NERS 544 Monte Carlo Methods Lecture 2

Before we adapt the pdf's, cdf information into an MC code, we need to have a high quality source of random floating-point numbers, uniform on the range

$$0 \overset{?}{\leq} r \overset{?}{\leq} 1.000 \text{ (exactly)}$$

A complete understanding of how this is done is quite involved!

We must understand how integers and floating point numbers are represented in computers

L2-2

From the ENG 101 book

2.4.1 Whole numbers

Humans are usually quite awful at doing binary arithmetic. If you are doing such a calculation, you can check by converting the binary numbers into their decimal equivalents.

In general, to convert a whole binary number to a whole decimal number, consider a binary number of the form:

$$b_n b_{n-1} b_{n-2} \cdots b_2 b_1 b_0,$$

where the b_i's are either 0 or 1. We compute a decimal number using the following formula:

$$d_{10} = b_n \times 2^n + b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} \cdots b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0.$$

For example, the result of the two more difficult calculations above can be checked as follows:

$10110_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 2^4 + 2^2 + 2^1 = 16_{10} + 4_{10} + 2_{10} = 22_{10}$

is the result of one of the additions $[1111_2(15_{10}) + 111_2(7_{10})]$ above, while

$$1101001_2 = 2^6 + 2^5 + 2^3 + 2^0 = 64_{10} + 32_{10} + 8_{10} + 1_{10} = 105_{10}$$

is the result of one of the multiplications $[1111_2(15_{10}) \times 111_2(7_{10})]$ above. Note that the subscript "2" indicates a binary or base-2 number while the subscript "10" indicates a decimal number.

2.4.2 Real numbers

To convert a real(non-whole) binary number to a real decimal number, consider a binary number of the form:

$$b_n b_{n-1} b_{n-2} \cdots b_2 b_1 b_0 \; . \; b_{-1} b_{-2} \cdots b_{-(m-2)} b_{-(m-1)} b_{-m},$$

where the b_i's are either 0 or 1 and a period, ".", separates the whole part, b_i where $i \geq 0$ from the fractional part, b_i where $i < 0$. We compute a real decimal number using the following formula:

$$r_{10} = b_n \times 2^n + b_{n-1} \times 2^{n-1} \cdots b_1 \times 2^1 + b_0 \times 2^0 + b_{-1} \times 2^{-1} + b_{-2} \times 2^{-2} \cdots + b_{-(m-1)} \times 2^{-(m-1)} + b_{-(m)} \times 2^{-m}.$$

It is not a whole lot different from the decimal numbers you are used to. However, sometimes the formulae can look a little messy.

2.5 Binary integer arithmetic on computers

Computers, because of their comprehension of data in binary terms, naturally do their arithmetic using the base-2 representation. The circuitry within a computer is most easily set up this way. However, when it comes to base-2 arithmetic and computers, there is a subtle difference! For practical reasons, computers must deal with integers that are finite in length. 16 bits, representing 65,536 separate possibilities, used to be the norm. Today,

32 bits is common (4294967296 possibilities). Soon, 64 bits (about $18{\times}10^{18}$) will be the standard. Some computers nowadays permit hardware 64-bit integer arithmetic.

Let us consider 32-bit computers only—the most common type of hardware available today. In the course of doing an arithmetic calculation, any bits that are set beyond the 32-bit boundary are lost—they fall into the "bit bucket" and disappear.

2.5.1 Two's-complement integer arithmetic

A string of 32-bits can represent many things but we will focus on hexadecimal, unsigned and signed integers. These signed integers must also carry information as to the sign of the number. Historically, there have been several ways to do this. The "industry" has "settled" on "two's-complement arithmetic" because it simplifies the process of adding positive, negative or mixed numbers. In this scheme, to negate a signed integer, one takes the complement (change all the 1s to 0s and all the 0s to 1s) of the binary representation and adds 1 to it. This is seen in the table at the end of this chapter.

Let's give the notation $\bar{i}$ to the complement of the integer i. So, for any i, $\bar{i}+i = 11111111111111111111111111111111$. That is:

	any 32-bit pattern
+	complement of the above
=	11111111111111111111111111111111

and

	11111111111111111111111111111111
+	00000000000000000000000000000001
=	00000000000000000000000000000000

since the 33rd bit can not be set! In two's-complement arithmetic the expression $i - j$ is calculated as $i+\bar{j}+1$. So you see, computers can not really add and subtract, they can only add (and take complements)!

A few examples help illustrate the simplicity of the two's-complement scheme. Consider a 4-bit representation of the number $3_{10} = 0011_2$ and $4_{10} = 0100_2$. There are two tricks you should apply here: a) every time you see a negative sign in front of a bit pattern, i, to be used in a calculation, convert it to a positive sign, change i to $\bar{i}+1$ and do the binary addition, and, b) if a final result has a leading order (leftmost) bit set to 1, it is a negative number, so, extract the minus sign and convert the resultant bit pattern j to $\bar{j}+1$.

$$3_{10} + 4_{10} = 0011_2 + 0100_2 = 0111_2 = 7_{10}$$

$$3_{10} - 4_{10} = 0011_2 - 0100_2 = 0011_2 + 1100_2 = 1111_2 = -0001_2 = -1_{10}$$

$$-3_{10} - 4_{10} = -0011_2 - 0100_2 = 1101_2 + 1100_2 = 1001_2 = -0111_2 = -7_{10}$$

See how easily it all worked out! This is why the two's-complement approach has been adopted. In order to handle negative integers one can convert them to their two's-complement counterparts and add. Otherwise, special circuitry would have to be developed for handling negative numbers and that would be less efficient.

2's compl $\tilde{b}+1$	compl $\tilde{b}$	Binary b	Hex	Unsigned Decimal	Signed Decimal
0000	1111	0000	0	0	0
1111	1110	0001	1	1	1
1110	1101	0010	2	2	2
1101	1100	0011	3	3	3
1100	1011	0100	4	4	4
1011	1010	0101	5	5	5
1010	1001	0110	6	6	6
1001	1000	0111	7	7	7
1000	0111	1000	8	8	-8
0111	0110	1001	9	9	-7
0110	0101	1010	A or a	10	-6
0101	0100	1011	B or b	11	-5
0100	0011	1100	C or c	12	-4
0011	0010	1101	D or d	13	-3
0010	0001	1110	E or e	14	-2
0001	0000	1111	F or f	15	-1

2.6 32-bit Binary, Hexadecimal, Unsigned and Signed Integers

32-bit computers can represent $2^{32} = 4,294,672,296$ different things but ONLY $4,294,672,296$ different things. Now we consider the conventional assignments to "unsigned" and "signed" integers. There is also the hexadecimal representation that is associated with every 4-bit sequence, as indicated in the table below.

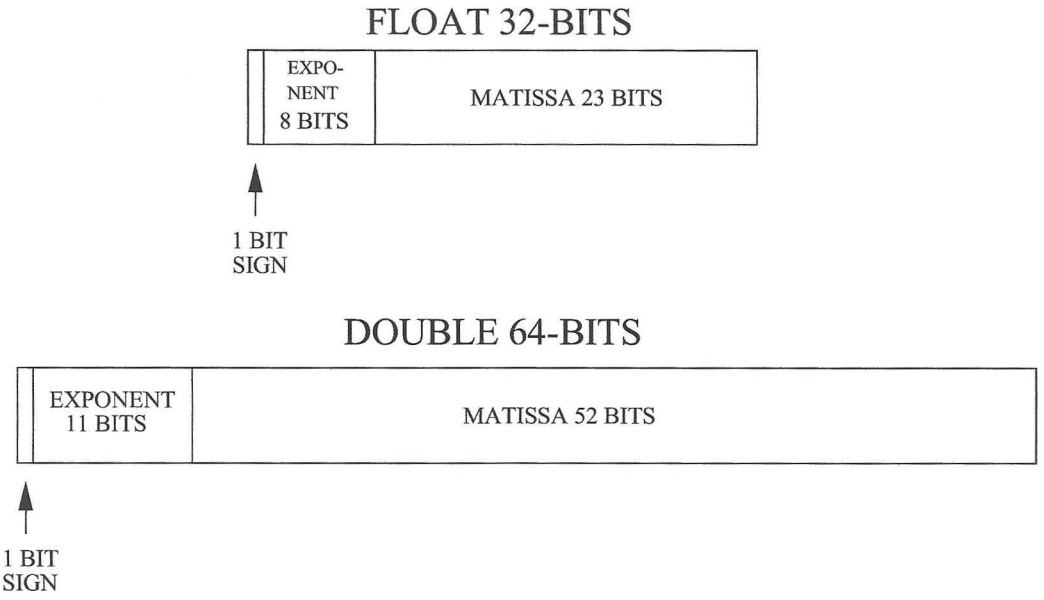
The unsigned integers associate the decimal number 0 to a string of 32 0 bits. The rest are formed by the addition by 1 in either binary or decimal. The maximum unsigned integer, therefore, is $4,294,672,295$. Adding a one to this causes all the 32 bits to go back to zero, exactly as if you had "cycled" the odometer in your car! It's the same principle. There is no 33rd bit, so you can imagine it just falls off the end.

The signed integers have the same sequence as the unsigned integers up to $2^{31} - 1 = 2147483647$ which is the bit pattern 01111111111111111111111111111111. Adding a one to this gives a bit pattern of 10000000000000000000000000000000 which is interpreted as

$-2^{31} = -2147483648$. Note that these two are complements of each other and that they add up to
11111111111111111111111111111111, which has the interpretation "-1", consistent with the "two's-complement" scheme described above.

Binary	Hexadecimal	Unsigned Int	Signed Int
00000000000000000000000000000000	00000000	0	0
00000000000000000000000000000001	00000001	1	1
00000000000000000000000000000010	00000002	2	2
00000000000000000000000000000011	00000003	3	3
00000000000000000000000000000100	00000004	4	4
.	.	.	.
.	.	.	.
.	.	.	.
00000000000000000000000000001111	0000000F	15	15
00000000000000000000000000010000	00000010	16	16
00000000000000000000000000010001	00000011	17	17
.	.	.	.
.	.	.	.
.	.	.	.
01111111111111111111111111111110	7FFFFFFE	2147483646	2147483646
01111111111111111111111111111111	7FFFFFFF	2147483647	2147483647
10000000000000000000000000000000	80000000	2147483648	-2147483648
10000000000000000000000000000001	80000001	2147483649	-2147483647
.	.	.	.
.	.	.	.
.	.	.	.
11111111111111111111111111111100	FFFFFFFC	4294967292	-4
11111111111111111111111111111101	FFFFFFFD	4294967293	-3
11111111111111111111111111111110	FFFFFFFE	4294967294	-2
11111111111111111111111111111111	FFFFFFFF	4294967295	-1

This is a representation of the difference between `float` and `double`:



Consider the following code called `precisionFloat.cpp`:

```
//File: precisionFloat.cpp
#include <iostream>

using namespace std;

float precision(void)
{  float one = 1.0f, e = 1.0f, onePlus;
   int counter = 0;
   do
   {  counter = counter + 1;
      e = e/2.0f;
      onePlus = one + e;
   }while(onePlus != one);
   cout << "Converged after " << counter << " iterations\n";
   return e;
}

int main(void)
{  cout << "Float resolution = " << precision() << "\n";
   return 0;
}
```

When c is $\neq 0$, it is called a

linear congruential (pseudo) rng (LCRNG)

when $c=0$, a

multiplicative congruential (pseudo) rng
(MCRNG)

For good a's (the magic #)
the sequence length is

$2^N/4$ about 10^9 for 32-bit
for an MCRNG

2^N for an LCRNG

Note that, a good LCRNG
will produce <u>every</u> possible integer
that is possible over its sequence

After the sequence is exhausted,
it repeats, reproducibly.

For that reason it is ill advised to
use more than 10% of the
sequence.

Congruential (pseudo) random number generators

All Monte Carlo codes use (and should use)

pseudo random number generators (RNGs)

They are pseudo because the sequences of numbers are algorithmically generated, and therefore, reproducible. This is most desirable for debugging purposes.

All congruential rngs have the form

$$I_{n+1} = \mathrm{mod}\left(I_n * a + c, 2^N\right)$$

I_n is the nth random (signed or unsigned) integer. N is the computer word size, typically 32 or 64 bits

a is a "magic number". Guidelines exist, but their quality is determined experimentally.

a is a fixed odd number.

Otherwise, correlations set in.

e.g. the 1/2 half is anti-correlated with the 2nd half.

Example of MC & LC rng's

< demos >

It would take about 400,000 days to cycle the 64-bit LCRNG!

About 2 days on the most powerful supercomputers today

Long sequence RNGs

Marsaglia's subtract-with-borrow

Sequence length 2^{144}

< demo >

Long sequence rng's are really all long word length LCRNGs! (Proof)

All (L/M)CRNGs suffer from the "spectral" problem, 1st noticed by George Marsaglia

If one seeds an n-dimensional cube with an (L/M)rng, the points will line up in (n-1)-dimensional hyperplanes!

Algorithm (pseudo code)

```
for i = 1:N
    pick x between 0 & 1
        y   "
        z   "   "   "
end
```

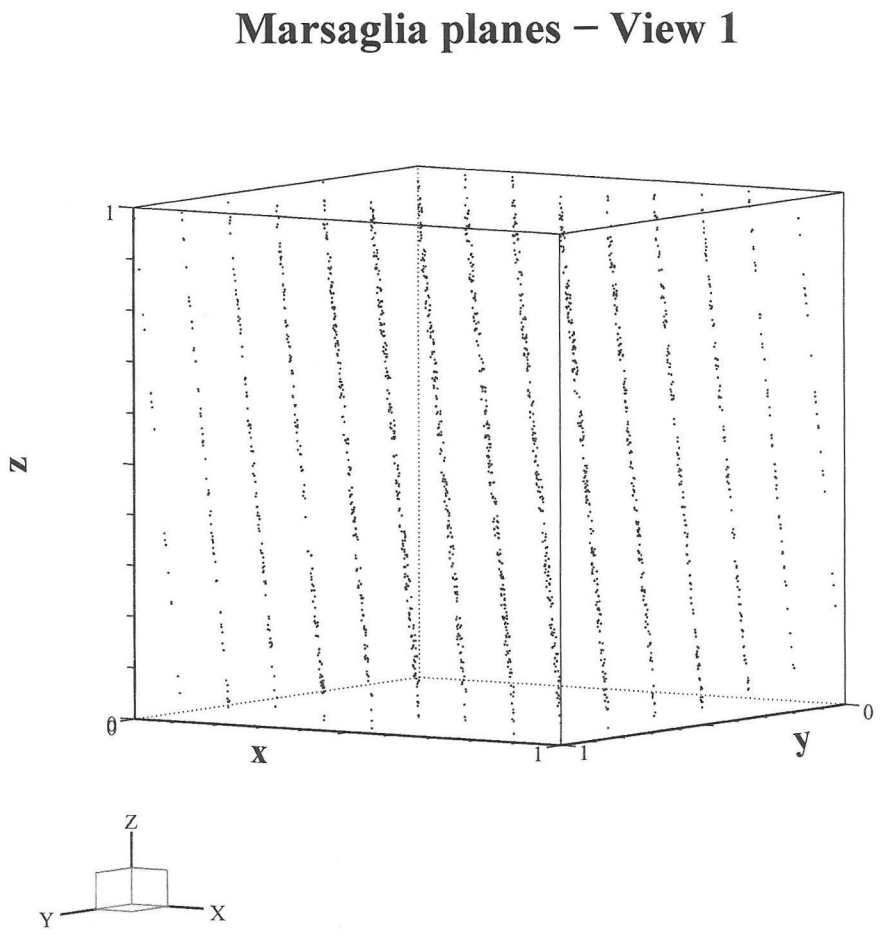


Figure 3.1: The gathering of random numbers into two-dimensional planes when a three-dimensional cube is seeded.

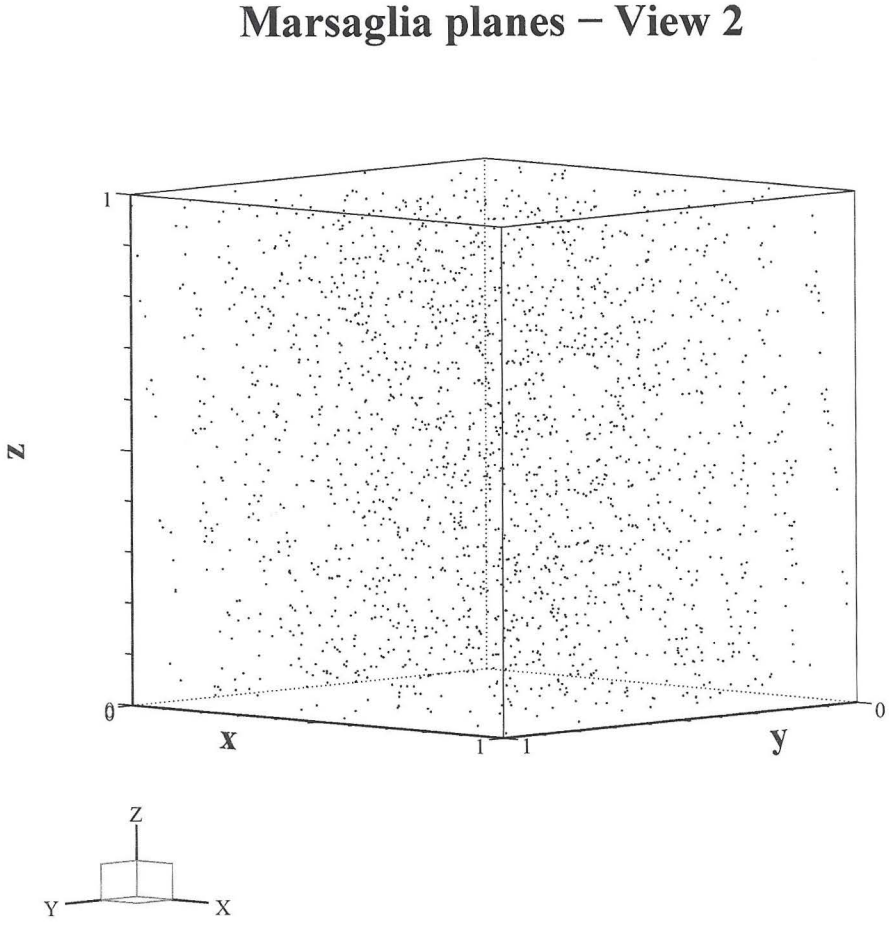


Figure 3.2: The identical data in Figure 3.1 but rotated by 10° about the z-axis.

Some final notes on random # generation

If you use C/C++, DO NOT use its rand() function.
This was implemented well before our modern understanding of rngs.

DO USE mcrng32, lcrng32, mcrng64 provided in the online notes.
They are MUCH better studied

If you use MATLAB,
DO USE its rand function

Within Matlab

>> help rand

for details on how to use/restart rand.

Also, google "rand algorithm in Matlab"

The sequence length is $2^{19937} - 1$

(!) that should be long enough
based on the "Mersenne twister" google it!

For those anticipating MC applications in parallel computing environments

google "SPRNG Mascagni"
(favored by nukes)
or "random number generator CERN"
(favored by high energy physicists)

demo

<cons.cpp>
<tt>
<rand0,1,2,3>
<randomPi>
<randomIn>

The rest of this lecture is straight out of the book
Chapter 4.

Chapter 4

Sampling Theory

Sir,
In your otherwise beautiful poem (The Vision of Sin) there is a verse which reads

> *"Every moment dies a man,*
> *every moment one is born."*

Obviously, this cannot be true and I suggest that in the next edition you have it read

> *"Every moment dies a man,*
> *every moment* $1\frac{1}{16}$ *is born."*

Even this value is slightly in error but should be sufficiently accurate for poetry.

...Charles Babbage (in a letter to Lord Tennyson)

Now that we have tackled the essentials of elementary probability theory and random number generation, it is now time to connect the two and demonstrate how random numbers may be employed to sample from probability distributions.

We will consider three kinds of sampling techniques, the direct approach, the rejection technique and the mixed method that combines the two. Then, we we go through a small catalogue of examples.

4.1 Invertible cumulative distribution functions (direct method)

A typical probability distribution function is shown in Figure 4.1. It is defined over the range

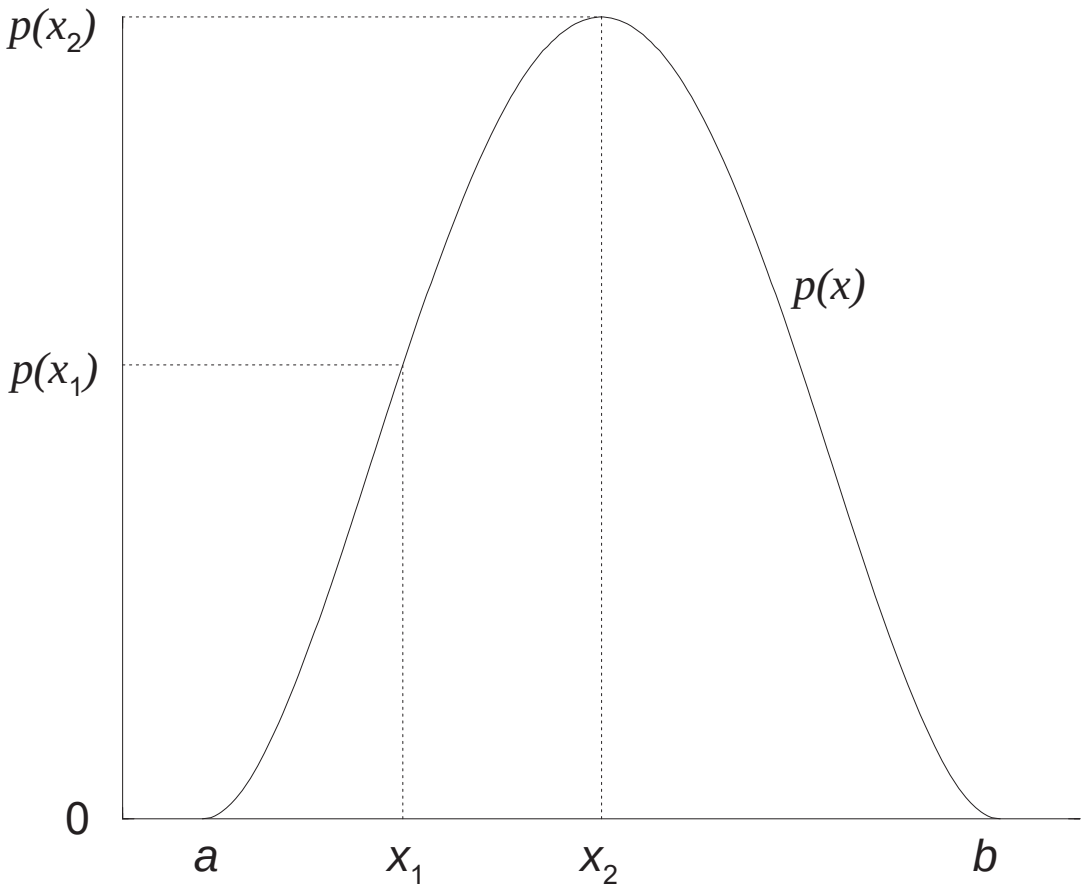


Figure 4.1: A typical probability distribution.

$[a, b]$ where neither a nor b are necessarily finite. A probability distribution function *must* have the properties that it is integrable (so that one can normalise it by integrating it over its entire range) and that it is non-negative. (Negative probability distributions are difficult to interpret.)

We now construct its cumulative probability distribution function:

$$c(x) = \int_a^x \mathrm{d}x' \; p(x') \tag{4.1}$$

and assume that it is properly normalised, *i.e.* $c(b) = 1$. The corresponding cumulative probability distribution function for our example is shown in Figure 4.2.

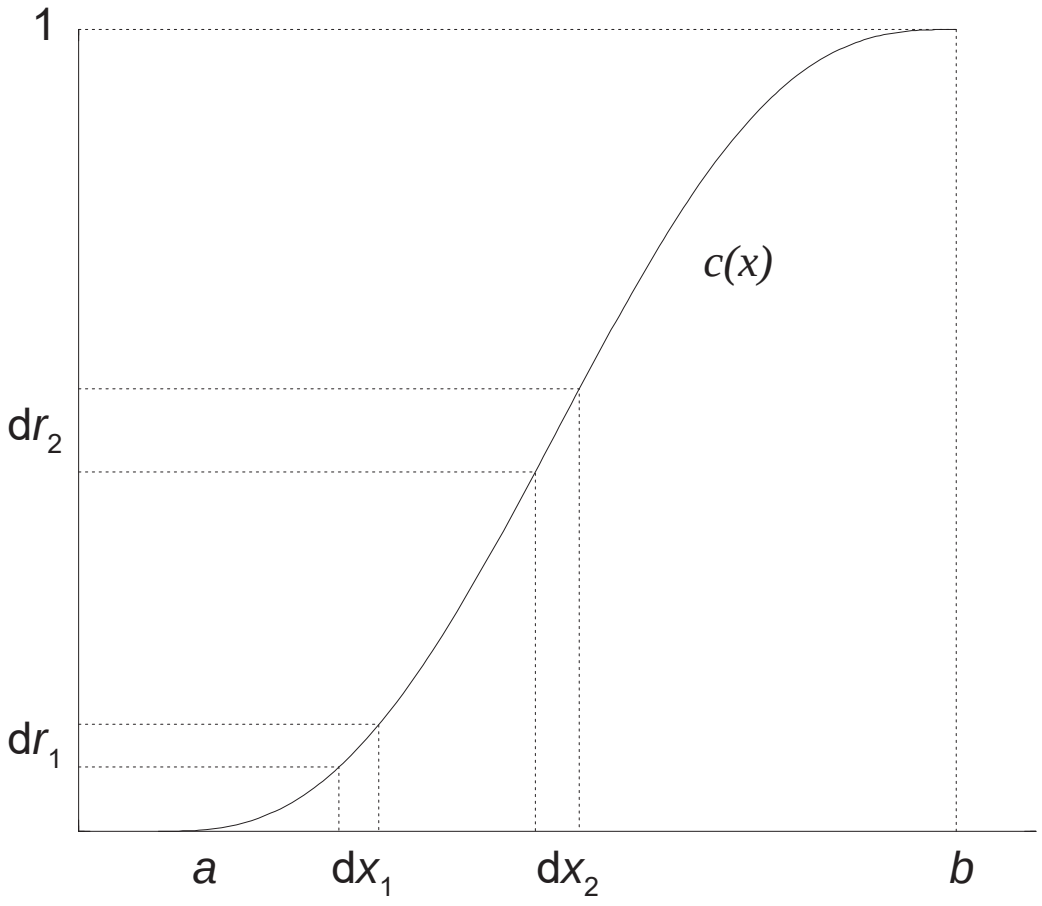


Figure 4.2: The cumulative probability distribution obtained by integrating the probability distribution function in Figure 4.1.

By its definition, we can map the cumulative probability distribution function onto the range of random variables, r, where $0 \leq r \leq 1$ and r is distributed uniformly. That is, $r = c(x)$.

Now consider two equally spaced intervals $\mathrm{d}x_1$ and $\mathrm{d}x_2$, differential elements in x in the vicinity of x_1 and x_2. Using some elementary calculus we see that:

$$\frac{\mathrm{d}r_1}{\mathrm{d}r_2} = \frac{(\mathrm{d}/\mathrm{d}x)c(x)|_{x=x_1}}{(\mathrm{d}/\mathrm{d}x)c(x)|_{x=x_2}} = \frac{p(x_1)}{p(x_2)} \ . \tag{4.2}$$

We can interpret this as meaning that, if we select many random variables in the range [0,1], then the number that fall within $\mathrm{d}r_1$ divided by the number that fall within $\mathrm{d}r_2$ is equal to the ratio of the probability distribution at x_1 to x_2. (Recall the interpretation of the probability distribution as given in Chapter 2.)

Having mapped the random numbers onto the cumulative probability distribution function, we may invert the equation to give:

$$x = c^{-1}(r) \ . \tag{4.3}$$

All cumulative probability distribution functions that arise from properly defined probability distribution functions are invertible, numerically if not analytically[1].

Then, by choosing r's randomly over a uniform distribution and substituting them in the above equation, we generate x's according to the proper probability distribution function.

Example:

As we will discuss in Chapter 8, the distance, z, to an interaction is governed by the well-known probability distribution function:

$$p(z)\mathrm{d}z = \mu e^{-\mu z}\mathrm{d}z \ , \tag{4.4}$$

where μ is the interaction coefficient. The valid range of z is $0 \leq z < \infty$ and this probability distribution function is already properly normalised. The corresponding cumulative probability distribution function and its random number map is given by:

$$r = c(z) = 1 - e^{-\mu z} \ . \tag{4.5}$$

Inverting gives:

$$z = -\frac{1}{\mu}\log(1-r) \ . \tag{4.6}$$

If r is uniformly distributed over $[0,1]$ then so is $1-r$. An equivalent form of the above equation (that saves one floating point operation) is:

$$z = -\frac{1}{\mu}\log(r) \ . \tag{4.7}$$

[1]However, there are subtleties. Sampling the probability distribution function is a differentiation process. Thus, if a cumulative probability distribution function is constructed numerically, differentiation leads to minor difficulties. For example, if a cumulative probability distribution function is represented by a set of linear splines, differentiation will lead to a step-wise continuous probability distribution function.

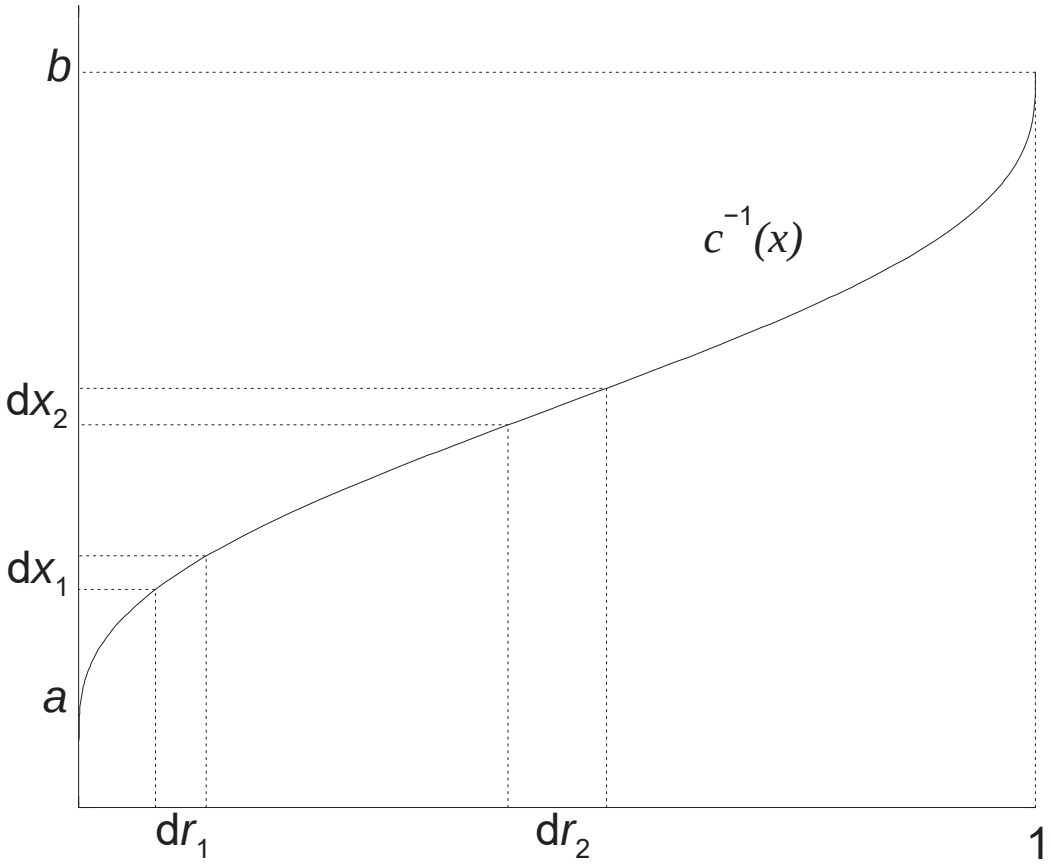


Figure 4.3: The inverse cumulative probability distribution obtained by inverting the cumulative probability distribution function in Figure 4.2.

This is exactly the form used to calculate a particle's distance to an interaction in all Monte Carlo codes. Recall, however, that if the random number generator provides an exact zero as a possibility, the sampling implied by Equation 4.7 will cause a floating-point error. (It is best to have a random number generator that does not provide exact zero's unless you really need them for something specific.)

4.2 Rejection method

While the invertible cumulative probability distribution function method is always possible, at least in principle, it is often impractical to calculate $c()^{-1}$ because it may be exceedingly complicated mathematically or contain mathematical structure that is difficult to control. Another approach is to use the rejection method.

In recipe form, the procedure is this:

1. Scale the probability distribution function by its maximum value obtaining a new distribution function, $f(x) = p(x)/p(x_{\max})$, which has a maximum value of 1 which occurs at $x = x_{\max}$ (see Figures 4.4 and 4.5). Clearly, this method works only if the probability distribution function is not infinite anywhere and if it is not prohibitively difficult to determine the location of the maximum value. If it is not possible to determine the maximum easily, then overestimating it will work as well, but less efficiently.

2. Choose a random number, r_1, uniform in the range $[0, 1]$ and use it to obtain an x which is uniform in the probability distribution function's range [a,b]. (To do this, calculate $x = a + (b - a)r_1$.) (Note: This method is restricted to finite values of a and b. However, if either a or b are infinite a suitable transformation may be found to allow one to work with a finite range. *e.g.* $x \in [a, \infty)$ may be mapped into $y \in [0, 1)$ via transformation $x = a[1 - \log(1 - y)]$.)

3. Choose a second random number r_2. If $r_2 < p(x)/p(x_{\max})$ (region under $p(x)/p(x_{\max})$ in Figure 4.5) then accept x, else, reject it (shaded region above $p(x)/p(x_{\max})$ in Figure 4.5) and go back to step 2.

The efficiency of the rejection technique is defined as:

$$\epsilon = \frac{1}{p(x_{\max}(b - a))} \int_a^b \mathrm{d}x \; p(x) \; . \tag{4.8}$$

This is the ratio of the expected number of random numbers pairs that are accepted to the total number of pairs employed.

Remarks:

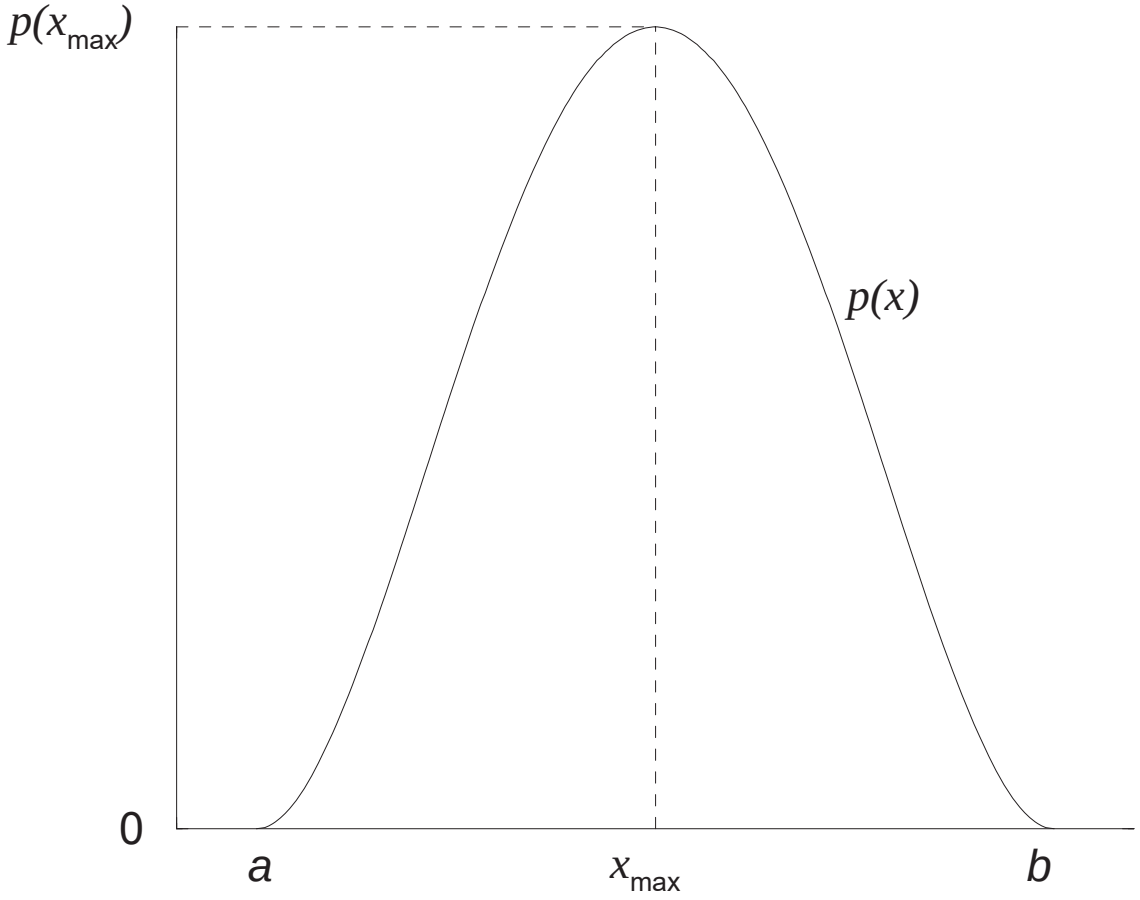


Figure 4.4: A typical probability distribution.

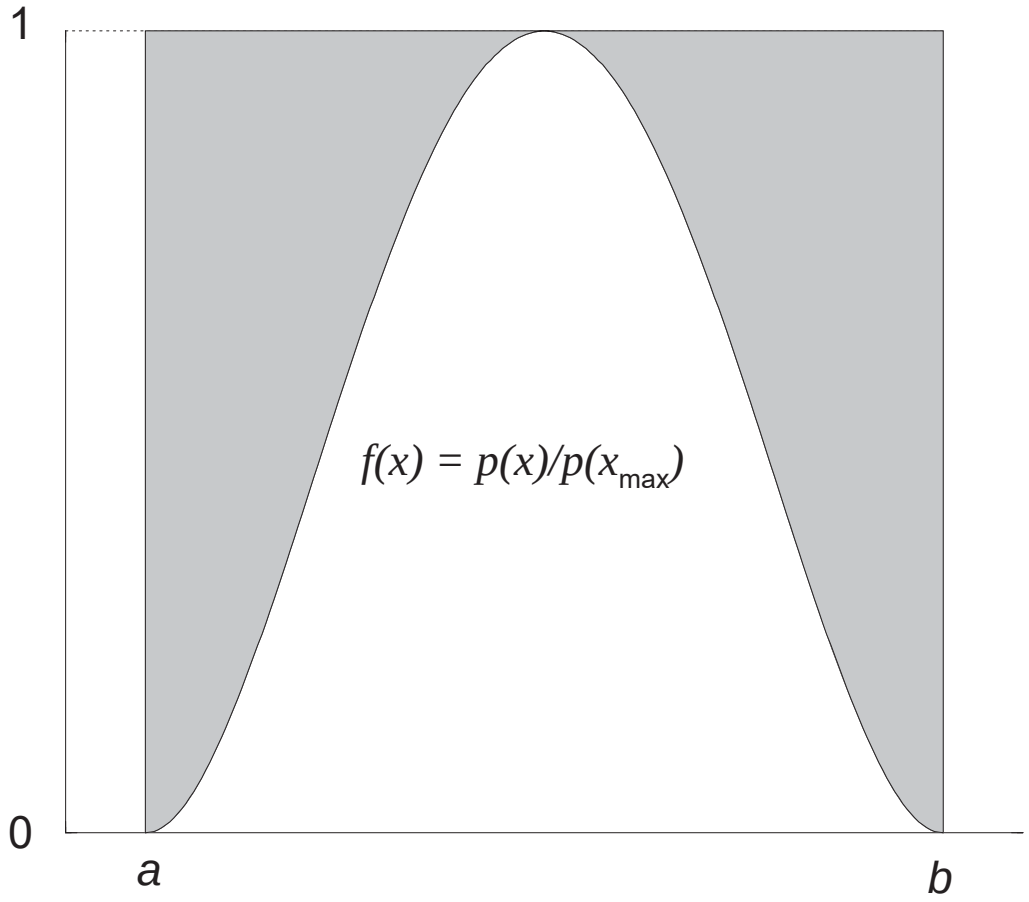


Figure 4.5: The probability distribution of Figure 4.4 scaled for the rejection technique.

This method will result in x being selected according to the probability distribution function. Some consider this method "crude" because random numbers are "wasted" unlike the invertible cumulative probability distribution function method. It is particularly wasteful for "spiky" probability distribution functions. However, it can save computing time if the $c()^{-1}$ is very complicated. One has to "waste" many random numbers to use as much computing time as in the evaluation of a transcendental function!

4.3 Mixed methods

As a final topic in elementary sampling theory we consider the "mixed method", a combination of the previous two methods.

Imagine that the probability distribution function is too difficult to integrate and invert, ruling out the direct approach without a great deal of numerical analysis, and that it is "spiky", rendering the rejection method inefficient. (Many probability distributions have this objectionable character.) However, imagine that the probability distribution function can be factored as follows:

$$p(x) = f(x)g(x) \tag{4.9}$$

where $f(x)$ is an invertible function that contains most of the "spikiness", and $g(x)$ is relatively flat but contains most of the mathematical complexity. The recipe is as follows:

1. Normalise $f(x)$ producing $\tilde{f}(x)$ such that $\int_a^b \mathrm{d}x\ \tilde{f}(x) = 1$.
2. Normalise $g(x)$ producing $\tilde{g}(x)$ such that $\tilde{g}(x) \leq 1\ \forall\ x \in [a, b]$.
3. Using the direct method described previously, choose an x using $\tilde{f}(x)$ as the probability distribution function.
4. *Using this* x, apply the rejection technique using $\tilde{g}(x)$. That is, choose a random number, r, uniformly in the range $[0, 1]$. If $\tilde{g}(x) \leq r$, accept x, otherwise go back to step 3.

Remarks:

With some effort, any mathematically complex, spiky function can be factored in this manner. The art boils down to the appropriate choice of $\tilde{f}(x)$ that leaves a $\tilde{g}(x)$ that is nearly flat. For two recent examples of this method as applied to a production-level code, see References [BR86] and [BMC89].

The mixed method is also tantamount to a change in variables. Let

$$p(x)\mathrm{d}x = f(x)g(x)\mathrm{d}x = (\tilde{f}(x)\mathrm{d}x)\left(\int_a^b \mathrm{d}x\ f(x)\right)g(x)\ , \tag{4.10}$$

where $\tilde{f}(x)$ is now a properly normalized probability distribution function. Employing $\tilde{f}(x)$ as the function for the direct part, we let

$$u = c(x) = \int_a^x \tilde{f}(x')\mathrm{d}x' \ , \tag{4.11}$$

be a transformation between x and u. Note the limits of u, $0 \leq u \leq \int_a^b \tilde{f}(x')\mathrm{d}x' = 1$. By definition, the inverse exists so that $x = c^{-1}(u)$. As well $\mathrm{d}u = f(x)\mathrm{d}x$. Thus, we can rewrite Equation 4.10 as:

$$p(x)\mathrm{d}x = \left(\int_a^b \mathrm{d}x \ f(x) \right) g(u)\mathrm{d}u \ , \tag{4.12}$$

which eliminates $f(x)$ through a change in variables. Thus, one can sample $g(u)$ using rejection (or some other technique) and relate the selected u to x through the inverse relation $x = c^{-1}(u)$.

If the rejection technique is employed for $g(x)$, then the efficiency of is calculated in the same way as in Equation 4.8.

4.4 Examples of sampling techniques

4.4.1 Circularly collimated parallel beam

The normalised probability distribution in this case is:

$$p(\rho, \phi) \ \mathrm{d}\rho \ \mathrm{d}\phi = \frac{1}{\pi \rho_0^2} \rho \ \mathrm{d}\rho \ \mathrm{d}\phi \qquad 0 \leq \rho \leq \rho_0 \quad 0 \leq \phi \leq 2\pi \tag{4.13}$$

where ρ is the cylindrical radius, ρ_0 is the collimation radius and ϕ is the azimuthal angle. $\rho \mathrm{d}\rho \mathrm{d}\phi$ is a differential surface element in cylindrical coordinates. This is a separable probability distribution of the form:

$$p(\rho, \phi) \ \mathrm{d}\rho \ \mathrm{d}\phi = \mathrm{d}p_1(\rho) \ \mathrm{d}p_2(\phi) \tag{4.14}$$

where:

$$p_1(\rho) \ \mathrm{d}\rho = \frac{2}{\rho_0^2} \rho \ \mathrm{d}\rho \qquad 0 \leq \rho \leq \rho_0 \tag{4.15}$$

and

$$p_2(\phi) \ \mathrm{d}\phi = \frac{1}{2\pi} \ \mathrm{d}\phi \qquad 0 \leq \phi \leq 2\pi \tag{4.16}$$

Direct method

The cumulative probability distribution functions in this case are:

$$c_1(\rho) = \frac{2}{\rho_0^2}\int_0^{\rho} \mathrm{d}\rho'\rho' = \frac{\rho^2}{\rho_0^2} \tag{4.17}$$

$$c_2(\phi) = c_2(\phi) = \frac{1}{2\pi}\int_0^{\phi} \mathrm{d}\phi' = \frac{\phi}{2\pi} \tag{4.18}$$

Inverting gives:

$$\rho = \rho_0\sqrt{r_1} \tag{4.19}$$

$$\phi = 2\pi r_2 \tag{4.20}$$

where the r_i are random numbers on the range $[0, 1]$.

The code segment that would produce accomplish this looks like:

```
rho = rho_0  * sqrt(rng())
phi = 2e0 * pi * rng()
x = rho * cos(phi)
y = rho * sin(phi)
```

where `rng()` is a function that return a random number uniformly on the range $[0, 1]$ [or $(0, 1]$ or $[0, 1)$ or $(0, 1)$].

Rejection method

In this technique, a point is chosen randomly within the square $-1 \leq x \leq 1; -1 \leq y \leq 1$. If this point lies within a circle with unit radius the point is accepted and the x and y values scaled by the collimation radius, ρ_0. The code segment that would accomplish this looks like:

```
1         x = 2e0 * rng() - 1e0
          y = 2e0 * rng() - 1e0
          IF (x**2 + y**2 .gt. 1e0) goto 1
      x = rho_0 * x
      y = rho_0 * y
```

Which is better?

Actually, both methods are equivalent mathematically. However, one or the other may have advantages in execution speed depending on other factors in the application. If the geometry is not cylindrically symmetric or all the scoring that is done does not make use of the inherent cylindrical symmetry, then the rejection method is about twice as fast as the direct method because the trigonometric functions are not employed in the rejection method.

If the geometry is cylindrically symmetric and the scoring takes advantage of this symmetry, then the direct method is about 2–3 times faster because symmetry reduces the calculation to:

```
x = rho_0  * sqrt(rng())
y = 0
```

Many computers now have hardware square root capabilities. With this capability the direct method may be advantageous, whether or not one makes use of the cylindrical symmetry.

4.4.2 Point source collimated to a planar circle

The normalised probability distribution in this case is:

$$p(\theta,\phi)\mathrm{d}\theta\mathrm{d}\phi = \frac{\mathrm{d}\phi}{2\pi}\frac{\sin\theta\ \mathrm{d}\theta}{1-\cos\theta_0} \qquad 0 \le \theta \le \theta_0 \quad 0 \le \phi \le 2\pi \tag{4.21}$$

where θ is the polar angle and ϕ is the azimuthal angle. $\sin\theta\ \mathrm{d}\theta\ \mathrm{d}\phi$ is a differential solid angle element in spherical coordinates. θ_0 is the collimation angle. In terms of the distance to the collimation plane z_0 and the diameter of the collimation circle on this plane ρ_0, $\cos\theta_0 = z_0/\sqrt{z_0^2+\rho_0^2}$.

This is a separable probability distribution of the form:

$$p(\theta,\phi)\mathrm{d}\theta\mathrm{d}\phi = p_1(\theta)\mathrm{d}\theta\ p_2(\phi)\mathrm{d}\phi \tag{4.22}$$

where:

$$p_1(\theta)\mathrm{d}\theta = \frac{\sin\theta\mathrm{d}\theta}{1-\cos\theta_0} \qquad 0 \le \theta \le \theta_0 \tag{4.23}$$

and

$$p_2(\phi)\mathrm{d}\phi = \frac{1}{2\pi}\mathrm{d}\phi \qquad 0 \le \phi \le 2\pi \tag{4.24}$$

The cumulative probability distribution functions in this case are:

$$c_1(\theta) = \frac{1}{1-\cos\theta_0}\int_0^\theta \sin\theta'\mathrm{d}\theta' = \frac{1-\cos\theta}{1-\cos\theta_0} \tag{4.25}$$

$$c_2(\phi) = c_2(\phi) = \frac{1}{2\pi}\int_0^{\phi} \mathrm{d}\phi' = \frac{\phi}{2\pi} \tag{4.26}$$

Inverting gives:

$$\cos\theta = 1 - r_1[1 - \cos\theta_0] \tag{4.27}$$

$$\phi = 2\pi r_2 \tag{4.28}$$

where the r_i are random numbers on the range $[0, 1]$.

The code segment that would accomplish this looks like:

```
cos_theta = 1e0 - rng() * (1e0 - cos_theta_0)
theta     = acos(cos_theta)
sin_theta = sin(theta)

phi = 2e0 * pi * rng()

u = sin_theta * cos(phi) ! u is sin(theta)*cos(phi),
                         ! the x-axis direction cosine
v = sin_theta * sin(phi) ! v is sin(theta)*sin(phi),
                         ! the y-axis direction cosine
w = cos_theta            ! w is cos(theta),
                         ! the z-axis direction cosine

x = z_0 * u/w            ! x = z_0 * tan(theta)*cos(phi)
y = z_0 * v/w            ! y = z_0 * tan(theta)*sin(phi)
```

In terms of the cylindrical coordinates on the collimation plane, Equation 4.27 becomes:

$$\frac{z_0}{\sqrt{\rho^2 + z_0^2}} = 1 - r_1\left[1 - \frac{z_0}{\sqrt{\rho_0^2 + z_0^2}}\right] \tag{4.29}$$

which yields a value for ρ on the collimation plane.

In the small angle limit, $\theta_0 \longrightarrow 0$, the circularly collimated parallel beam result should be recovered. If one employs the small angle approximation, $\rho \ll z_0$ and $\rho_0 \ll z_0$, Equation 4.29 obtains the result of Equation 4.19, *i.e.* $\rho = \rho_0\sqrt{r_1}$.

4.4.3 Mixed method example

Consider the probability function:

$$p(x)\mathrm{d}x = Ne^{-x^2}\frac{2x\mathrm{d}x}{(1+x^2)^2} \qquad 0 \le x < \infty\ , \tag{4.30}$$

where N is the normalization factor such that $\int_0^\infty p(x)\mathrm{d}x = 1$. Although $p(x)$ is integrable analytically[2], it can not be inverted analytically. Therefore, we consider the "spiky" part that we can integrate analytically:

$$f(x)\mathrm{d}x = \frac{2x\mathrm{d}x}{(1+x^2)^2}\mathrm{d}x \qquad 0 \le x < \infty \ , \tag{4.32}$$

which can be integrated directly,

$$r = c(x) = 1 - \frac{1}{1+x^2} \ , \tag{4.33}$$

and inverted,

$$x = \sqrt{\frac{r}{1-r}} \ . \tag{4.34}$$

This is equivalent to a the change of variables,

$$u = 1 - \frac{1}{1+x^2} \qquad x = \sqrt{\frac{u}{1-u}} \ , \tag{4.35}$$

and we must now sample,

$$g(x)\mathrm{d}x = \exp\left(-\frac{u}{1-u}\right)\mathrm{d}u \qquad 0 \le u \le 1 \ . \tag{4.36}$$

If we apply rejection to $g(x)$ directly, it can be shown that the "efficiency", $\epsilon = 0.404$.

Interestingly enough, we can choose to do this example the other way! We can choose as our direct function:

$$f(x)\mathrm{d}x = 2xe^{-x^2}\mathrm{d}x \qquad 0 \le x < \infty \ , \tag{4.37}$$

which can be integrated directly,

$$r = c(x) = 1 - e^{-x^2} \ , \tag{4.38}$$

and inverted,

$$x = \sqrt{-\log(1-r)} \ . \tag{4.39}$$

[2]The cumulative probability function can be written

$$c(x) = 1 - \frac{e^{-x^2} + e\,\left(1+x^2\right)\,Ei(-1-x^2)}{(1+x^2)\,\left(1+e\,Ei(-1)\right)} \tag{4.31}$$

where $Ei(z)$ is the exponential integral [AS64].

This is equivalent to a the change of variables,

$$u = 1 - e^{-x^2} \qquad x = \sqrt{-\log(1-u)} \ , \tag{4.40}$$

and we must now sample,

$$g(x)\mathrm{d}x = \frac{1}{[1-\log(1-u)]^2 \mathrm{d}u} \qquad 0 \leq u \leq 1 \ . \tag{4.41}$$

This approach has the same efficiency as the previous approach. However, it is more costly because is involves the use of more transcendental functions.

4.4.4 Multi-dimensional example

Consider the joint probability function:

$$p(x,y)\ \mathrm{d}x\ \mathrm{d}y = (x+y)\ \mathrm{d}x\ \mathrm{d}y \qquad 0 \leq x,y \leq 1 \ . \tag{4.42}$$

The marginal probability in x is:

$$m(x) = \int_0^1 \mathrm{d}y\ (x+y) = x + \frac{1}{2} \ , \tag{4.43}$$

the conditional probability of y given x is:

$$p(y|x) = \frac{p(x,y)}{m(x)} = \frac{x+y}{x+\frac{1}{2}} \ , \tag{4.44}$$

so that

$$p(x,y) = m(x)p(y|x) \ . \tag{4.45}$$

First we sample the marginal probability distribution in x. The cumulative distribution function and its associated random number map is:

$$r_1 = c(x) = \int_0^x \mathrm{d}x' \ \left(x' + \frac{1}{2}\right) = \frac{x^2}{2} + \frac{x}{2} \ , \tag{4.46}$$

which is a quadratic relation that can be inverted to give:

$$x = \frac{-1 + \sqrt{1+8r_1}}{2} \ . \tag{4.47}$$

The choice of the plus sign in the inversion of the quadratic relation was made based on the having $x = 0$ when $r_1 = 0$ and $x = 1$ when $r_1 = 1$.

Now that x is determined, we form the conditional cumulative probability distribution, $c(y|x)$, and its associated random number mapping:

$$r_2 = c(y|x) = \int_0^y \mathrm{d}y \; p(y|x) = \frac{y^2 + 2xy}{2x + 1} \; , \tag{4.48}$$

which itself can be inverted using quadratic inversion:

$$y = -x + \sqrt{x^2 + r_2(2x + 1)} \; , \tag{4.49}$$

which again involved a choice of sign based upon the expected limits, $y = 0$ when $r_2 = 0$ and $y = 1$ when $r_2 = 1$. For intermediate values of r_2, y depends upon the choice of x.