21 Nov 23 - Activity: Introduction to Random Processes

So far we have focused on mostly deterministic systems. These are systems that are typically described by regular ODEs and PDEs; think the SHO or the Wave Equation. In these systems, the game is to analytically determine or estimate the trajectory, field, or wave given the set of equations that describe it. But many processes in nature are random or have a random element ot them.

How do we model the decay of nuclear isotopes? The gas atoms in a container? Even things like when your bus will arrive.

Each of these processes has a randomness to it and how we model that randomness and what we can do depends a lot on the nature of the randomness. From a mathematical standpoint, what is the distribution of possible events, or the underlying probability distribution are key questions to ask in making these models. We will get there, but we will start by getting intuition from a place you have probably experienced: rolling dice.

```
import numpy as np
import matplotlib.pyplot as plt
import random as rand
import numpy as np
```

The Random Library

If you have not used `random`, please read this section because it has some implementation things that are useful.

Random numbers generated by a computer are not random, so it's important to know how to use them. They aren't truly random because a computer must generate them with an algorithm and thus we can potentially determine every number we'd receive if we knew the algorithm. There's tons of research into how to generate good random numbers and there's precious few ways. The main way right now is the Mersenne Twister method. Testing the quality of a random number generator is beyond the scope of this class, but a useful exercise if you are doing research into random processes.

Seeding the number generator

But because it's an algorithm, we can control the sequence of random numbers we get. That is we can tell the random number generator to create random numbers for us in the same way. This helps because we can use them and test them over and over before running a full simulation. See below on how to generate random integers. Re-run to observe what happens.

```
start = 1
stop = 10
```

```python
a = rand.randint(start,stop)
print('a =', a)
rand.seed(42)
b = rand.randint(start,stop)
print('b =', b)
rand.seed(18)
c = rand.randint(start,stop)
print('c =', c)
rand.seed(42)
d = rand.randint(start,stop)
print('d =', d)
```

```
a = 5
b = 2
c = 3
d = 2
```

Creating a uniform distribution

One of the simplest distributions to generate is one that you are likely most familiar with - a fair die or pair of dice. In the case of a single fair die, each outcome is equally likely and distinct from each other. That is rolling a one and rolling a six are different, distinct outcomes - and we can say that for each outcome.

Below, write a short piece of code (maybe as a function), that rolls one N-sided die as many times as you like and returns all the outcomes. **Be careful using rand.seed() here because you can just an array of all the same numbers!**

Plot a histogram of the outcomes; vary the sides of the die and the number of times it rolls.

Can you come up with a scheme that quantifies how close your distribution is to uniform?

```python
## your code here
```

Make a biased die

Adjust your program so that you get even numbered rolls twice as frequently as odd rolls. This is the basis for baking in a probability function into our models. So do try to this. **Make a histogram of the results of the biased die**

```python
## Your code here
```

Micro vs Macro states

In many random processes, the results are not distinct. Think about rolling two dice (colors are green and white) instead. How many ways can you get a sum of 2 vs a sum of 6? The state of the two dice where we indicate the number of each die can be called a "microstate". The state of the sum can be called a "macrostate."

So as an example consider the macrostate with a sum of 4. This can be constructed of out 3 potential microstates. First the green die could have a 1 and the white one a 3, the green could have a 2 and the white a 2, and then the green could have a 3 and the white a 1. But the result I care about is not the individual die, but the sum. Regardless I get 4.

- Microstate - tracks individual constituent states as if they were unique
- Macrostate - a group of microstates that share something

Modify or use your code above to roll two fair die and count their sum

Plot the resulting histogram of macrostates. Vary the number of sides and number of rolls. What do you notice?

```
## your code here
```

Poisson Processes

Many events in physics can be represented by a Poisson Distribution, which results from events that have the following properties:

- The occurrence of one event does not affect the probability that a second event will occur. That is, events occur independently.
- The average rate at which events occur is independent of any occurrences. For simplicity, this is usually assumed to be constant, but may in practice vary with time.
- Two events cannot occur at exactly the same instant; instead, at each very small sub-interval, either exactly one event occurs, or no event occurs.

Oftentimes, it's the average rate (think lifetime of an isotope) that we think about as helping us decide. If it's true that the rate is roughly constant and the events are independent, then this modeling can work reasonably well. So an event can be random, but the average result should be relatively simple (e.g., a single average decay time), then we can consider the events Poisson distributed.

Decay of an isotope

Consider a collection of ^{208}Tl that decays into ^{208}Pb with a half-life of 3.053 minutes. Note that any one atom decays randomly at a random time, but on average, we see that $1/e$ atoms decay in 3.053 minutes (this is characteristic of a Poisson Process).

The number of atoms N that will remain in our sample (that is still be Thallium 208) will fall off exponentially over time (we could observe that from data):

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

where τ is the half-life. The fraction of remaining Thallium 208 atoms at any given time t is:

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

The probability that an atom decayed into Lead 208 is just this number subtracted from one:

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

```
# Constants
NTl = 1000           # Starting Thallium Atoms
NPb = 0              # Starting Lead Atoms
tau = 3.053*60       # Half life in seconds
h = 1.0              # Time step between checking if a decay occurred
p = 1 - 2**(-h/tau) # Probability the decay occurs in that time elapsed
tmax = 1000          # Total time in seconds

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

# Model
for t in tpoints:

    Tlpoints.append(NTl)
    Pbpoints.append(NPb)

    ## Calculate the number of atoms that decay
    decay = 0
    for i in range(NTl):         # For each atom in the sample
        if rand.random() < p:    # Check if it decayed in the last dt
            decay += 1
    NTl -= decay
    NPb += decay

# Plot the data
```

```
plt.figure(figsize=(8,6))
plt.plot(tpoints, Tlpoints, label=r'$^{208}Tl$')
plt.plot(tpoints, Pbpoints, label=r'$^{208}Pb$')
plt.xlabel('Time (s)')
plt.ylabel('Number of atoms')
plt.legend()
plt.grid()
```

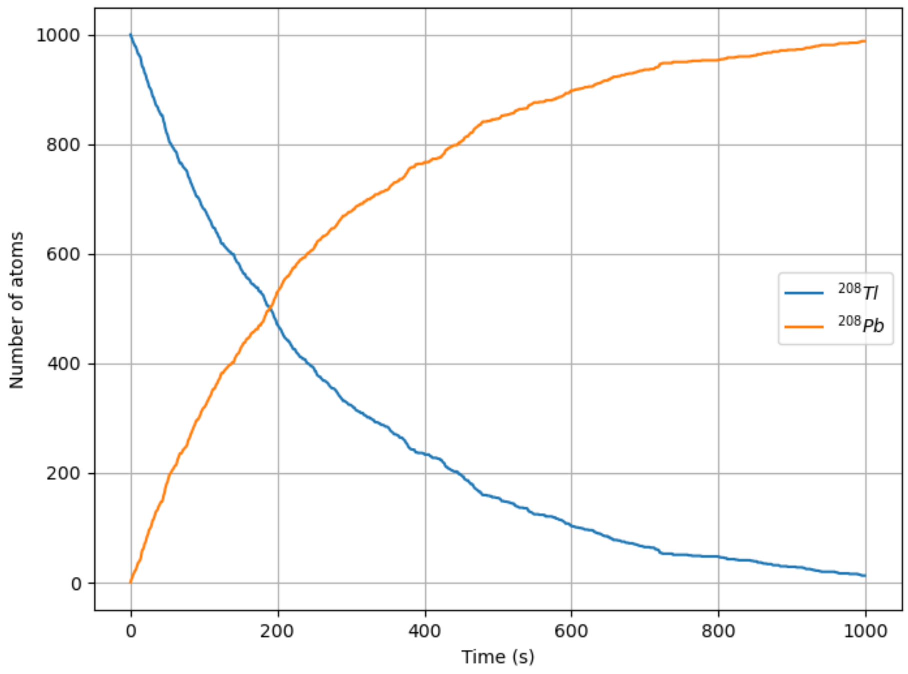

Figure 1: png

Decay Chains

Here's the big project for today! It is less often the case that decays happen like above. Many times there are [Decay Channels] where the atoms decay over time into different elements with different probabilities. Consider the Bismuth 213 decay to Bismuth 209. It decays in two channels:

$$^{213}Bi \rightarrow ^{209}Tl \rightarrow ^{209}Pb \rightarrow ^{209}Bi$$
$$^{213}Bi \rightarrow ^{209}Pb \rightarrow ^{209}Bi$$

The first channel happens to about 2.09% of atoms. So we will ignore it for now. Let's focus on the two step decay to Lead.

Rewrite the model above to account for both decays, which have lifetimes of 46 minutes ($^{213}Bi \rightarrow ^{209} Pb$) and 3.3 hours ($^{209}Pb \rightarrow ^{209} Bi$) respectively.

Plot the number of all three atoms as a function of time

```
## Your code here
```

A Challenge

Model the entire decay including the random but clearly biased split between Lead 209 and Thallium 209, which has a halflife of 2.2 minutes.

Note: We are not including the transition between Bismuth 213 and Polonium 209, which is a very short lived isotope.

Plot the number of all four atoms as a function of time

```
## Your code here
```