3rd Year Computational Physics Assignment 4

Dara Corr 18483836

In this assignment we are tasked with modelling the decay of radioactive atoms and the modelling of neutrons trying to pass through a nuclear reactor shield of a certain thickness.

In task 1 we must model the decay of an atom that results in a daughter atom. The parent atom has a probability p of decaying in each second. To model this decay I used a Monte Carlo method which generates a random number between 0 and 1 in each time step (1 second) and checks whether it is within the 0 and p ($p = 0.05$ here) or not. Doing this process for a large number of atoms over a period of time gives a plot which resembles the exponential decay curve given by the following formulae:

$$N(t) = N_0 exp(-\lambda t)$$
$$D(t) = N_0 - N(t)$$

Where N(t) is the number of parent atoms at a given time, λ is the decay constant, t is time and D(t) is the number of daughter atoms at a given time. The plot can be used to estimate the half life of the parent atom. This can be compared with the theoretical value of the half life which is given using these formulae:

$$\lambda = ln\left(\frac{1}{1-p}\right)$$
$$t_{\frac{1}{2}} = \frac{ln(2)}{\lambda}$$

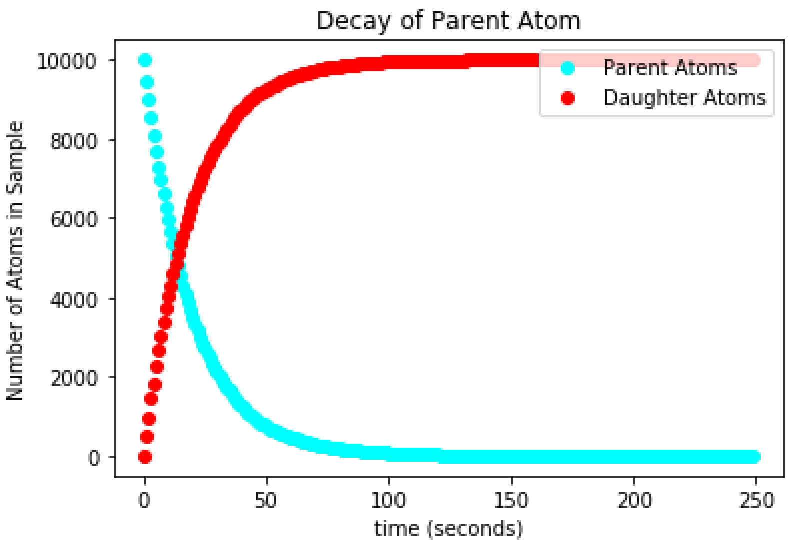
Task 2 uses the same method as task 1 but this time the daughter atom is unstable and decays with a probabilty of $p = 0.02$. This is modelled using the Monte Carlo method as before and graphed. It is observed that the Daughter atoms peak at a certain point and the time this happens can be found using the plot generated in this part of the exercise.

In Task 3, the Monte Carlo Method is used to model a nuclear reactor shield of B atoms thickness. There is a probability of p = 0.03 that each atom of the shield will absorb an incoming neutron. This part is modelled similarly to part 1, but with some changes for the Nuclear reactor shielding. The value of B where 50% of the incident neutrons are absorbed can be estimated in this part.

Task 4 develops on what was used in task 3 and uses it to plot the fraction of transmitted neutrons against different thicknesses B. Finally the thickness B required to reduce the exit flux of neutrons to 1% or less is computed.

In [108]:
```
#task 1

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

p1 = 0.05 #define probability of daughter atom decaying
N_0 = 10000 #define initial number of atoms

def monte_carlo_sim1(N0, t, n_timepoints, p1): #parameters are no. of atoms, t
otal time to simulate decay,the number of time points and probability of an at
om decaying in the time interval for parent atom
    count_parent = np.zeros((n_timepoints)) #empty arrays to hold number of co
unts
    count_daughter = np.zeros((n_timepoints))
    atoms = np.ones((N0)) #Creating an array of numbers to represent the atoms
in the simulation

    for time in range(n_timepoints):
        count_parent[time]   = (atoms == 1).sum() #Counting how many parent an
d daughter atoms remain in the interval
        count_daughter[time] = (atoms == 2).sum()

        for n in range(N0):
            if atoms[n] == 1: #Deciding whether the given atom should decay
                if random.random() <= p1: # if p is less than or equal the p v
alue given then decay takes place
                    atoms[n] = 2 #decay occurs in this time step
                else:
                    atoms[n] = 1 #decay does not occur in this time step
            elif atoms[n] == 2:
                atoms[n] = 2

    return count_parent, count_daughter

t = 250 #time simulation is run for
n_timepoints = 250 #number of time points in simulation
time = np.arange(0,t,t/n_timepoints) #create array of times for simulation
n_parent = monte_carlo_sim1(N_0, t, n_timepoints, p1)[0] #call monte carlo sim
ulation for parent atom decay
n_daughter = monte_carlo_sim1(N_0, t, n_timepoints, p1)[1] #formula to find nu
mbers of daughter atoms during the simulation

plt.figure() #plotting graph
plt.plot(time, n_parent, "o", label = "Parent Atoms", color = "cyan",)
plt.plot(time, n_daughter, "o", label = "Daughter Atoms" , color = "red",)
plt.title("Decay of Parent Atom")
plt.xlabel("time (seconds)")
plt.ylabel("Number of Atoms in Sample")
plt.legend(loc = 'upper right')
plt.show()

for i in range(n_timepoints):
    if n_parent[i] <= n_daughter[i]: #when the parent atoms roughly equal the
 daughter atoms, half the parent atoms have then decayed at this time
        print("Estimate of the half life of the Daughter atom from the numeric
```

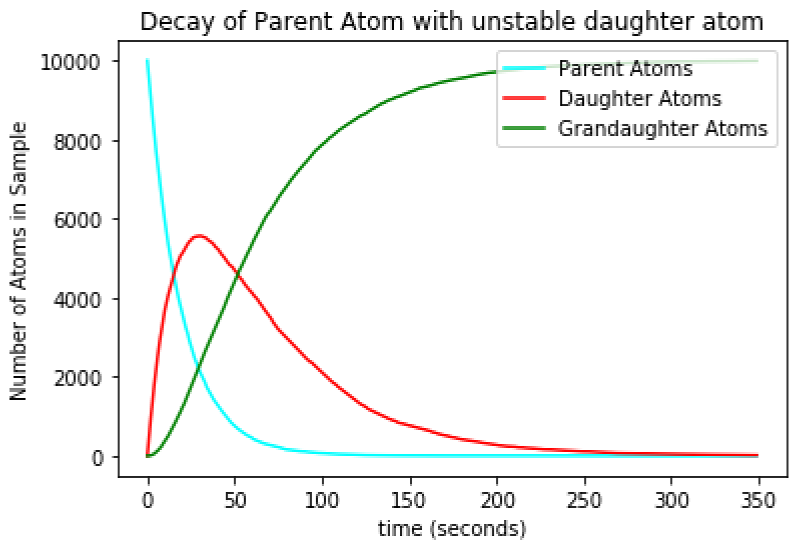
```
al simulation is {0:0.2f} s".format(time[i]))
            break
        else:
            pass


Lambda = np.log(1/(1-p1))
theoretical_halflife = np.log(2)/Lambda #theoretical way to find half-life val
ue
print("The theoretical value of half-life for the parent atom is: {0:0.2f} s".
format(theoretical_halflife))
```




```
Estimate of the half life of the Daughter atom from the numerical simulation
is 14.00 s
The theoretical value of half-life for the parent atom is: 13.51 s
```

In [5]:
```
#task 2

#given new value of p for daughter atom
p2 = 0.02


def monte_carlo_sim2(N0, t, n_timepoints, p1, p2): #parameters are no. of atom
s, total time to simulate decay,the number of time points and probability of a
n atom decaying in the time interval for parent atom

    count_parent = np.zeros((n_timepoints)) #empty arrays to hold number of co
unts
    count_daughter = np.zeros((n_timepoints))
    count_grandaughter = np.zeros((n_timepoints))
    atoms = np.ones((N0)) #Creating an array of numbers to represent the atoms
in the simulation

    for time in range(n_timepoints):
        count_parent[time]   = (atoms == 1).sum() #Counting how many atoms rem
ain in the interval (sums all terms that equal 1)
        count_daughter[time] = (atoms == 2).sum() #Counts how many daughter at
oms remain in the interval (sums all terms that equal 2)
        count_grandaughter[time] = (atoms == 3).sum()

        for n in range(N0):
            if atoms[n] == 1: #Deciding whether the given atom should decay
                if random.random() <= p1: # if p is less than or equal the p v
alue given then decay takes place
                    atoms[n] = 2 #decay occurs in this time step
                else:
                    atoms[n] = 1 #decay does not occur in this time step
            elif atoms[n] == 2:
                    if random.random() <= p2: # if p is less than or equal the
p value given then decay takes place
                        atoms[n] = 3 #decay occurs in this time step
                    else:
                        atoms[n] = 2 #decay does not occur in this time step
            elif atoms[n] == 3:
                atoms[n] = 3


    return count_parent, count_daughter, count_grandaughter


t = 350 #time simulation is run for
n_timepoints = 350 #number of time points in simulation
time = np.arange(0,t,t/n_timepoints) #create array of times for simulation
n_parent = monte_carlo_sim2(N_0, t, n_timepoints, p1, p2)[0] #call monte carlo
simulation
n_daughter = monte_carlo_sim2(N_0, t, n_timepoints, p1, p2)[1]
n_grandaughter = monte_carlo_sim2(N_0, t, n_timepoints, p1, p2)[2]

plt.figure() #plotting graph
plt.plot(time, n_parent, label = "Parent Atoms", color = "cyan")
plt.plot(time, n_daughter, label = "Daughter Atoms" , color = "red")
plt.plot(time, n_grandaughter, label = "Grandaughter Atoms" , color = "green")
```

```
plt.title("Decay of Parent Atom with unstable daughter atom")
plt.xlabel("time (seconds)")
plt.ylabel("Number of Atoms in Sample")
plt.legend(loc = 'upper right')
plt.show()

print("The daughter population reaches a peak at {0:0.2f} s".format(np.argmax(
n_daughter)))
```

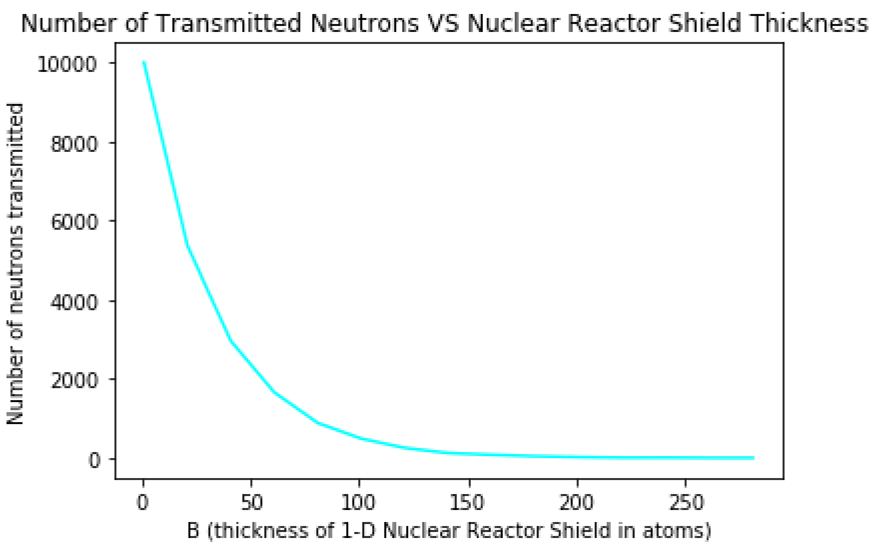



```
The daughter population reaches a peak at 30.00 s
```

In [134]:
```
#task 3
#reactor shield is B atoms thick
#use monte carlo method to measure no. of atoms absorbed

p3 = 0.03 #probability of a neutron being absorbed
flux = 10000 #number of incident neutrons


#this function returns the number of neutrons that pass through the nuclear re
actor shielding
def MC_reactor_shield_sim(flux_0, B, p3): #flux_0 is the incident flux of neut
rons, B is the thickness of the shield in atoms
    count_transmitted = np.zeros((B)) #empty arrays to hold number of absorbed
neutrons
    neutrons = np.ones((flux_0)) #Creating an array of numbers to represent th
e free neutrons in the simulation

    for i in range(B): #loop over total no. of incident neutrons
        count_transmitted[i]   = (neutrons == 1).sum() #sums all the transmitt
ed neutrons for each time step

        for n in range(flux_0): #see if a neutron will get absorbed by one of
 the shield atoms
            if neutrons[n] == 1: #Deciding whether the given atom should be ab
sorbed
                if random.random() > p3: # if p is less than or equal the p va
lue given then it is absorbed
                    neutrons[n] = 1 #absorption occurs in this time step
                    #loop stops when the neutron is absorbed
                else:
                    neutrons[n] = 2 #absorption does not occur in this time st
ep

    return count_transmitted[-1], count_transmitted[-1]/flux_0 #returns the nu
mber of neutrons that have passed through and the fraction that have passed th
rough


neutrons_transmitted = MC_reactor_shield_sim(flux, 23 ,p3)[0]
print(neutrons_transmitted)
print("A value of Thickness B of 23 atoms gives a number of exit flux of neutr
ons approximately equal to 50% of the incident neutron flux")
```

```
5082.0
A value of Thickness B of 23 atoms gives a number of exit flux of neutrons ap
proximately equal to 50% of the incident neutron flux
```

In [127]:
```
#task 4

B_range =  np.arange(1,301,20) #range of thicknesses to plot
Neutron_transmitted_range = np.zeros(len(B_range))
for i in range(len(B_range)):
    Neutron_transmitted_range[i] = (MC_reactor_shield_sim(flux, B_range[i] ,p3
)[0]) # corresponding neutron transmission values to be plotted


plt.figure() #plotting graph
plt.plot(B_range, Neutron_transmitted_range, color = "cyan")
plt.title("Number of Transmitted Neutrons VS Nuclear Reactor Shield Thickness
 ")
plt.xlabel("B (thickness of 1-D Nuclear Reactor Shield in atoms)")
plt.ylabel("Number of neutrons transmitted")
plt.show()
```



In [131]:
```
for i in range(flux):
    if (MC_reactor_shield_sim(flux, i+1 ,p3)[1]) <= 0.01: #finding B required
 to reduce exit flux of atoms to 1% or less
        print("The value of B required to reduce the exit flux of neutrons to
 1% or less is {0:0.1f} atoms".format(i))
        break
    else:
        pass
```

```
The value of B required to reduce the exit flux of neutrons to 1% or less is
150.0 atoms
```