03 February 2025 – PHYS 6260
Prof. John Wise
Newman Chapter 10: Random Processes and Monte Carlo Methods

There are some random processes in physics, such as the quantum processes of radioactive decay or electrons decaying into the electronic ground state. Because they are random, the exact time of the decay cannot be predicted. We know the probability per unit time of the decay, and we can use that information in a computation.

On the other hand, there are non-random processes that can be modeled as random. For example, Brownian motion of a particle through air seems like a random process, but if we know the exact positions and velocities of the gas molecules, we could calculate the exact motion of the particle. However, this would be computationally intractable, but we will treat them as random processes, which works surprisingly well computationally.

10.1: Random Numbers

Random number generators

As a basis for any random process, we need to know how to generate (pseudo-)random numbers. We can use integer arithmetic (with well chosen factors) to produce some number from a starting number. Consider the equation:

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

that takes some integer x as an input and returns x', and a, c, and m are integer constants. We can then take x' and plug it into the equation again and repeat as needed. This will give a sequence of apparently random numbers. This type of random number generator is known as *linear congruent*. The following code shows an example of one that generates $N = 100$ numbers.

```
import matplotlib.pyplot as plt

N = 100
a = 1664525
c = 1013904223
m = 4294967296
x = 1
results = []

for i in range(N):
    x = (a*x+c)%m
    results.append(x)
```

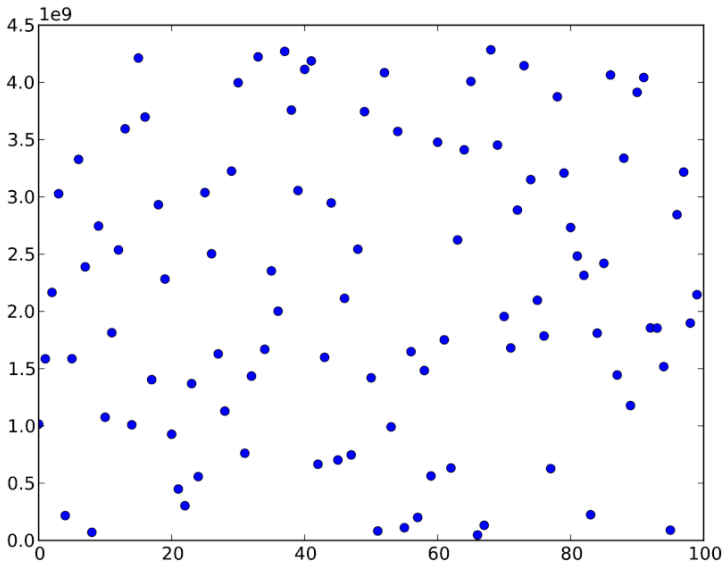

Figure 1: Output of the linear congruential random number generator

```
plt.plot(results,"o")
plt.show()
```

Figure 1 shows the resulting plot. There are a few points to notice.

1. The sequence is not random but deterministic, depending on the starting value of x.
2. The values will always be between 0 and $m - 1$.
3. The values of a, c, and m are carefully chosen, and other choices might result in some biases in the number distribution. For instance, c is prime; a only has three factors, and $m = 2^{32}$.

In general, linear congruential generators produce pretty bad random numbers because there start to be correlations between successive numbers. For high quality calculations, there needs to be no to very little correlation between successive draws, striving for true randomness. The most widely-used generator in physics calculations is known as the *Mersenne twister*, which is complicated to code but is computationally cheap and is the default random number generator in `numpy.random`. There are many routines in this module, which can be found at http://docs.scipy.org/doc/numpy/reference/routines.random.html.

We will be using the function `numpy.random.random()` the most in this class, which generates a uniformly distributed random number between 0.0 and 1.0. It takes an optional argument of the size of returned random number array. If no argument is given, then the output will be a scalar. If the argument is `(3,10)` for instance, the routine will return a 3×10 array.

Random number seeds

For any random number generator to be deterministic, it needs to be "seeded" with an initial number, just like $x = 1$ in the example provided earlier. Once the seed has been planted, then the whole sequence of numbers will be determined. Having a deterministic generator is extremely important in physics simulations because they need to be reproducible. In `numpy`, the generator can be (optionally) seeded with the routine `numpy.random.seed()` that takes the seed as its only argument.

Probabilities and biased coins

In various physics calculations, there are instances where an event has a probability p to occur. For example, we could have a particle that has a 20% chance to move and remain stationary otherwise. This is known as the "toss of a biased coin" by statisticians. It is very straightforward to determine whether this event happens by drawing a random number between 0 and 1; for example,

```
from numpy.random import random
if random() < 0.2:
    print("Heads")
else:
    print("Tails")
```

Example 10.1: Decay of an isotope

Radioactive isotopes have some probability per unit time to decay into another isotope. One example is ^{208}Tl (thallium) that decays into stable ^{208}Pb (lead) with a half-life of 183.2 s. We can mimic the randomness of radioactive decay by using random numbers.

On average, the number N of original atoms will exponentially decrease over time according to

$$N(t) = N(0)2^{-t/\tau}, \tag{2}$$

where τ is the half-life. One minus the surviving fraction $N(t)/N(0)$ is the decayed fraction, and it also represented the probability $p(t)$ that a single atom has decayed,

$$p(t) = 1 - 2^{-t/\tau} \tag{3}$$

We can simulate this random processes by dividing $N = 1000$ atoms into two sets: thallium and lead. Initially, it is all thallium, and we advance the system with a timestep of 1 s in which we decide how many atoms decay into lead. Here is a program that performs this calculation for 1000 s and plots the atom population as a function of time.

```
import numpy as np
from numpy.random import random
import matplotlib.pylab as plt

# Constants
NTl = 1000              # Number of thallium atoms
NPb = 0                 # Number of lead atoms
tau = 3.053*60          # Half life of thallium in seconds
h = 1.0                 # Size of time-step in seconds
p = 1 - 2**(-h/tau)     # Probability of decay in one step
tmax = 1000             # Total time

# Lists of plot points
tpoints = np.arange(0.0,tmax,h)
Tlpoints = []
Pbpoints = []

# Main loop
for t in tpoints:
    Tlpoints.append(NTl)
    Pbpoints.append(NPb)

    # Calculate the number of atoms that decay
    #
    # To be completed in class
    #

# Make the graph
plt.plot(tpoints,Tlpoints)
plt.plot(tpoints,Pbpoints)
plt.xlabel("Time")
plt.ylabel("Number of atoms")
plt.show()
```

Figure 2 shows the evolution of both populations, and the overall exponentially decay/rise is captured well. The noise in the trends come from the inherent randomness of the process.

Non-uniform random numbers

So far, we have only inspected uniformly distributed random numbers, but in most cases, random physical processes have some non-uniformity to them, favoring some variable, such as energy, position, or velocity. Let's go back to the previous example with radioactive decay. We showed that a single atom has a probability to decay in time t of $1 - 2^{-t/\tau}$. Therefore,

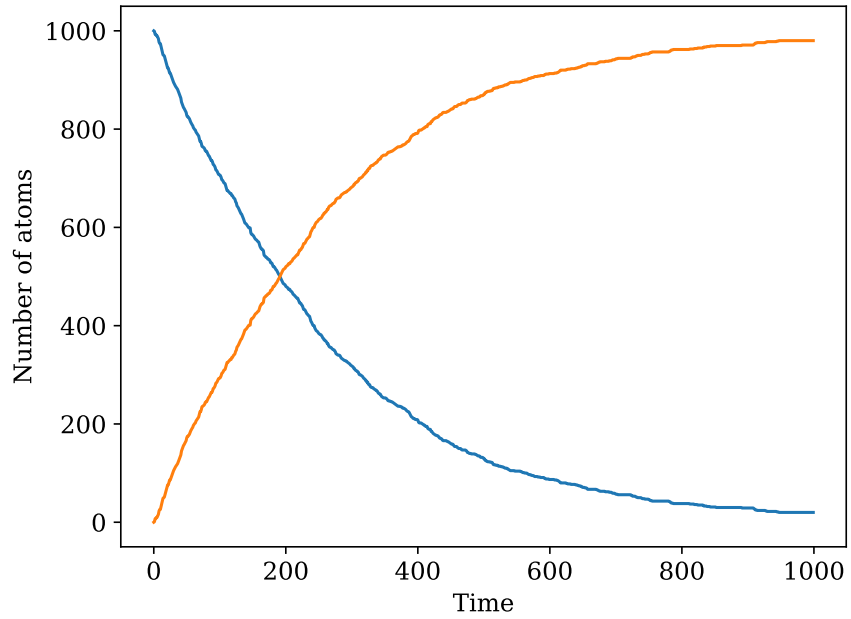

Figure 2: Decay of a sample of radioactive atoms

the probability to decay within some time interval dt is

$$1 - 2^{-dt/\tau} = 1 - \exp\left(-\frac{dt}{\tau}\ln 2\right) = \frac{\ln 2}{\tau}dt, \tag{4}$$

where we neglect terms with order dt^2 and higher. Now we can ask what is the decay probability between the times t and $t + dt$. For an atom to decay within that interval, it must survive until time t, which has a probability of $2^{-t/\tau}$. So the total probability $P(t)dt$ of decay during this interval is

$$P(t)dt = 2^{-t/\tau}\frac{\ln 2}{\tau}dt \tag{5}$$

This is a clear example of a non-uniform probability distribution, in which case, atoms have a higher probability of decaying earlier than later.

With a probability distribution as a function of time, we don't have to step forward in time, like we did in Example 10.1. We can simply draw random numbers from the $P(t)$ distribution to determine the decay times of N atoms. We can do so by generating a set of uniform random numbers and turn them into a non-uniform distribution through the *transformation method*.

Suppose that we have a set of random numbers z drawn from a distribution with a probability density $q(z)$. We also have a function $x = x(z)$, so when z is a random number then x is also a random number but with a different probability density $p(x)$. Our goal is to choose the function $x(z)$ so that x has the distribution we desire.

The probability of generating a number between x and $x + dx$ is equal to the probability of generating a value z within the corresponding interval,

$$p(x)dx = q(z)dz \tag{6}$$

The most common situation is that we generate a uniform distribution of random numbers between 0 and 1, so that $q(z) = 1$ in that range and zero everywhere else. Integrating both sides to some value $x(z)$, we find

$$\int_{-\infty}^{x(z)} p(x')\,dx' = \int_0^z dz' = z \tag{7}$$

If we can solve this equation for $x(z)$, then we're finished. However, it's not always possible to integrate and solve for the function $x(z)$.

But let's look at a case where we can always solve for it. Suppose that we have an exponential probability density,

$$p(x) = \mu e^{-\mu x}, \tag{8}$$

where μ is some constant, and notice that the indefinite integral of $p(x)$ is normalized to unity. We can insert this expression into Equation (7) to find

$$\mu \int_0^{x(z)} e^{-\mu x'} dx' = 1 - e^{-\mu x} = z, \tag{9}$$

and solving for x, we get

$$x = -\frac{1}{\mu} \ln(1 - z) \tag{10}$$

Now all that's needed to generate an exponential distribution is to draw uniform random numbers z and plug them into the equation above.

Gaussian random numbers

In many fields of physics, processes often generate Gaussian distributions,

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right), \tag{11}$$

where the standard deviation σ controls the width of the distribution and the pre-factor normalizes it to unity. One prime example is the Maxwellian velocity distribution in one component. Applying this probability density to Equation (7), we obtain

$$\frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{x} \exp\left(-\frac{x^2}{2\sigma^2}\right) dx = z. \tag{12}$$

This is a case where we cannot analytically solve this integral. However, there is a workaround to this problem.

Imagine that we have two independent random numbers x and y, both drawn from a Gaussian distribution with the same standard deviation σ. The probability that the point falls in some element $dx\,dy$ at point (x, y) is

$$p(x)\,dx\,p(y)\,dy = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)\,dx\,dy. \tag{13}$$

Notice that it has some circular symmetry, so we can re-express this in polar coordinates. Making the necessary substitutions ($x^2 + y^2 \to r^2$ and $dxdy \to rdrd\theta$) and separating the variables, we obtain

$$p(r,\theta)drd\theta = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{r^2}{2\sigma^2}\right)rdrd\theta \tag{14}$$

$$p(r)dr \times p(\theta)d\theta = \frac{r}{\sigma^2}\exp\left(-\frac{r^2}{2\sigma^2}\right)dr \times \frac{d\theta}{2\pi} \tag{15}$$

Now we can generate random numbers in r and θ with the appropriate distributions and then transform them into Cartesian coordinates. If we need an odd number of Gaussian random numbers, we can just ignore one of them. Generating a random θ value is easy since it's uniformly distributed between 0 and 2π, i.e. $p(\theta) = 1/2\pi$. For the radial component, we have

$$p(r) = \frac{r}{\sigma^2}\exp\left(-\frac{r^2}{2\sigma^2}\right) \tag{16}$$

and inserting this into Equation (7) we get

$$\frac{1}{\sigma^2}\int_0^r \exp\left(-\frac{r^2}{2\sigma^2}\right)rdr = 1 - \exp\left(-\frac{r^2}{2\sigma^2}\right) = z. \tag{17}$$

Solving for r,

$$r = \sqrt{-2\sigma^2\ln(1-z)}, \tag{18}$$

where recall that z is a uniformly distributed random number. With random values of r and θ, we convert back into Cartesian coordinates: $x = r\cos\theta$, $y = r\sin\theta$.

Example 10.2: Rutherford Scattering

Rutherford scattering occurs when a positively charged particles, such as an α-particle, is deflected by an atom by the Coulomb force, as shown in Figure 3. The scattering angle θ obeys

$$\tan\tfrac{1}{2}\theta = \frac{Ze^2}{2\pi\epsilon_0 Eb}, \tag{19}$$

where Z is the atomic number, e is the electron charge, E is the kinetic energy of the incident particle, and b is the impact parameter (see the figure).

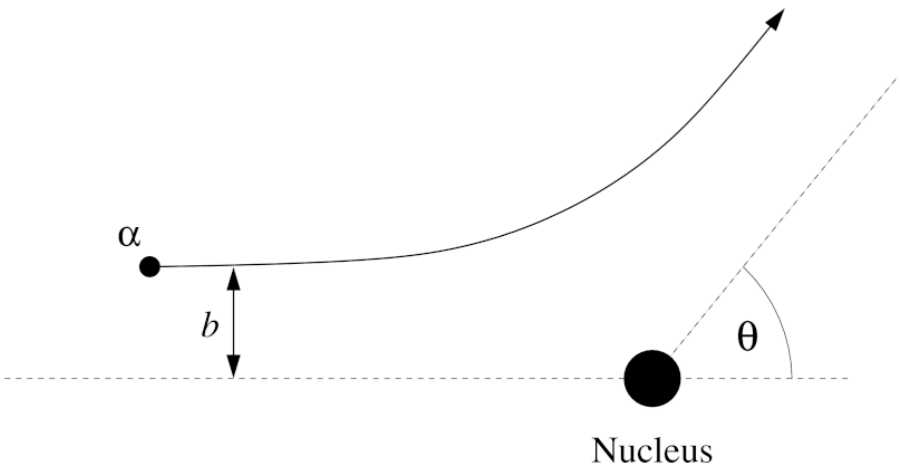

Figure 3: Rutherford scattering

In this example, we will consider a beam of α-particles each with an energy of 7.7 MeV. The beam has a Gaussian profile in both x and y directions with a spread $\sigma = a_0/100$, where a_0 is the Bohr radius. It is fired at a gold atom. Our goal is to calculate the fraction of α-particles that back-scatter, i.e. $\theta > \pi/2$. We can plug this scattering angle into the equation above to determine the critical impact parameter

$$b = \frac{Ze^2}{2\pi\epsilon_0 E} \tag{20}$$

below which the particles will be back-scattered. The program below simulates this process with 10^6 α-particles.

```
from math import sqrt,log,cos,sin,pi
from numpy.random import random

# Constants
Z = 79
e = 1.602e-19
E = 7.7e6*e
epsilon0 = 8.854e-12
a0 = 5.292e-11
sigma = a0/100
N = 1000000

# Complete in-class
# Function to generate two Gaussian random numbers
# Optional: Use array arithmetic to speed-up
def gaussian():
    x = y = 0
```

```
    return x,y

# Main program
count = 0
for i in range(N):
    # Complete in-class
    # For every particle, calculate a random position (x,y) and determine
    # whether it's back-scattered (b < some critical radii)

print("%d particles were reflected out of %d" % (count,N))
```

When I ran it, it returned `1597 particles were reflected out of 1000000`. You will find a slightly different value because of the randomness of the process. In class, we will vectorize this sample program so that much larger samples can be analyzed.