Section7

Elliott Ashby

April 10, 2023

1 q1

```
from numpy import random, sqrt
from random import getrandbits


def neutron(L):
    loc = L * random.random()
    num = 2
    return (loc, num)


def chain(loc, num, L):
    R = sqrt(2 * 0.017 * 0.21)
    newloc = []
    temp = []
    for i in range(num):
        direc = bool(getrandbits(1))
        if direc:
            newloc.append(loc + R)
        else:
            newloc.append(loc - R)
    for i in range(len(newloc)):
        if newloc[i] < L and newloc[i] > 0:
            pass
        else:
            temp.append(newloc[i])
    for i in temp:
        del newloc[newloc.index(i)]
    return len(newloc)


if __name__ == '__main__':
    L = 1
    sims = 100
    count = 0
    for i in range(sims):
        count += chain(*neutron(L), L=L)
```

```
    print(count)
```

This code has 2 functions, neutron which returns a location between 0 and L and the number of nuetrons from the fission (in this case 2). And the other function is chain, which takes a location, number of neutrons and L. It then randomly determines which direction the neutrons travel and excludes them if they are outside the range of 0 to L. It then returns the number of total nuetrons that stay within the range.

2 q2

```
def neutron(L):
    loc = L * random.random()
    num = neutrons.neutrons()
    return (loc, num)

if __name__ == '__main__':
    L = 0.1
    initfissions = 100
    avcount = 0
    while avcount < 100:
        temp = 0
        for i in range(1000):
            temp1 = 0
            for j in range(initfissions):
                temp1 += chain(*neutron(L), L=L)
            temp += temp1
        avcount = temp / 1000
        L += 0.001
    print(L)
```

Here we update neutron to include the function neutrons which returns a random number with average of 2.5 and make a new while loop in order to determine the critical value. It does this by taking an average count of secondary fissions of 1000 fissions each with 100 initial fissions. Once the average is larger than the inital fissions, it means more secondary fissions occur than initial fissions. Using this we get a critical value:

$$L_{criticalvalue} \approx 0.412 \pm 0.0005$$

Using more than 100 inital neutrons simply gets a more precision average allowing for a more more precise answer... if the small increase in L allows. In our case increasing the initial fissions doesnt increase the precision of $L_{criticalvalue}$.

3 q3

In order to implement 3 dimensions we need to modify both our functions but not our main.

```
def neutron(L):
    # array of random coords from 0 to L
```

```
    loc = [L * random.random(), L * random.random(), L *
        ↪ random.random()]
    # random number with average of 2
    num = neutrons.neutrons()
    return (loc, num)


def chain(loc, num, L):
    R = sqrt(2 * 0.017 * 0.21)
    newloc = []
    temp = []
    for i in range(num):
        # generate random direction vector
        direc = [R * neutrons.diffusion() * sin(acos(2.0
            ↪ * random.random() - 1.0)) * cos(2.0 * pi *
            ↪ random.random()),
                 R * neutrons.diffusion() * sin(acos(2.0
                     ↪ * random.random() - 1.0)) * sin
                     ↪ (2.0 * pi * random.random()),
                 R * neutrons.diffusion() * cos(acos(2.0
                     ↪ * random.random() - 1.0))]
        # add random vector to the original location
        newloc.append([direc[i] + loc[i] for i in range(
            ↪ len(loc))])
    # if any of the locations of neutrons are outside of
        ↪ the cube, delete them from the array
    for i in range(len(newloc)):
        if newloc[i][0] < L and newloc[i][0] > 0 and
            ↪ newloc[i][1] < L and newloc[i][1] > 0 \
                and newloc[i][2] < L and newloc[i][2] >
                    ↪ 0:
            pass
        else:
            temp.append(newloc[i])
    for i in temp:
        del newloc[newloc.index(i)]
    # return a count of how many neutrons are still in
        ↪ the cube (hence generate new fission reactions)
    return len(newloc)
```

Since python is dynamically typed we can simply change out loc to a list instead of a float. And in chain we can randomly determine a vector as a list and add it to the the location passed into the function.
The final change is to the check whether the new value is in range. We simply expand this to check all dimensions are within 0 to L.

4 q4

Using the above code we can determine a critical value and hence the critical mass, using an increment of 0.001 and initial fissions of 100:

$$L_{criticalvalue} \approx 0.1478 \pm 0.00005$$

$$m_{critical} = L^3_{criticalvalue} \times \rho_{Uranium}$$

Using $\rho_{Uranium} = 18.7\text{Mgm}^{-3}$:
$m_{critical} \approx 60.38\text{kg}$