Random Processes and Monte Carlo Methods

So far we have only considered systems that are deterministic. Deterministic systems can be chaotic, but they are not random. A process is random if it's impossible to determine the outcome of a single event, but the probability of a particular outcome can be predicted. And sometimes a process is not truly random, but we treat it as random because the information required to determine the exact outcome is not available, and so the process is best described using probabilities and averages. To apply randomness in a calculation, we need to use a random number generator.

1 Random Numbers

True random number generators generate random numbers based on some physical process that is truly random, like radioactive decay or thermal noise. In computational problems, the random generators we use are technically pseudo-random generators; they generate numbers by a deterministic formula. The numbers appear to be random, but can be reproduced. One of the most common random number generators is the *linear congruential random number generator*.

$$x' = (ax + c) \mod m \tag{1}$$

where a, c, and m are integer constants and x is an integer variable and $\mod$ is the modulo operation which returns the remainder when the first number is divided by the second number. Given x, compute x'. Then let $x = x'$ and repeat, for as many "random" numbers as needed.

Note that this function returns a number between 0 and $m - 1$. The first value chosen for x is called the *seed*. Given the same seed, you will get the same sequence of numbers. Some care must be given to the choice of the constants (a, c, and m). For example, choosing both x and c to be even would produce a sequence of only even numbers. It is recommended to use the values given in the example.

Example: Linear Congruential Generator

The linear congruential generator is easy to implement, but it's not the best random number generator. Fortunately, Python (and other languages) provide built-in random number generators that use better algorithms. Python uses the *Mersenne twister* algorithm, which is sufficient for most physics calculations. The `random` package provides several functions.

For example, calling `random.random()` produces a number uniformly distributed between zero and one. You also have the option to choose a seed for the generator. If you don't set a seed, your program will produce a different sequence of random numbers every time it is run. This is often the desired behavior, but there are some cases (like for debugging purposes), when it is useful to be able to reproduce the same sequence of numbers. In this case, you can call `random.seed(n)`, where n is an integer of your choice, to set the seed.

Example: Random Numbers

The `random` package also provides these functions:

- `randrange(n)`: returns an integer from 0 to $n-1$
- `randrange(m,n)`: returns and integer from m to $n-1$
- `randrange(m,n,k)`: returns an integer in the range m to $n-1$ in steps of k

Section 5, Exercise 1

Often, we want to simulate a random outcome based on a probability. For the coin example, with a probability $p = 0.5$ of the coin coming up heads, we could generate a random number between 0 and 1, and if the number is less than p, we say it's heads. If it's a biased coin that has a probability $p = 0.3$ of coming up heads, then a random number < 0.3 is counted as heads.

Example: Coin Toss

Let's see another example of this. Suppose we have a sample of Magnesium-23, which has a mean lifetime of $\tau = 16.3$ seconds. The standard equation for radioactive decay tells us that the number of uranium atoms remaining after a time t is

$$N(t) = N_0 e^{-t/\tau} \tag{2}$$

where N_0 is the number of magnesium atoms at time $t = 0$. This means that the fraction of atoms that have decayed after time t is

$$\begin{aligned} p(t) &= 1 - \frac{N(t)}{N_0} \\ p(t) &= 1 - e^{-t/\tau} \end{aligned} \tag{3}$$

This represents the probability for a single atom to decay in time t. Let's use this to plot the number of magnesium atoms as a function of time for 10 minutes = 600 s.

Example: Radioactive Decay

All the Python functions we have used so far generate uniformly distributed random numbers, i.e. all numbers in the interval are equally likely. What if we want to generate numbers

according to a probability distribution that is not uniform?

Suppose you have a source of random numbers z drawn from a distribution with probability density $q(z)$. Recall a probability density is a probability per unit length, so the probability of generating a number in the interval $z+dz$ is $q(z)dz$. We have also have the function $x(z)$. If z is a random number, then x will be too, but it will have a different probability density $p(x)$. We want to choose the function $x(z)$ so that p is the non-uniform distribution that we want.

The probability of generating a value of x between x and $x+dx$ must be equal to the probability of generating a value of z in the *corresponding* z interval, i.e.

$$p(x)dx = q(z)dz \tag{4}$$

Let's assume that random numbers z are generated using the Python function `random.random()`. Then $q(z) = 1$ for $0 < z < 1$ and zero everywhere else. If we integrate the above equation from $-\infty$ to z on the right side, the corresponding interval on the left-hand side is $-\infty$ to $x(z)$:

$$\begin{aligned}
\int_{-\infty}^{x(z)} p(x')dx' &= \int_{-\infty}^{z} q(z')dz' \\
\int_{-\infty}^{x(z)} p(x')dx' &= \int_{\infty}^{0} (0)dz' + \int_{0}^{z} (1)dz' \\
\int_{-\infty}^{x(z)} p(x')dx' &= z
\end{aligned} \tag{5}$$

As an example, let's say we have numbers generated uniformly from 0 to 1 using Python's `random.random()` function. We want to generate random numbers in the interval zero to infinity with the distribution

$$p(x) = \mu e^{-\mu x} \tag{6}$$

Using the equation above, we have

$$\begin{aligned}
\int_{0}^{x(z)} p(x')dx' &= z \\
\int_{0}^{x} (z)\mu e^{-\mu x'}dx' &= z \\
e^{-\mu x'}\Big|_{0}^{x(z)} &= z \\
1 - e^{-\mu x} &= z \\
1 - z &= e^{-\mu x} \\
\ln(1-z) &= -\mu x \\
x &= -\frac{1}{\mu}\ln(1-z)
\end{aligned} \tag{7}$$

Let's look at this with $\mu = 1$.

Example: Transformation Method

Python also offers some built-in functions for generating non-uniform random numbers from often-used probability distributions, such as a Gaussian.

Recall back in Section 1 when I provided some height data for you to plot. I looked up the mean and standard deviation for adult male height in the US, and used Python's `random.gaus()` function to generate the data.

Example: Generating Height Data

2 Random Systems: Diffusion

As an example of a random system, we will consider diffusion, like the spreading of a drop of cream in coffee. The drop of cream is made up of many particles, which have complicated trajectories due to collisions between the particles. In principle, this system is deterministic. The interactions of the particles are governed by Newton's law, and we could write a system of differential equations (one for each particle), the solutions of which would tell us the trajectory of each particle. Solving $\sim 10^{23}$ differential equations is not practically possible! Furthermore, the trajectory of each particle is really more information than we need. We care to know information about the system as a whole. For example, how long does it take the cream to be fully mixed into the coffee? What parameters affect this time? It's better to rely on a statistical description of the system.

We will model the deterministic process of the particles interacting with each other as a random process that is designed to have the same average properties as the deterministic system. In the case of diffusion, this process can be modeled with a "random walk," a process in which each particle moves one step at a time, according to certain rules.

Let's model the simplest version of a random walk. Each walker is allowed to take steps in one dimension (confined to the x-axis) with size $\Delta x = 1$. The walker starts out at $x = 0$, and for each step, the walker has an equal probability of going left or right.

Example: Random Walk in One Dimension

We see that the average value of x at step i is equal to zero, $\langle x_i \rangle = 0$. This makes sense, since a walker is equally as likely to step right or left. We see that the average value of x^2 at step i is approximately equal to the step number, $\langle x_i^2 \rangle \approx i$. Each step is sequential in time (for example, we can assume that a step happens every 1 s), so we can also write this

as $\langle x_i^2 \rangle \approx t$. Diffusion can be described as

$$\langle x^2 \rangle = 2Dt \tag{8}$$

where D is the diffusion constant. In this case, we see that $D = 1/2$.

How does this behavior differ from that of "free" particles, which move without interactions with other particles? For free particles, $x = vt$, so the distance from the origin grows linearly with time. For our random walker particles, the root-mean-square distance, $\sqrt{\langle x^2 \rangle} \sim \sqrt{t}$. This $t^{1/2}$ growth is slower than the t^1 growth of the free particle. So a random walker takes more time to escape from the origin than a free particle.

Let's look at a similar model in three dimensions. Assume the particle can take a step of unit length in one of six directions: $\pm x$, $\pm y$, or $\pm z$.

Example: Random Walk in Three Dimensions

Section 5, Exercise 2

So why are random walks representative of the process of diffusion? Diffusion is described by the following:

$$\frac{\partial \rho}{\partial t} = D \nabla^2 \rho \tag{9}$$

where ρ is the density of particles, D is the diffusion constant, and $\nabla^2 \rho = \frac{\partial^2 \rho}{\partial x^2} + \frac{\partial^2 \rho}{\partial y^2} + \frac{\partial^2 \rho}{\partial z^2}$ is the Laplacian. In one dimension, we can write

$$\frac{\partial \rho}{\partial t} = D \frac{\partial^2 \rho}{\partial x^2} \tag{10}$$

We haven't talked about partial differential equations, but numerical solutions are possible. However, let's consider one special case:

$$\rho(x,t) = \frac{1}{\sigma(t)} \exp\left[-\frac{x^2}{2(\sigma(t))^2} \right] \tag{11}$$

where

$$\sigma(t) = \sqrt{2Dt} \tag{12}$$

This is a Gaussian function with a time dependent standard deviation. (You can verify that it satisfies the differential equation above.) Figure 1 shows a plot of this function for different values of t. When time is close to zero, the particles are located near $x = 0$ with high density. As time increases, the density of particles decreases, and the particles spread out to cover more space in x with lower density.

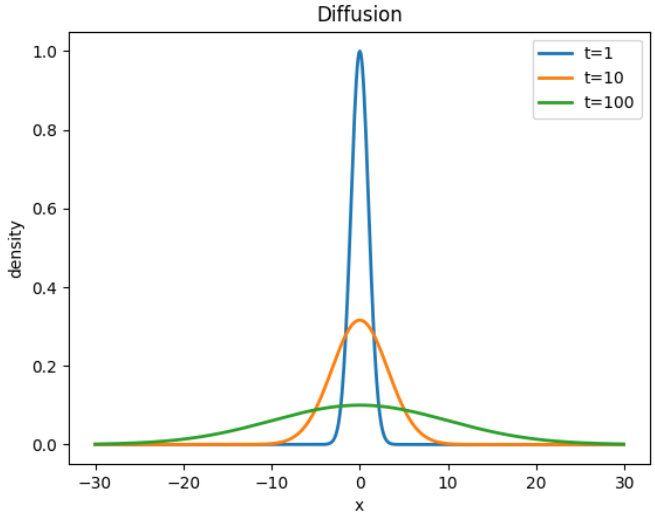

Figure 1:

Recall that for a distribution of random variables, the standard deviation is given by the root-mean-square:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2} = \sqrt{\langle x^2 \rangle} \tag{13}$$

For the random walk, this was

$$\sigma = \sqrt{\langle x^2 \rangle} = \sqrt{2Dt} \tag{14}$$

which matches the standard deviation of the density function. In both cases, $\sigma \sim t^{1/2}$. So if we execute random walks for many particles, we expect the distribution of positions to look roughly Gaussian.

Example: Random Walk in One Dimension, N Particles

So far, we have only considered a lattice, i.e. equal step sizes (1) in x, y, or z directions. What if we consider a random step size? For the example, we'll stick to one dimension, but let the step size be a random number between 0 and 1.

Example: Random Walk in One Dimension, Random Step Size

For the unit step size above, we found $\langle x_i^2 \rangle \approx t$. However, for the random step size, we see $\langle x_i^2 \rangle \approx t/3$. The position after n steps is the position at $n-1$ steps plus the step size Δx:

$$x(n) = x(n-1) \pm \Delta x \tag{15}$$

If we average this over many steps, we get

$$\langle x(n)\rangle = \langle x(n-1) \pm \Delta x\rangle \tag{16}$$

Since there is equal probability of moving left or right, the average of Δx is zero, and the average x is the same at every step, $\langle x(n)\rangle = \langle x(n-1)\rangle = \langle x(n-2)\rangle = ...\langle x(0)\rangle$. However, if we average x^2 over many steps, we have

$$\begin{aligned}\langle x^2(n)\rangle &= \langle (x(n-1) \pm \Delta x)^2\rangle \\ &= \langle x^2(n-1) \pm 2(\Delta x)x(n-1) + (\Delta x)^2\rangle\end{aligned} \tag{17}$$

The middle term is proportional to Δx and therefore averages to zero. But at each successive step, we add $(\Delta x)^2$ to the variance. Therefore, the average x^2 grows linearly in the number of steps:

$$\langle x^2(n)\rangle = n(\Delta x)^2 \tag{18}$$

When $\Delta x = \pm 1$, then $\langle x_i^2\rangle$ is proportional the number of steps, which we saw before. For the example with the random step size, the average step size-squared comes out to about 1/3. Therefore, $\langle x_i^2\rangle \approx t/3$. Therefore the diffusion constant is

$$\begin{aligned}2D &= \frac{1}{3} \\ D &= \frac{1}{6}\end{aligned} \tag{19}$$

Now let's model the drop of cream in the coffee cup. We'll do this in two dimensions, x and y. For this, we'll assume the particle can only take a step of unit length, now in one of four directions: $\pm x$ or $\pm y$. We'll start with 400 particles, all located near the center, in a 20x20 grid, $-10 < x < 10$ and $-10 < y < 10$. We will enforce a boundary (your coffee is in a container) so that the particle must stay in the square region $-100 < x < 100$ and $-100 < y < 100$. We'll take a picture of the particle locations at $t = 0$ and $t = 10^5$ (i.e. after 10^5 steps).

Example: Diffusion

Let's talk about diffusion in terms of entropy. Entropy provides a quantitative measure of the amount of disorder in a system. According to the second law of thermodynamics, entropy will either remain the same or increase with time. The initial state of the drop of cream in the cup of coffee is highly ordered and has a small value of entropy. As the particles spread out, the system becomes more disordered. The statistical definition of entropy is given by

$$S = -\sum_i P_i \ln P_i \tag{20}$$

where the sum is over all possible states and P_i is the probability of finding the system in state i. Let's divide our y vs x into an 8×8 grid. Each cell of the grid represents a state

i. P_i is the probability of finding a particle in this cell at any particular time. If we simulate n total particles, and w_i is the number of particles in cell i at a particular time, then $P_i = w_i/n$ is the probability of finding a particle in cell i. We can calculate this at any step in the evolution.

Example: Entropy

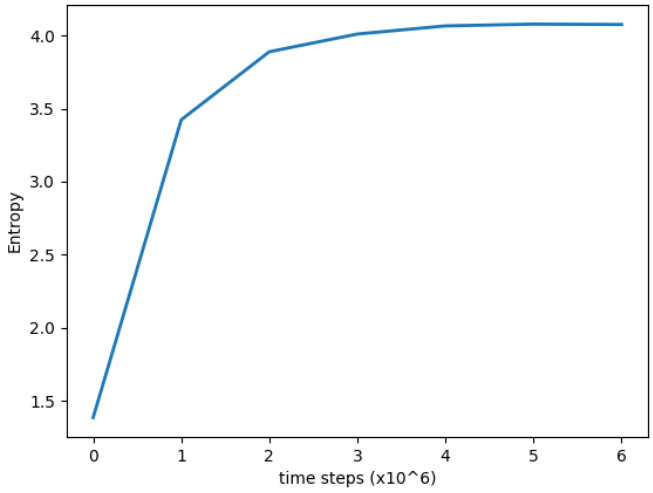

Figure 2:

Figure 2 shows the result of this calculation, at 7 steps in time. At $t = 0$, the system is in a highly ordered state, and the entropy is small. As time passes, entropy increases and levels off; this signals the system has reached equilibrium (i.e. the cream is mixed in with the coffee). The equilibrium condition is where all available states have equal probabilities.

3 Monte Carlo Simulation

Let's return to the coin toss, where the outcome is random.[1] Let's assume a fair coin, where the "true" probability of the coin landing tails is 50%. Assuming you didn't know the true probability, how would you go about measuring it? You could measure the probability of getting tails by tossing the coin many times in a row and counting how many times it comes up tails. The number of tails over the total number of tosses would be your measurement

[1]OK, OK, in principle, this is also a deterministic process - if you know all the related conditions, like the initial speed and angle of the toss, the spin, the atmospheric conditions in the vicinity of the coin etc, you could determine the outcome. But the initial conditions are essentially random (unless you spend a great amount of time perfecting your toss and controlling the coin toss environment), making the outcome random.

of the probability. How many times would you need to toss the coin to get a good measurement of the probability? See Table 1. Clearly, if you only flip the coin a few times, your measurement of the probability could be very inaccurate.

Number of tosses	Possible outcomes	Measured probability of tails
1	H	0
	T	1
2	HH	0
	HT	0.5
	TH	0.5
	TT	1
3	HHH	0
	HHT	0.33
	HTH	0.33
	THH	0.33
	HTT	0.67
	THT	0.67
	TTH	0.67
	TTT	1
4	HHHH	0
	HHHT	0.25
	HHTH	0.25
	HTHH	0.25
	THHH	0.25
	HHTT	0.5
	HTTH	0.5
	TTHH	0.5
	HTHT	0.5
	THTH	0.5
	THHT	0.5
	HTTT	0.75
	THTT	0.75
	TTHT	0.75
	TTTH	0.75
	TTTT	1

Table 1:

What is the probability that with n tosses, the coin lands tails k times? Each coin toss has 2 possible outcomes (heads or tails), so for n tosses, there are 2^n possible outcomes. There are $\binom{n}{k}$ ways to have k tails in n tosses. Therefore the probability of k tails in n tosses is:

$$P(k \text{ tails}) = \frac{\binom{n}{k}}{2^n} = \frac{n!}{k!(n-k)!}\frac{1}{2^n} \tag{21}$$

For the case of 4 tosses, the probability of 2 tails is

$$P(2 \text{ tails}) = \frac{4!}{2!(4-2)!}\frac{1}{2^4} = \frac{24}{4}\frac{1}{16} = \frac{3}{8} \tag{22}$$

as we saw in Table 1.

What can we expect with more tosses, say 100?

Example: Probability of landing k tails in n tosses

The probability of getting exactly 50 tails in 100 tosses is ∼0.08, which is small. But the probability of getting 51 tails is also ∼0.08, as well as the probability of getting 49 tails. Therefore the probability that measure a number *close to* 0.5 is much higher with 100 tosses than it was with 4 tosses.

Now let's simulate this experiment and look at some possible outcomes.

Example: Coin Toss, part 2

So we see that the measured probability approaches the true value as we increase the number of tosses. We also see that the ultimate result is not exactly the same in each trial. This is the result of what we call "statistical fluctuations" or "statistical uncertainty." The measurement will approach the true value, but not reach it exactly due to the randomness of the system. Note this is different from a measurement uncertainty. Our measurement is perfect in this case (we can always determine whether the coin is heads up or tails up). Even in the case of perfect measurement, there is still some fluctuation purely due to the random nature of the system.

What we have just performed is a Monte Carlo simulation. A Monte Carlo[2] simulation is any algorithm that relies on repeated random sampling to obtain numerical results. We make some assumptions about the probabilities associated with the system, then run some "pseudo-experiments" to see the results.

We can compare the Monte Carlo results to results from actual experiments to try to determine what the true probabilities are. We can also use Monte Carlo to determine how typical our real experimental results are. Let's see an example of that. Here, we will record the final measured probability after 2000 coin tosses, and look at the distribution of that number over many (1000) trials.

Example: Coin Toss, part 3

If we conducted a real experiment with 2000 coin tosses, and got a result of $p_{\text{meas}} = 0.52$, the results of our pseudo-experiments show us that this is a reasonable outcome for a true

[2]Named after the Monte Carlo Casino in Monaco

probability $p_{\text{true}} = 0.5$, i.e. within expected statistical fluctuations. However, if we measured a value of $p_{\text{meas}} = 0.57$, we could conclude that this is unlikely to be a statistical fluctuation from a true value of $p_{\text{true}} = 0.5$, and there might be another explanation (like the coin we used in the experiment is not a fair coin).

We've seen that Monte Carlo allows us to study the statistical fluctuations in a measurement. What if there is also a measurement uncertainty, in addition to statistical uncertainty? Let's say that you are not very good at counting, so that when counting the number of tosses that come up tails, you have a counting error of ± 25. Let's build this into our Monte Carlo simulation by modeling the number of tails counted using a Gaussian distribution with a mean of the true count and a standard deviation of 25.

Example: Coin Toss, part 4

This counting error causes additional spread in the possible measurements. Your counting error is an example of a "systematic uncertainty." (Systematic uncertainty is essentially any source of uncertainty that is not statistical.) Since we have a range for this uncertainty ± 25, we can model it with a probability. Note this is different from a "bias." A bias would be that you always count 5 more tails than there really are. (We would model this by adding 5 to the number of tails in each trial.)

Section 5, Exercise 3

4 Monte Carlo Integration

Let's say we have a square dart board that contains a circular region as shown in Figure 3.

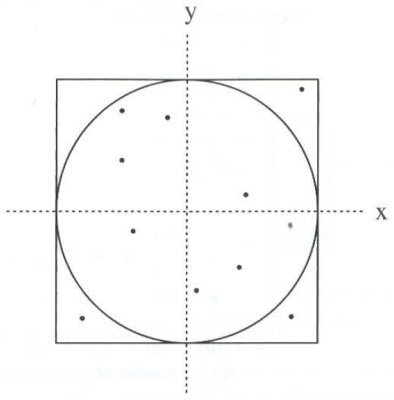

Figure 3:

You throw darts at the board, and all of your darts hit, but are randomly and uniformly distributed over the square surface. The probability that the dart lands in any particular region is proportional to the area of that region. Therefore, we can use this method to

calculate the area of the circle, given that we know the area of the square:

$$\begin{aligned}\frac{P_{\text{circle}}}{P_{\text{square}}} &= \frac{A_{\text{circle}}}{A_{\text{square}}} \\ P_{\text{circle}} &= \frac{A_{\text{circle}}}{A_{\text{square}}} \\ A_{\text{circle}} &= P_{\text{circle}} A_{\text{square}}\end{aligned} \tag{23}$$

where P is probability and A is area. The probability the darts hit the square, $P_{\text{square}} = 1$, because we assume all the darts hit the board. Let's assume the square is defined by $-1 < x < 1$ and $-1 < y < 1$. As we can see in the figure, the circle has a radius of 1, such that $x^2+y^2 = 1$. For one of our random points, if $|y| < \sqrt{1-x^2}$, then the point is inside the circle. The true area of the circle is of course $A_{\text{circle}} = \pi(1)^2 = \pi$.

Example: Area of a Circle

This method can be used to calculate the area under any function (i.e. the integral) as long as there is a way to check if a randomly generated point lies inside or outside the region of interest. In general, it is less accurate than other numerical integration methods we have seen, so it is most useful 1) if those other methods are not possible or 2) for multi-dimensional integrals. We will discuss examples of both cases.

To use this method, we generate N points in a region with known area A. If k points lie underneath the curve, then the approximation for the integral is given by

$$I \approx \frac{k}{N} A \tag{24}$$

where k/N is the probability of a point landing underneath the curve.

Consider the integral

$$\begin{aligned}I &= \int_0^2 f(x)dx \\ I &= \int_0^2 \sin^2\left[\frac{1}{x(2-x)}\right] dx\end{aligned} \tag{25}$$

The function is plotted in Figure 4. In the middle of the range, the function is well-behaved, but close to $x = 0$ and $x = 2$, the variations become rapid. Due to this rapid variation at the edges, our typical methods of calculating numerical integrals don't work well for this function.

Let's use the Monte Carlo method to calculate the integral. The function is contained within the rectangle given by $0 < x < 2$ and $0 < y < 1$, so we will use that area as our known area (with $A_{\text{box}} = (2)(1) = 2$). We will pick points randomly in this region, and calculate the fraction where $y < f(x)$. That fraction, times the known area of the box, gives the area underneath the curve.

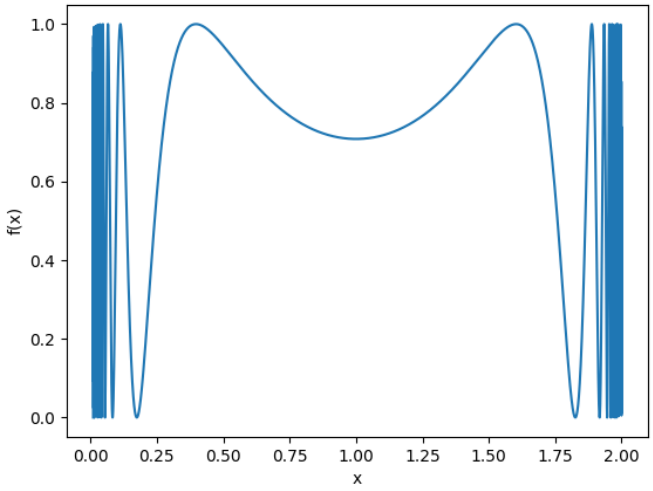

Figure 4:

Example: Monte Carlo Integration

What is the approximation error in this method? The probability that a point lies below the curve is $p = I/A$, where I is the integral of the function (area underneath the curve) and A is the known area in which our points are randomly generated. The probability a point lies above the curve is $(1-p)$. Therefore the probability that k points fall below and $N-k$ points fall above for a particular choice of k points is $p^k(1-p)^{N-k}$. There are $\binom{N}{k}$ ways to choose k points, so the total probability is

$$P(k) = \binom{N}{k} p^k (1-p)^{N-k} \tag{26}$$

This is the binominal distribution. The variance of this distribution is

$$\begin{aligned} \sigma_k^2 &= Np(1-p) \\ &= N\frac{I}{A}\left(1 - \frac{I}{A}\right) \end{aligned} \tag{27}$$

Given the relationship between k and I given in equation 24, the standard deviation in I is

given by

$$\begin{aligned}
\sigma_I &= \frac{\sigma_k}{N} A \\
&= \frac{\sqrt{N \frac{I}{A}\left(1 - \frac{I}{A}\right)}}{N} A \\
&= \frac{\sqrt{A^2 \frac{I}{A}\left(1 - \frac{I}{A}\right)}}{\sqrt{N}} \\
&= \frac{\sqrt{AI\left(1 - \frac{I}{A}\right)}}{\sqrt{N}} \\
\sigma_I &= \frac{\sqrt{I\left(A - I\right)}}{\sqrt{N}}
\end{aligned} \tag{28}$$

So the approximation error varies with N as $1/\sqrt{N}$. How does this compare with other numerical integration methods we've seen?

For the trapezoidal rule, the approximation error was of order Δx^2. For an integral from $x = a$ to $x = b$ with N slices, $\Delta x = (b-a)/N$, so that the approximation error varies with N as $1/N^2$. With Simposon's rule, the approximation error varies with N as $1/N^4$. So we see that the Monte Carlo method is not as accurate as even the trapezoidal rule, and the error does not improve as dramatically as we increase N. (For example, with $N = 25$, $\sigma \sim 1/5$ with the Monte Carlo method and $\sigma \sim 1/625$ with the trapezoidal rule. If we increase N by a factor of 4 to $N = 100$, the error Monte Carlo method only improves by a factor of 2 to $\sigma \sim 1/10$, while the trapezoidal rule error improves to $\sigma \sim 1/10000$, an improvement by a factor of 16.)

One slight improvement in the Monte Carlo method is called the "Mean Value Method." Suppose we want to integrate the function $f(x)$ from a to b:

$$I = \int_a^b f(x)dx \tag{29}$$

The average value of a function $f(x)$ in the range a to b is given by

$$\langle f \rangle = \frac{1}{b-a} \int_a^b f(x)dx \tag{30}$$

Therefore,

$$I = (b-a)\langle f \rangle \tag{31}$$

If we can estimate the average value of f, then we can estimate the integral. To estimate this average, we choose N random points between $x = a$ and $x = b$:

$$\langle f \rangle \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i) \tag{32}$$

Then the approximation for the integral is

$$I \approx (b-a)\langle f \rangle$$
$$\int_a^b f(x)dx \approx \frac{(b-a)}{N}\sum_{i=1}^{N} f(x_i) \tag{33}$$

This looks more similar to the other numerical integration methods we've studied. The integral is a weighted sum over function values. In this case, the weights are equal, but the points are chosen randomly.

Example: Mean Value Method

The approximation error for the mean value method also goes as $1/\sqrt{N}$, but it is slightly more accurate than the previous Monte Carlo method.

Section 5, Exercise 4

Now we turn to multi-dimensional integrals. To apply the numerical integration techniques we studied before, like the trapezoidal rule or Simpson's rule, we would divide *each axis* up into N slices. For example, for a 2D integral over x and y, we would pick a fixed value of y, do the x integral numerically using N slices, then repeat that for N values of y. So we end up with a calculation involving N^2 terms. For a 3D integral, we would have N^3 terms and so on. So for a multi-dimensional integral where N is large, the calculation time blows up. Using the Monte Carlo method, we can pick N points randomly in the multi-dimensional region, and get a pretty good approximation for the integral with much less calculation time.

We've actually already done an example of this when we found the area of a circle of radius 1. Let's reframe that problem as a 2D integral. Here's the analytical solution. (This integral is of course easier in polar coordinates, but let's stick with Cartesian coordinates.)

$$\begin{aligned} A_{\text{circle}} &= \int_{y=-1}^{y=1}\int_{x=-\sqrt{y^2-1}}^{x=\sqrt{y^2-1}} dxdy \\ &= \int_{y=-1}^{y=1}\left(\sqrt{y^2-1}-(-\sqrt{y^2-1})\right)dy \\ &= \int_{y=-1}^{y=1} 2\sqrt{y^2-1}dy \end{aligned}$$

Let $y = \sin z$, so that $dy = \cos z dz$, $y = -1$ corresponds to $z = -\pi/2$ and $y = 1$ corresponds

to $z = \pi/2$.

$$\begin{aligned} A_{\text{circle}} &= \int_{z=-\pi/2}^{z=\pi/2} 2\sqrt{\sin^2 z - 1}\cos z dz \\ &= 2\int_{z=-\pi/2}^{z=\pi/2} \sqrt{\cos^2 z}\cos z dz \\ &= 2\int_{z=-\pi/2}^{z=\pi/2} \cos^2 z dz \\ &= 2\left(\frac{z}{2} + \frac{\sin(2z)}{4}\right)_{z=-\pi/2}^{z=\pi/2} \\ &= 2\left(\frac{\pi/2}{2} + \frac{\sin\pi}{4} - \frac{-\pi/2}{2} - \frac{\sin(-\pi)}{4}\right) \\ A_{\text{circle}} &= \pi \end{aligned} \tag{34}$$

We've already done this with the Monte Carlo method, let's try it with the Mean Value method. To generalize the mean value method to two dimensions:

$$\int_{\mathcal{S}} f(x,y)dxdy \approx \frac{(b_x - a_x)(b_y - a_y)}{N}\sum_{i=1}^{N} f(x_i, y_i) \tag{35}$$

where $x = a_x$ and $x = b_x$ are the beginning and end points on the x axis, and $y = a_y$ and $y = b_y$ are the beginning and end points on the y axis. The points (x_i, y_i) are picked randomly in the region $\mathcal{S}$, which is the two dimension region over which we want to integrate.

For the circle we have

$$f(x,y) = \begin{cases} 1 & x_i, y_i \in \mathcal{S} \\ 0 & \text{otherwise} \end{cases} \tag{36}$$

where $\mathcal{S}$ is defined by $x^2 + y^2 \leq 1$. The factor in front of the sum is

$$(b_x - a_x)(b_y - a_y) = (1 - (-1))(1 - (-1)) = (2)(2) = 4 \tag{37}$$

So

$$\int_{\mathcal{S}} f(x,y)dxdy \approx \frac{4}{N}\sum_{i=1}^{N} f(x_i, y_i) \tag{38}$$

where $\mathcal{S}$ is the circle and (x_i, y_i) are points picked randomly in the circle.

Example: Circle with Mean Value Method

Now let's do the volume of a sphere or radius 1:

$$\int_{\mathcal{V}} f(x,y,z)dxdydz \approx \frac{(b_x - a_x)(b_y - a_y)(b_z - a_z)}{N}\sum_{i=1}^{N} f(x_i, y_i, z_i) \tag{39}$$

where the points (x_i, y_i, z_i) are picked randomly in the region $\mathcal{V}$, which is the three dimensional region over which we want to integrate.

For the sphere we have

$$f(x,y,z) = \begin{cases} 1 & x_i, y_i, z_i \in \mathcal{V} \\ 0 & \text{otherwise} \end{cases} \tag{40}$$

where $\mathcal{V}$ is defined by $x^2 + y^2 + z^2 \leq 1$. The factor in front of the sum is

$$(b_x - a_x)(b_y - a_y)(b_z - a_z) = (1-(-1))(1-(-1))(1-(-1)) = (2)(2)(2) = 8 \tag{41}$$

So

$$\int_{\mathcal{V}} f(x,y,z)dxdydz \approx \frac{8}{N}\sum_{i=1}^{N} f(x_i, y_i, z_i) \tag{42}$$

where $\mathcal{V}$ is the sphere and (x_i, y_i) are points picked randomly in the circle.

Example: Sphere with Mean Value Method

Let's also do the sphere with the Monte Carlo Method. The sphere is entirely contained within the cube defined by $-1 < x < 1$, $-1 < y < 1$, $-1 < z < 1$. We generate N points in the volume of the cube with known area A. If k points lie inside the sphere, then the approximation for the volume is given by

$$I \approx \frac{k}{N} V \tag{43}$$

where k/N is the probability of a point landing inside the volume of the sphere.

Example: Sphere with Monte Carlo Method