Computational Physics Weekly Assessments

Week 5: Monte Carlo methods

Weekly assessment task & hints

This week you are going to produce a script that uses analytical, ODE and Monte-Carlo approaches to simulate the progression of a collection of atoms through part of their radioactive decay chain. We will look at the decay chain ^{225}Ra > ^{225}Ac > ^{221}Fr.

1. Create a module, named '`cp_5.py`'

- As with previous weeks this will produce and display a single figure when run, illustrating your results.

2. In the global scope of the module create the following variables:

Variable name	*Value*	*Description*
`t_half_rad`	`20.8`	`Half life of` ^{225}`Ra (days)`
`t_half_act`	`10.0`	`Half life of` ^{225}`Ac (days)`
`N0`	`250`	`Initial number of` ^{225}`Ra atoms`
`t1`	`100`	`End time for simulation (days)`
`n_time`	`50`	`Number of timepoints to solve to`

3. Analytically calculate the number of ^{225}Ra atoms at each timepoint and plot this on your figure.

4. Create a function `simulate_monte_carlo(N0, t1, n_time)` that simulates the decay of ^{225}Ra atoms at `n_time` evenly spaced points between time 0 and time t1. The function should return a single 1d array representing the number of ^{255}Ra atoms remaining at each time, `count_rad`.

- Compute a variable, `dt`, which is the duration of a single timestep.
- Compute a variable, `p_decay_rad`, which is the probability that a ^{225}Ra atom will decay within `dt`.
- Initialise a numpy array, `atoms`, to be a 1D array of numbers, one per atom. We will use this array to represent the state of each individual atom in our simulation. Initialise the array to be all ones; a '1' will represent a ^{225}Ra atom, a '2' will represent a ^{225}Ac atom and a '3' will represent a ^{221}Fr atom.
- Use a 'for loop' to go over each timepoint sequentially
  - For each timepoint, use another 'for loop' to examine every element in `atoms`. If an element is equal to '1', it represents is a ^{225}Ra atom. In this case use a suitable random number and the probability of decay within the timestep to decide if the atom decays within the timestep. If it does decay, set the element to now be '2' representing ^{225}Ac, the decay product.
- Once the stochastic decay for the timestep has been applied to all atoms, count the number of ^{225}Ra atoms remaining and store it in the correct timepoint in `count_rad`. The number of ^{225}Ra atoms remaining is the number of elements in atoms with a value equal to '1'
- Add this atom count vs time to your plot

5. Refine your part 4 so that it now returns two arrays, '`return (count_rad, count_act)`' that returns counts for each atom type vs time.

- You will need to expand upon the code to decay atoms of type '1' (Radium) and atoms of type '2' (Actinium) using the appropriate decay probabilities.
- Update your plot from part 4 to show both populations.

6. Implement the differential equations for the decay of both atom types as a single function, f((N_rad, N_act), t) and solve this using scipy.integrate.odeint .

- You should draw on your lecture notes and weekly problems from the last two weeks for this.
- Update your plot from part 5 to show both populations.
- **This part of the problem only requires about 6 additional lines of code.**

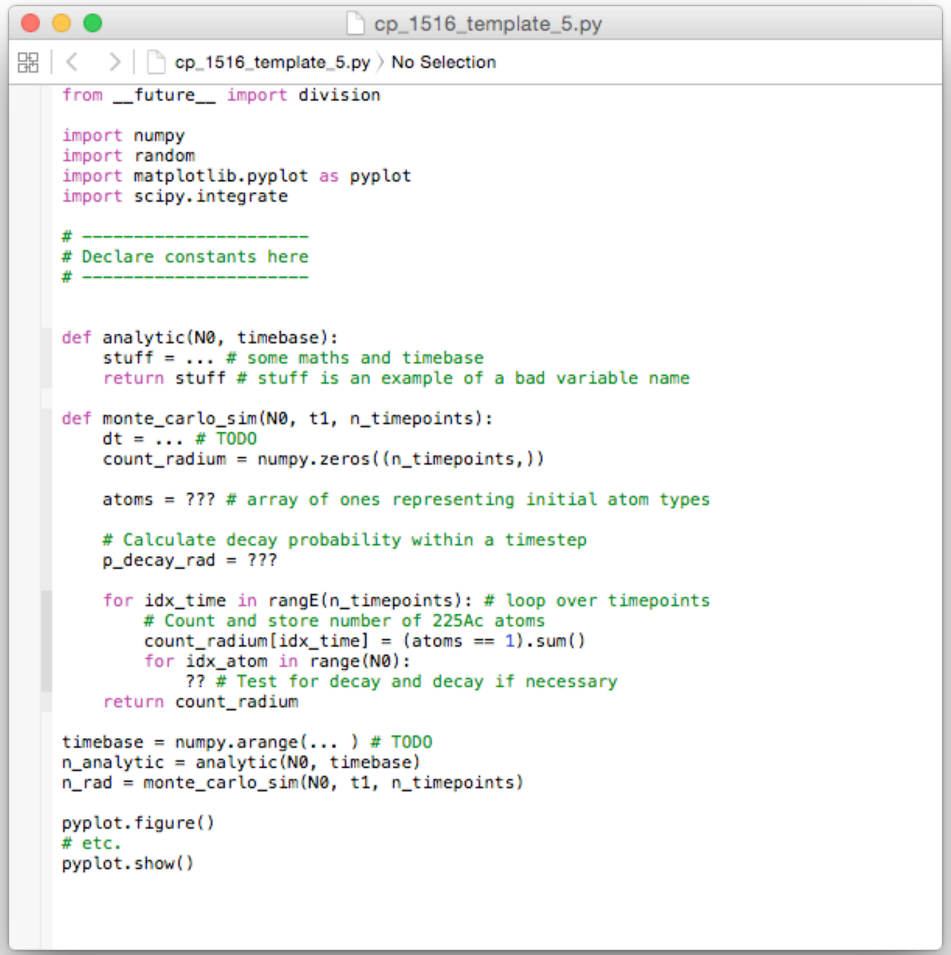
A comment on your figure:

- This week you will be plotting many curves. Ensure that you give sufficient thought to creating a plot that is clear, concise and informative.

There is no question this week.

General comments:

- Place two variables at the start of the code,
  - USER=”your name”
  - USER_ID = “your CIS login”
- Your module must run in order to be awarded any marks
- Your solution should contain no more than about 60 lines of code and perhaps 10 lines of comments. Excessively long submissions may be penalised.
- Sample code is on the next page



```python
from __future__ import division

import numpy
import random
import matplotlib.pyplot as pyplot
import scipy.integrate

# ----------------------
# Declare constants here
# ----------------------


def analytic(N0, timebase):
    stuff = ... # some maths and timebase
    return stuff # stuff is an example of a bad variable name

def monte_carlo_sim(N0, t1, n_timepoints):
    dt = ... # TODO
    count_radium = numpy.zeros((n_timepoints,))

    atoms = ??? # array of ones representing initial atom types

    # Calculate decay probability within a timestep
    p_decay_rad = ???

    for idx_time in rangE(n_timepoints): # loop over timepoints
        # Count and store number of 225Ac atoms
        count_radium[idx_time] = (atoms == 1).sum()
        for idx_atom in range(N0):
            ?? # Test for decay and decay if necessary
    return count_radium

timebase = numpy.arange(... ) # TODO
n_analytic = analytic(N0, timebase)
n_rad = monte_carlo_sim(N0, t1, n_timepoints)

pyplot.figure()
# etc.
pyplot.show()
```