to as "random number generators".

As implied by the figure, these form the raw material from which every Monte Carlo method creates its estimates. And, as with any human enterprise, the quality of the inputs has a strong influence on the quality of the outputs. In the early days of Monte Carlo (the 40s and 50s), it was common for researchers to suggest physical devices—based on random physical phenomena such as nuclide decay—to generate truly random uniform deviates. As computers have gotten faster and faster, this has gotten less and less feasible. (For nuclide decay for example, we need to DETECT about 500 particles for each uniform deviate generated. If we were to try to keep up with a 1 gigaHz clock speed at, say, a 10% counting efficiency, we would need a (500 particles detected/uniform deviate)*(1.e9 numbers/sec needed)/(10% counting efficiency) ~= 100 curie source!

The most common sources of "random" numbers nowadays are the pseudo-random numbers, which (though not truly random) can be designed to be "random enough" for our purposes.

## *Linear congruential generators*

For the most part, Chapter 7 in the Numerical Recipes book (available at www.nr.com, which I recommend you get a copy of if you plan to do much computational work in your career) covers the material well.

Here are the principal ideas that I want you to remember:

- Any random number generator from a computer is not truly random (since it is repeatable).
- This repeatability is actually very important to us for “code verification” reasons. (That is, we want to be able to duplicate calculations to show that a computer code is working correctly.) Also, this repeatability is the basis of some Monte Carlo perturbation methods.
- The most common form is that of "linear congruential generators", which generate a series of bounded integers with each one using the one before it with the formula:

$$i_{n+1} = a i_n + b (\operatorname{mod} m) \tag{1-21}$$

$$\xi_{n+1} = \frac{i_{n+1}}{m} \, , \tag{1-22}$$

which makes 1 impossible, but 0 possible.

*[NOTE: In case your number theory is a little weak, b(mod m)—where both are positive integers in our case—denotes the remainder once the integer m has been subtracted from the b as many times as it can be (with a positive remainder). 13(mod7)=6, 17(mod 4)=1, etc.]*

There are several problems that can show up in random number generators:

1. Periodicity: The string of numbers is finite in length and so will repeat itself eventually. If this number is lower than the number of random deviates that we use in our algorithm, then we will lose accuracy (and not know it).
2. Bad choices of *a, b,* and *m* can lead to correlation among successive numbers (e.g., a very low number always followed by another very low number). It can be shown that a "full" period of m can be assured by choosing:
   - *a(mod 4)*=1
   - *m* as a power of 2
   - b odd.
3. Bad choices of *a, b,* and *m* might also mean that certain domains of the problem may be unreachable. (I have actually seen this occur.)
4. We have to be careful not to depend on the later digits of a uniform deviate: they may be less random than the early digits (since our "continuous" $\xi$'s are really fractions with *m* in the denominator).

These problems often show up in the "default" random number generators on computers.

---

**Example:** Find the pseudo-random series corresponding to $m$=8, $a$=5, $b$=3, $i_0$=1 (which fits Category 2 above, so should have period m=8).

**Answer:**

| # | i | *ai+b* | Mod *m* | $\xi$ |
|---|---|---|---|---|
| 0 | 1 | 8 | 0 | 0.000 |
| 1 | 0 | 3 | 3 | 0.375 |
| 2 | 3 | 18 | 2 | 0.25 |
| 3 | 2 | 13 | 5 | 0.625 |
| 4 | 5 | 28 | 4 | 0.5 |
| 5 | 4 | 23 | 7 | 0.875 |
| 6 | 7 | 38 | 6 | 0.75 |
| 7 | 6 | 33 | 1 | 0.125 |

---

## *Extending the period using more than one generator*

One consequence of use of the linear congruential generators is that no more than *m* uniform deviates can be generated before the series begins to repeat. Fortunately, you can get around this limitation by combining the results of several generators. If you use *N* different generators, add the *N* uniform deviates, and keep the fractional part, the resulting "combined" uniform deviate is uniformly distributed and the series has a period equal to the PRODUCT of the periods of the *N* generators. For example, the FORTRAN subroutine used to get a (double precision) uniform deviate for the KENO and MORSE computer codes at ORNL is the following:

```
DOUBLE PRECISION FUNCTION FLTRN()
INTEGER X, Y, Z, X0, Y0, Z0, XM, YM, ZM
DOUBLE PRECISION V, X1, Y1, Z1
EXTERNAL RANDNUM
COMMON /QRNDOM/ X, Y, Z
SAVE /QRNDOM/
PARAMETER ( MX = 157, MY = 146, MZ = 142 )
PARAMETER ( X0 = 32363, Y0 = 31727, Z0 = 31657 )
PARAMETER ( X1 = X0, Y1 = Y0, Z1 = Z0 )
X      = MOD(MX*X,X0)
Y      = MOD(MY*Y,Y0)
Z      = MOD(MZ*Z,Z0)
V      = X/X1 + Y/Y1 + Z/Z1
FLTRN  = V - INT(V)
RETURN
END
```

Close examination of this coding reveals that it uses 3 linear congruential generators (with *b*=0, *m*=prime number, and period=*m*-1).

*[NOTE: I have no idea where they found these three underlying generators, but each of them will cycle through every integer from 1 and m-1, leaving 0 out. This is necessary because, with b=0, if $i_n$ ever got to be 0, it would be 0 for all larger n.]*

The individual generators have periods of about 32000 (which makes them codable on 16-bit integer computer systems), but combined they have a limiting period of over 30 trillion; since their *m* values are prime numbers, it will take this long for the same combination of the three generators to come up again. This should be large enough for most (present-day) applications, although it is becoming inadequate for large massively parallel computing systems. (But all they would have to do is find a 4th one to throw in the mix to give them another factor of 30,000 in the period, right?)

## 1.7 Dimensionality

A nuclear particle transport expert would say that the neutron flux distribution in a material is a 7-dimensional value: three space coordinates, two direction angles—polar and azimuthal—, energy, and time. Particle transport problems are, to the Monte Carlo theorist, infinite dimensional problems.

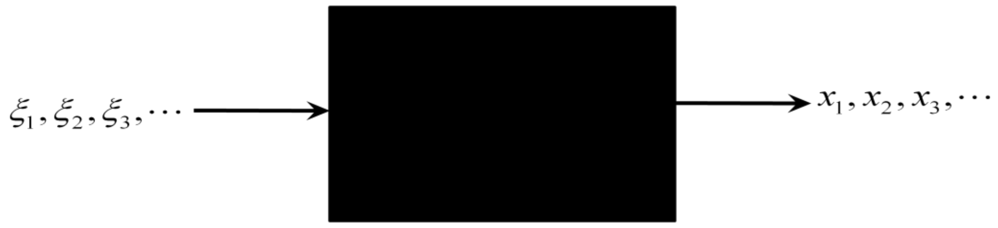
**Dimension**: The number of uniform deviates used for each estimate of the answer.

(So, every time a particle scatters, the code has to pull some random deviates to find its next energy, direction, and distance to next collision. And since there is no upper limit on how many times a particle can scatter, particle transport has infinite dimensionality.)

We began the course by saying that a Monte Carlo method consists of two steps. The first step is to design a numerical experiment and the second is to run it. No matter how revolutionary, innovative, etc., we are in the first step, the second looks the same: We "draw out" a certain number of uniform deviates (say, *N*) and, from them, produce a single number as the resulting estimate of whatever we are looking for.

The process is mathematically indistinguishable from estimating the average value of a function $f(x_1, x_2, \ldots, x_N)$ of *N* dimensions, each dimension having a domain (0,1).

That is, even though we engineers see the "black box" in my recurring graphic:



as being a very complicated statistical transformation (where all the work is done), if we back off far enough and squint our eyes a bit we can see that this is no more than a fancy function. (A function just consistently maps input scalars into output scalars; this black box just maps $\xi$ values into scalar estimates of our effect of interest, *x*. If you were to put in the same uniform deviates every time, you would get out the same estimates every time.)

---

**Example:** Find the expected value of the sum of two fair dice (i.e., 6 sides with values 1, 2, 3, 4, 5, and 6—each of which is equally likely to come up).

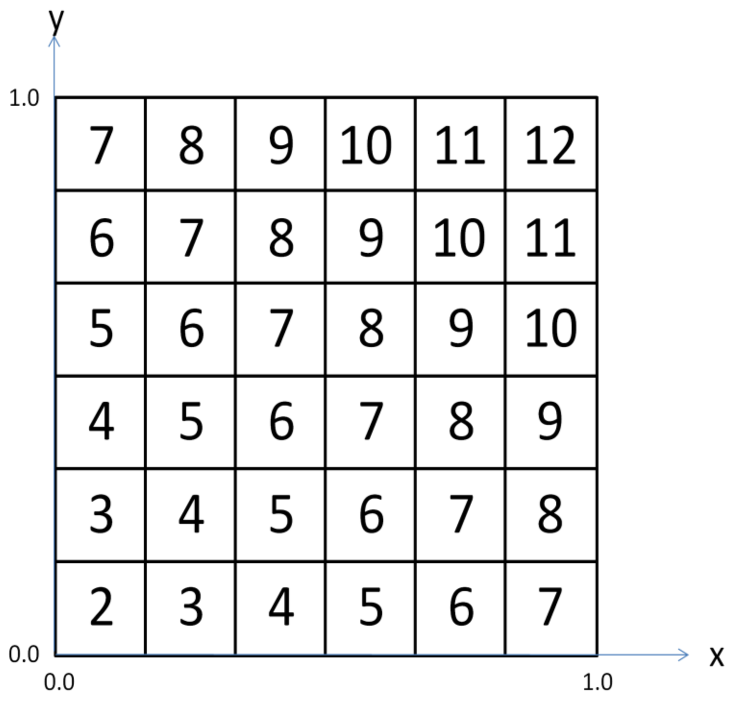
**Probability point of view:** The expected value of a single die is the average of its 6 values, which is 3.5. Therefore, two dice have an expected value of 7.

**Analog Monte Carlo point of view:** Devise a Monte Carlo method (numerical experiment) that mimics the action:

1. Pick an integer value among the first 6 integers. (This could be accomplished by "pulling" a uniform deviate, multiplying it by 6, and rounding up to the next integer.)
2. Pick another integer value among the first 6 integers. (Repeat step #1 with a different uniform deviate.)
3. Sum the values from step 1 and step 2 to form the estimate.

Repeat steps 1-3 *N* times. The resulting ANSWER is the sum of the scores divided by N.

**Functional point of view:** Looking at the Monte Carlo process we just ran, we can see that the process was one that turned two uniform deviates into an estimate. Although our thought processes involved chance and probability, the result does not—the same two uniform deviates would produce the same estimate every time. In other words, the Monte Carlo process actually defines a deterministic function of the uniform deviates that are used. In this case, the function is a two dimensional histogram (which actually looks like columns of blocks if you think of the numbers as the number of blocks in each of the 36 stacks of blocks you are looking down on.):



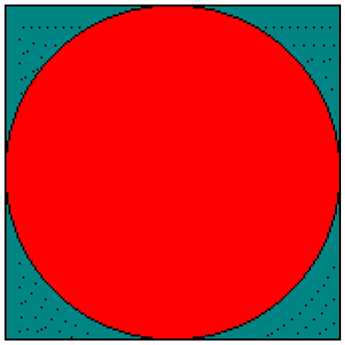
The two dimensions, x and y, correspond to the two uniform deviates that are used in the Monte Carlo process and have domains limited to (0,1). The answer itself can be written as the average of the function, which corresponds to the integral of the function:

$$\text{Answer}=\int_0^1\int_0^1 f(x,y)\,dx\,dy$$

where *f(x,y)* is the height of the structure at *(x,y)*—i.e., the height of the column that *(x,y)* falls into.

---

The example is applicable to every Monte Carlo process. All of them turn a set of uniform deviates between 0 and 1 into estimates of a desired value, so each of them is the same as the problem of finding the integral of a multi-dimensional function. If we were willing to do the work, we could even construct the function that maps the $\xi$ inputs into the scores. For example, you can imagine the chart we used for the original pi problem:



as you look down on a red cylinder (of height 4) on a blue table (with the lower left corner at (0,0) and width/height of 1). Again, if *f(x,y)* denotes the height (or lack thereof) at point *(x,y)*, then our process comes down to evaluating:

$$\pi = \int_0^1 \int_0^1 f(x,y)\,dx\,dy$$

And the result of a Monte Carlo process is the average of the estimates, so correspond (through the LLN) to an estimate of the integral of the underlying "scoring" function. Of course, the number of dimensions corresponds to the number of uniform deviates used to get each estimate of the answer, which explains the strange-sounding definition of "dimension" that we are using.

We can actually say a little more than this, because each of the dimensions is integrated from 0 to 1, so the functions are defined inside a **hypercube**, which is a multidimensional "box" with unit length sides, which we will consider bounded by 0 and 1. (That is, a 1D "box" is the line segment from 0 to 1; a 2D box is a square of sides 0 to 1; a 3D box is a cube of unit sides, a 4D box is a unit cube—of changing contents—that only exists from t=0 to t=1, etc.)

## 1.8 Discrepancy

Again, we are going to delve into a mathematical theorem and pull out the part we are interested in. The Koksma-Hlawka inequality states that the error of a Monte Carlo integration is the product of two terms:

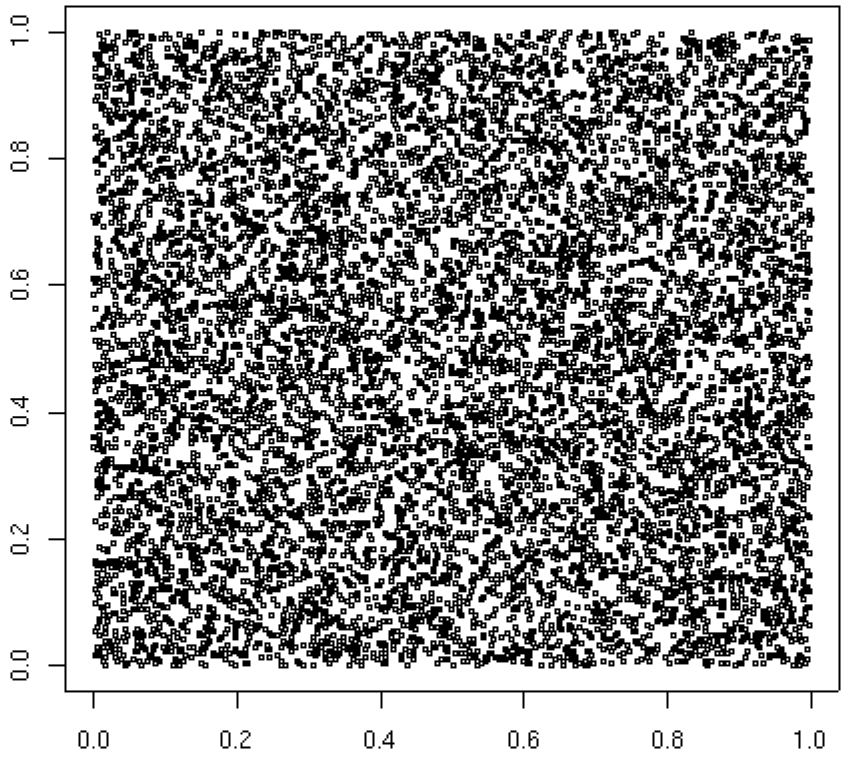
1. A value, $V(f)$ , called the "bounded variation" that depends on how smooth the function being integrated is. This value is very particularly defined "in the sense of Hardy and Krause" (whatever that means), but is neither of particular interest to us nor is it surprising. (It is just a rigorous way to express—and quantify—the idea that the variation among the $f(x_n)$ values will affect the accuracy of our results and that someone has figured out a way to quantify it. This is not hard to believe, but is not of particular interest to us; whatever we have to integrate is what it is and we have to deal with it.)
2. A value $D_N(x_1, x_2, \ldots, x_N)$, called the "discrepancy" of the sample points themselves. This is the interesting one for us because it turns out that this is the source of the $1/\sqrt{N}$ term that limits the accuracy of pseudo-random Monte Carlo methods. We will spend a little

time on this because it is NOT intuitive and affects our results; and besides, the choice of the sample points IS under our control.

Putting the two together says that the error is determined by two factors: How hard the function is to integrate ("variation") and how well our chosen sample points "cover" the domain of integration ("discrepancy").

In its most general form, "local discrepancy" of any "box" of values in the sample set is defined as the difference between the fraction of samples that should have been chosen from inside the "box" and the fraction of samples that actually fell within that "box". The total sample set discrepancy is just the largest "local discrepancy" among the infinite number of possible "boxes" in the sample set (which, as is true of much of mathematics, is easier to define than it is to calculate).

Below is a plot of 10,000 random (x,y) points (taken from "http://en.wikipedia.org/wiki/Low-discrepancy_sequence" ). What rectangular "box" looks to have the largest discrepancy to you? (It is a little hard to quantify where the points get thick, so concentrate on the places where the points get thin.)



The largest totally empty "hole" that is apparent to me (I am just eyeballing this) is near the point (0.65,0.90); it looks like an area that is about 0.04 wide and 0.02 high has no points at all, even though it should have 0.04*0.02*10000=8 points. So, it represents a seriously undersampled region (and there appear to be about a dozen others of about the same size).

Don't read too much into this simple example. The "box" surrounding the maximum local discrepancy doesn't have to be completely empty—in fact it usually isn't. For this example, even if you found the largest "empty" box in the drawing—which by definition would butt right up against 4 points marking the top, bottom, left, and right of the box—, it might very well be that you could get a lower discrepancy by expanding the box to include one of those limiting points and move that boundary to (just epsilon short of) the next point in that direction. As long as the box got more than 1/10000 larger for each extra point included, the discrepancy would increase. (In fact, the figure above seems to have an area at about (0.05, 0.38) that has a large open area with only one point in the middle of it. This probably has a higher discrepancy than my first example.)

Strangely (at least it seems strange to me), it can be shown that putting 10 times as many points on the graph does not reduce the maximum discrepancy by a factor of 10, but by a factor of about the square root of 10.

And this is the reason that Monte Carlo standard deviations decrease as the inverse of the square root of N; i.e., the size of the largest unsampled region goes down only as 1/sqrt(N), so our error does as well). And, a whole branch of Monte Carlo theory is concerned with replacing our pseudo-random numbers with sampling schemes that provide a more even spread of samples ("low discrepancy" sets) to make the accuracy improve faster.

A related measure (much easier to calculate) is the "star discrepancy" which is similarly defined, but instead of allowing any box, requires the candidate boxes to have a lower left hand corner at the origin. This is much easier to calculate (since the candidate "boxes" are butt up against points on only TWO sides instead of FOUR).

Applied to one dimensional samples (e.g., the uniform deviates delivered by an LCG), the star discrepancy is defined as:

$$D_N^*(x) = \max_{0<x<1} \left| \text{x-Fraction of random numbers in (0,x)} \right| \tag{1-23}$$

In practice (which for you will be in a homework problem), you do not have to check every value of x in the domain, because the maximum will occur either right before or right after one of the samples. For example, if the first of 10 samples is $\xi_n = 0.06$, then the only possible candidates for the star discrepancy from this point are 0.06 and 0.04—based on the fact that:

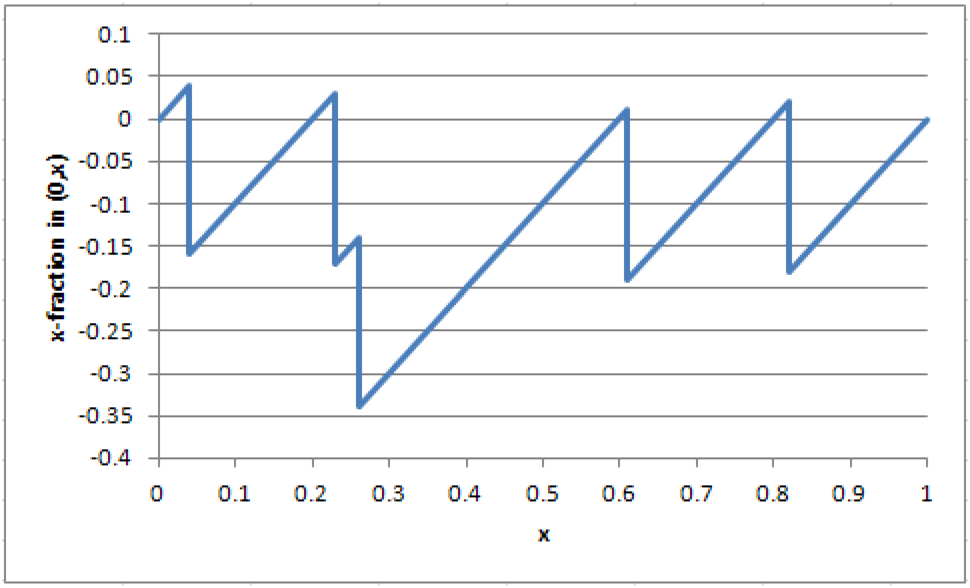
1. For x JUST LESS than that first value we SHOULD have picked 6% of our samples already but had ACTUALLY chosen 0%, giving us a "candidate" star discrepancy of 0.06; and
2. For x JUST GREATER than that first value we SHOULD have picked 6% of our samples already but had ACTUALLY chosen 10%, giving us a "candidate" star discrepancy of 0.04.

The winner (so far) is 0.06 from (1.)—but, of course, if the first number had been less than 0.05 the candidate start discrepancy would have come from (2.)—, and we move on to the next uniform deviate "point" to see if a larger star discrepancy is encountered.

---

**Example:** Find the star discrepancy for the uniform deviates 0.82, 0.26, 0.23, 0.04, and 0.61.

**Answer:** First we sort them into increasing order: 0.04, 0.23, 0.26, 0.61, 0.82. The graph of "x-fraction of uniform deviates in (0,x)" is this:



So, for each of the chosen uniform deviates, we find the star discrepancy "candidates" just before and after the value:

| Value | Fraction selected | Fraction expected | Difference |
|---|---|---|---|
| 0.04 | 0.0 | 0.04 | 0.04 |
| | 0.20 | 0.04 | -0.16 |
| 0.23 | 0.20 | 0.23 | 0.03 |
| | 0.40 | 0.23 | -0.17 |
| 0.26 | 0.40 | 0.26 | -0.14 |
| | 0.60 | 0.26 | -0.34 |
| 0.61 | 0.60 | 0.61 | 0.01 |
| | 0.80 | 0.61 | -0.19 |
| 0.82 | 0.80 | 0.82 | 0.02 |
| | 1.00 | 0.82 | -0.18 |

The largest absolute value of these candidates is 0.34, which becomes the star discrepancy. (Notice that, for each "pair" of candidates from each point, the first one minus the second one is always 1/N.)

## 1.9 Quasi-random sequences

As briefly mentioned in the previous section, Monte Carlo error can be reduced (theoretically) by using random number sequences that are more evenly distributed. These so-called **quasi-random numbers**, are not actually random—they follow fixed, repeatable patterns—but can be "plugged in" to Monte Carlo algorithms in place of the pseudo-random numbers to result in more accurate approximations.

They are not really used very much for Monte Carlo methods in radiation transport. But, because they are of such importance to the general field of Monte Carlo (i.e., beyond transport methods), you should have a taste of what they are, how to get them, and what they are useful for. "Quasi"-random numbers are not random at all; they are a deterministic, equal-division, sequence that are "ordered" in such a way that they can be used by Monte Carlo methods.

The easiest to generate are Halton sequences, which are "van der Corput" sequences based on prime number bases. In a Halton sequence the uniform deviates are found by "reflecting" the digits of prime base counting integers about their radix point. Clear as mud, right? A simple example makes it clear how to do it. ("Radix point" is the proper term for what we have always called the "decimal point"—but using "deci-" shows our chauvinism toward the number of our own fingers, so must be avoided to keep from insulting other species!)

Okay. You pick a prime number to be the base (e.g., 2). You then simply count in that base, like the first two columns below:

| Base 10 | Base 10 number translated to Base 2 | "Reflected" Base 2 | $\xi$=Reflected Base 2 translated into Base10 |
|---|---|---|---|
| 1 | 1 | 0.1 | 0.5 |
| 2 | 10 | 0.01 | 0.25 |
| 3 | 11 | 0.11 | 0.75 |
| 4 | 100 | 0.001 | 0.125 |
| 5 | 101 | 0.101 | 0.625 |
| 6 | 110 | 0.011 | 0.375 |
| 7 | 111 | 0.111 | 0.875 |
| 8 | 1000 | 0.0001 | 0.0625 |

The third column shows what we mean by "reflecting" the number; we put a mirror at the "radix point" so the digits become fractions (with the order reversed). When translated back to base 10, each of these become the next uniform deviate in the sequence.

If you follow the formula, you can see that after M=base$^{N-1}$ numbers in the sequence (1,3,7, etc., for base 2), the sequence consists of the equally spaced fractions (i/(M+1), i=1,2,3,...M). Therefore, the sequence is actually deterministic, not stochastic. Basically, it “covers” the (0,1) domain multiple times, each time with a finer resolution.

The beauty of the sequence is that the re-ordering results in a very nearly even coverage of the domain (0,1) even if the number of uniform deviates chosen does not correspond to base$^{N-1}$ .

Because you need ALL of the numbers in the sequence to cover the (0,1) domain (which is not true of the pseudo-random sequence), it is important that all of the numbers of the sequence be used on the SAME decision.

> *[NOTE: Pay attention to this: In our pseudo-random Monte Carlo algorithms, we have ONE random number generator. We use its first value for the first decisions, second for second decision, etc.. You CANNOT do this with quasi-random sequences.]*

That is, for a Monte Carlo process that has more than one decision, a different entire sequence must be used for each decision. The most common way this situation is handled is to use each of the low prime numbers in order—i.e., use the Base 2 sequence for the first decision, the Base 3 sequence for the second decision, then Bases 5, 7, 11, etc. for the subsequent decisions.

For reference, here is a Java internal class that delivers the Halton sequence for any base:

```
static class Halton
{
  public int base;
  int count=0;
  double previous=0.;
  Halton(int base0)
  {
    base=base0;
  }
  double next()
  {
    count++;
    int j=count;
    double frac=1./base;
    double ret=0.;
    while(j != 0)
    {
      int idigit=j - (j/base)*base;
      ret+=idigit*frac;
      j=(j-idigit)/base;
      frac/=base;
    }
    return ret;
  }
}
```

To use it, you add the above lines at the bottom of your Java coding (but inside the final close braces of your code). Then for each decision you want to use it for you:

1. Initialize it with a "Halton *decisionX*=new Halton(*baseX*);" line.
2. Subsequently, every time you want a new uniform deviate, you use the method *decisionX*.next().

For example, to simply count in base 5, you could use the following Java class:

```
class Test
{
  public static void main(String[] args)
  {
    Halton decision1=new Halton(5);
    for(int i=0;i<25;i++)
    {
      double newValue=decision1.next();
      System.out.println(" Entry "+(i+1)+" is "+newValue);
    }
  }

  static class Halton
  {
     …
  }
}
```

Note that the statistical formulas that we developed in this chapter are only applicable to problems using a pseudo-random number generator. So, if we use a quasi-random generator, the standard deviations printed by our code from those formulas will not reflect the fact that these results are more accurate than pseudo-random results. Therefore, you will need to add a calculation of the true error (if you know it) to your printed results so you will be able to gauge the improvement.

## Why do we use pseudo- instead of quasi-random?

If quadi-random sequences have lower discrepancy than pseudo-random sequences, why don't we use them in our transport codes?

The answer to this lies in the concept of dimensionality: the gains from using quasi-random sequences decreases with dimensionality of the problem being run. Whereas use of the pseudo-random sequence results (as we have seen) in errors that reduce by a factor of is $1/\sqrt{N}$ regardless of dimensionality, use of quasi-random sequences result in errors that reduce by a factor of $\frac{(\log N)^{d-1}}{N}$ for dimensionality *d*.

## Chapter 1 Exercises

**IMPORTANT:** For ALL Monte Carlo runs you make in this course, I expect you to report:

- The mean
- The standard deviation of the mean
- The number of histories you used

Please, for my convenience, always use $N=10^n$, where n is an EVEN number (i.e., N=100, N=10000, etc., as many as you can afford to wait for!)

Find the true mean, the true standard deviation of the distribution, and the standard deviation expected in a mean calculated from 1 million Monte Carlo samples of the following distributions. Do NOT run a Monte Carlo calculation. These problems only require paper and pencil.

1-1. $\pi(x) = 1,\ \ 0 < x < 1$

1-2. $\pi(x) = x,\ \ 0 < x < 3$

1-3. $\pi_1 = 1\ \ \pi_2 = 0.5\ \ \pi_3 = 0.25\,;\, x_1 = 1\ \ x_2 = 2\ \ x_3 = 3$

1-4. $\pi(x) = e^{-2x},\ \ 1 < x < 2$

1-5. $\pi_1 = 1\ \ \pi_2 = 2\ \ \pi_3 = 4\ \ \pi_4 = 8\,;\, x_1 = 1\ \ x_2 = 2\ \ x_3 = 3\ \ x_4 = 4$

1-6. For the distributions of problems 1-1 through 1-5, tell me how many Monte Carlo samples (N) would be required to get an estimate of the mean with a fractional standard deviation less than 1%. (Label them a-e.) Do NOT run the Monte Carlo calculations.

1-7. Given two uniform deviates $\xi_1$ and $\xi_2$ demonstrate (with Monte Carlo runs) that:

a. The standard deviation of $\xi_1 + \xi_2$ is $\sqrt{\sigma_1^2 + \sigma_2^2}$ .

b. The standard deviation of $\xi_1 - \xi_2$ is $\sqrt{\sigma_1^2 + \sigma_2^2}$ .

1-8. Research the Central Limit Theorem and prepare a short (< 5 page) report. (Show me something more than what I told you in the text.)

1-9. Find the period and last five numbers in the sequence (i.e., just before it repeats for the first time) for LCGs with the following properties:

a. m=16, a=9, b=1, i0=5

b. m=32, a=5, b=3, i0=3
c. m=64, a=13, b=5, i0=1

1-10. Research a pseudo-random number generator other than LCG and prepare a short (< 5 page) report.

1-11. If the first five uniform deviates drawn are 0.12509, 0.15127, 0.08885, 0.94886, and 0.54036, find the star discrepancy.

1-12. Demonstrate that the star discrepancy of an Linear Congruential Generator is proportional to 1/sqrt(N), by finding the expected star discrepancy for samples of 10, 40, 160, and 640 uniform deviates. (Of course, to find the expected star discrepancy you will have to get at least 100 samples of each, i.e., 100 averaged samples of the star discrepancy with N=10, 100 averaged samples with N=40, etc.)

1-13. Rework the PI problem from Chapter 1 with a Halton sequence of base 2 for x and base 3 for y. Plot the error (NOT the printed standard deviation) for *N* values of 100, 1000, 10000, 100000, and 1000000 for pseudo- vs. quasi-random solutions. Compare to the theoretical slope of $\frac{(\log N)^{d-1}}{N}$.

1-15. If you regularly run Monte Carlo problems with one million samples, what is the highest dimensionality for which quasi-random sequences might reduce the error? What about one billion? Assume proportionality constants are the same, i.e.,

$$\frac{1}{\sqrt{N}} = \frac{(\log N)^{d-1}}{N}$$

# Answers to selected exercises

Chapter 1

1-1. Mean=0.5, SDdist=0.289,SDmean=0.000289

1-2. Mean=2.0, SDdist=0.707, SDmean=0.000707

1-3. Mean=1.571, SDdist=0.728, SDmean=0.000728

1-4. Mean=1.3435, SDdist=0.263, SDmean=0.000263

1-5. Mean=3.267, SDdist=0.929, SDmean=0.000929

1-6.
- a. 3333
- b. 1250
- c. 2149
- d. 382
- e. 808

1-9.
- a. Period=16. Last 5: 0.0625,0.625,0.6875,0.25,0.3125
- b. Period=32. Last 5: 0.96875,0.9375,0.78125,0,0.09375
- c. Period=64. Last 5: 0.828125,0.84375,0.046875,0.6875,0.015625

1-11. 0.44873

1-12.
$$\begin{aligned} &10 \to 0.25906 \ \pm 0.00007 \\ &40 \to 0.13334 \ \pm 0.00012 \\ &160 \to 0.0679 \ \pm 0.0002 \\ &640 \to 0.03407 \ \pm 0.0001 \end{aligned}$$