Simulations

Name: Sankalp Saoji (EE16B063)

Date: 13/03/2018

Abstract

In this week, we tried to simulate a one-dimensional model of a tube-light.

1 INTRODUCTION

This week's Python assignment focusses on simulating the model of a tubelight.

We use a 1-dimensional model of the tubelight.

A uniform electric field is present, that accelerates electrons. Electrons are emitted by the cathode with zero energy and accelerate in this field. When they get beyond a

threshold energy 'E_0' , they can drive atoms to excited states. The relaxation of these atoms results in light emission. In our model, we will assume that the relaxation is

immediate. The electron loses all its energy and the process starts again. Electrons reaching the anode are absorbed and lost. Each "time step", an average of 'M' electrons

are introduced at the cathode. The actual number of electrons is determined by finding the integer part of a random number that is "normally distributed" with standard

deviation of 'σ ' and mean 'μ '.

2 IMPLEMENTATION

We create a simulation universe. The tube is divided into 'n' sections. In each time instant, 'M' electrons are injected. We run the simulation for nk turns. The electrons are unable to excite the atoms till they have a velocity of 'u_0'. Beyond this velocity, there is a probability 'p' in each turn that a collision will occur and an atom excited. The electron's velocity reduces to zero if it collides.

2.1 Importing Libraries and Modules :

```
# Importing libraries and modules

import numpy
import math
import matplotlib.pyplot as plt
from pylab import *
import sys
from tabulate import tabulate
```

2.2 Declaring parameters :

```
#Declaring parameters

#n = 100
#grid size, i.e partitions of the tubelight
n = int(sys.argv[1])
#M = 5
#number of electrons emitted in each turn
M = int(sys.argv[2])
#nk = 500
#number of times simulation is done
nk = int(sys.argv[3])
#u0 = 5
#threshold velocity
u0 = float(sys.argv[4])
#p = 0.25
#probability of ionization
p = float(sys.argv[5])
#sigma = 2
#sigma of the probability distribution function
sigma = float(sys.argv[6])
#m = 5
#mean of the probability distribution function
m = float(sys.argv[7])
```

In the above declaration, I have provided the user the freedom to put his values with the help of sys.argv.

So while running my program, do,

```
python EE16B063_Lab_Assignment_6.py 100 5 500 5 0.25 2 5
(You can change the parameters if wanted).
```

2.3 Declaring matrices to hold electron information while iterating and after iteration :

```
#Declaring matrices to hold electron information while iterating
```

```
xx = numpy.zeros(n*M)
#electron position
u = numpy.zeros(n*M)
#electron velocity
dx = numpy.zeros(n*M)
#displacement current
```

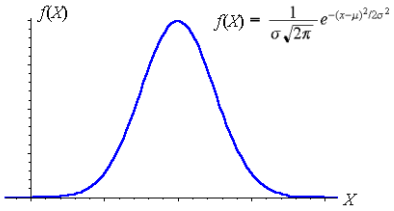
I have initialised the matrices to zeros initially and I have defined them of size n*M.

```
#Declaring vectors to hold electron information after iteration

I = []                                  #intensity of emitted light
X = []                                  #electron position
V = []                                  #electron velocity
```

2.4 Creating the 'for' loop for iteration :

2.4.1 Creating the electrons and setting their positions

First, we need to emit electrins in the chamber. That we will do using the 'numpy.random.rand()' function which is a 'normally distributed random number'. We multiply it by the standard deviation and add the mean value. But we need to find where the injected electrons go in the xx array. We find the unused indices in the electron vector and add the new electrons there. For these electrons, we initialize the position $x_i = 1$. So the tubelight stretches from 1 to n. A position of 0 means an unused slot. We search for that using 'numpy.random.choice()' to find the unused slots. 'numpy.random.randn()' is function of the form,



Here, in my code, we have,

μ = M

σ= sigma

```
m = abs(int(numpy.random.randn()*sigma + M))
#creating electrons
hh = numpy.where(xx == 0)[0]
#finding the location where position of the electron is zero
m_slots = numpy.random.choice(hh,m)
#finding the location of electrons and making its position value 1
```

2.4.2 Finding the electrons in the chamber and updating their positions and velocities

Now, we need to find the electrons present in the chamber. We defined vectors of dimension nM. So not all the entries in the vectors represent electrons. There will be a constraint on the electron present in the chamber. It must have the following properties:

1) If an electron is in the chamber, its position must satisfy $0 < x < L$, where $L = n$ for this simulation.

2) Anytime the electron reaches $x = L$, it is reset to $x = 0$. So, if an entry has zero x position, that electron has not yet been injected. Only $x > 0$ entries correspond to

electrons within the chamber. We do this by finding all those electrons whose position is greater than zero. For this, we use the 'where' command.

```
ii = where(xx>0)
```

ii is a vector containing the indices of vector xx that have positive entries. Now, we calculate dispacement and velocity during this turn, assuming that acceleration of 1. Hence, the displacement of the i^{th} electron is given by,

$dx_i = u_i \Delta t + \frac{a(\Delta t)^2}{2}$

$u_i = u_0 + a(\Delta t)$

Since, $a = 1$ and we are already in one iteration in an instant of time,

$dx[ii] = u[ii] + 0.5$

$u_i = u_i + 1$

Now, we advance the pararmeters of these electrons,
The equations are,

$x_i = x_i + dx_i$

$u_i = u_i + 1$

```
ii = numpy.where(xx>0)[0]
#finding indices where xx>0
dx[ii] = u[ii] + 0.5
#updating the position and velocity of the electron
xx = xx + dx
u[ii] = u[ii] + 1
```

2.4.3 Determining the particles which have hit the anode

Now, we determine the particles which have hit the anode (their positions would be beyond n). So, we set the positions, displacements and velocities of these particles to zero.

```
jj = numpy.where(xx>=n)[0]
#putting constraints on the electrons
whose position went beyond n
xx[jj] = 0
dx[jj] = 0
u[jj] = 0
```

2.4.4 Determining the electrons which had collision and setting their positions and velocities

Now, we find those electrons whose velocity is greater than or equal to the threshold. And then, we want the ionised electrons.Assuming that 'kk' is the vector of indices corresponding to energetic electrons. We create a uniformly distributed random vector of length 'len(kk)' and find those indices that are less than 'p', the probability of a collision. So, we do this like,

```
ll = where(rand(len(kk[0]))<=p)
                                                    kl=kk[ll]
```

The first line creates the random vector and uses 'where' to locate those entries that are less than or equal to p. I use len(kk[0]) since kk is a list and I want the first array in the list. Now 'll' is a vector of indices that tells us the indices in vector kk. We want the electron indices. So, kl now contains the indices of those energetic electrons that will suffer a collision. We reset the velocities of these electrons to zero (they suffered an inelastic collision). The collision could have occurred at any point between the previous x_i and the current x_i. So we determine the actual point of collision and update the xx array suitably as,

$x_i \leftarrow x_i - dx_i\rho$
Here ρ is a random number between 0 and 1.

```
kk = numpy.where(u>=u0)[0]
#indices where velocity of the electron is greater than the threshold
ll = numpy.where(rand(len(kk))<=p)[0]
#indices of the electrons which have the probability of collision
kl = kk[ll]
#indices of the electrons which are colliding
u[kl] = 0
#setting the velocity of electron as zero after the inelastic collision
xx[kl] = xx[kl] - dx[kl]*(rand(len(xx[kl])))
#finding the location of collision
```

2.4.5 Updating the intensity of light, positions and velocties of electrons

The excited atoms result in emission of light from the point. So we have to add a photon at that point.

```
I.extend(xx[kl].tolist())
#appending to the lists
X.extend(xx[ii].tolist())
V.extend(u[ii].tolist())
```

In above codes, we are extending the list after converting the array xx[kl] to a list. This is a slow process, since Python may need to allocate a new, larger vector and copy over the old one. So it should be done only when we cannot know the size of the vector ahead of time.

Whole 'for' loop looks like below,

```
for k in range(1,nk):
                m = abs(int(numpy.random.randn()*sigma + m))
#creating electrons
                hh = numpy.where(xx == 0)[0]
#finding the location where position of
the electron is zero
                m_slots = numpy.random.choice(hh,m)
#finding the location of electrons and
making its position value 1
                xx[m_slots] = 1

                ii = numpy.where(xx>0)[0]
#finding indices where xx>0
                dx[ii] = u[ii] + 0.5
#updating the position and velocity of
the electron
                xx = xx + dx
                u[ii] = u[ii] + 1

                jj = numpy.where(xx>=n)[0]
#putting constraints on the electrons
whose position went beyond n
                xx[jj] = 0
                dx[jj] = 0
                u[jj] = 0

                kk = numpy.where(u>=u0)[0]
#indices where velocity of the electron
is greater than the threshold
                ll = numpy.where(rand(len(kk))<=p)[0]
#indices of the electrons which have
```

```
the probability of collision
                kl = kk[ll]
#indices of the electrons which
are colliding
                u[kl] = 0
#setting the velocity of electron
as zero after the inelastic collision
                xx[kl] = xx[kl] - dx[kl]*(rand(len(xx[kl])))
#finding the location of collision

                I.extend(xx[kl].tolist())
#appending to the lists
                X.extend(xx[ii].tolist())
                V.extend(u[ii].tolist())
```

2.5 Obtaining the population plost and the phase-space plot :

At the end of the run, we will have I, the intensity vector, X the position vector and V the velocity vector.

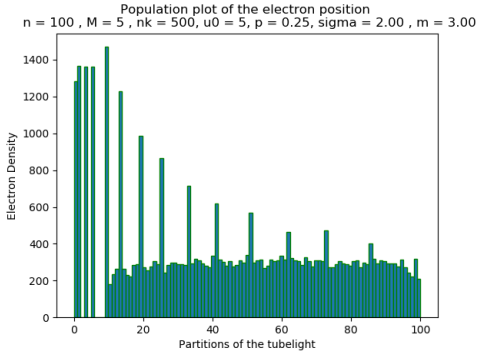
We first plot the "electron density", i.e., the number of electrons between i and i + 1. We can do this by generating a population plot of X. N

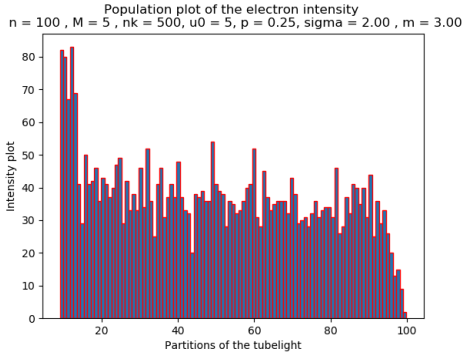
Now, we plot a population plot of the light intensity.

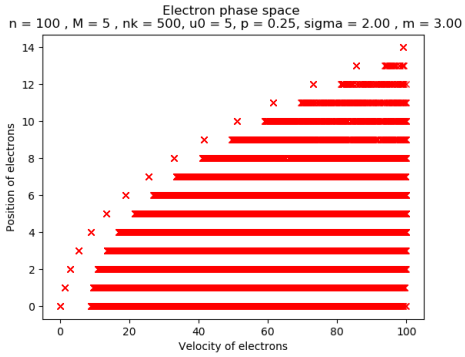
Lastly, we have the plot of "electron phase space" for each electron corresponding to x = x_i and y = v_i .

```
# Creating population and phase-space plot

figure(0)
plt.hist(X,101,edgecolor='green')
plt.xlabel('Partitions of the tubelight')
plt.ylabel('Electron Density')
plt.title('Population plot of the electron position \n n = %d ,
M = %d , nk = %d, u0 = %d, p = %0.02f, sigma = %0.02f ,
m = %0.02f' %(n, M, nk, u0, p, sigma, m))
plt.show()

figure(1)
#hist = (population, bins, data)
pop_count, bins, rect = plt.hist(I, 101, edgecolor = 'red')
plt.xlabel('Partitions of the tubelight')
plt.ylabel('Intensity plot')
plt.title('Population plot of the electron intensity \n n = %d ,
M = %d , nk = %d, u0 = %d, p = %0.02f, sigma = %0.02f ,
m = %0.02f' %(n, M, nk, u0, p, sigma, m))
plt.show()
```

```
figure(2)
plt.plot(X, V, 'rx')
plt.xlabel('Velocity of electrons')
plt.ylabel('Position of electrons')
plt.title('Electron phase space \n n = %d , M = %d , nk = %d,
u0 = %d, p = %0.02f, sigma = %0.02f , m = %0.02f'
%(n, M, nk, u0, p, sigma, m))
plt.show()
```







2.6 Obtaining the table of intensity values :

We also want to print out the intensity as a table.

This data is returned by the hist function when it plots the histogram. What is returned is 'tuple' with three elements,

1) The first is an array of population counts

2) The second is the bin position

3) The third (which we do not use) is a list of the rectangles that are used to build up the histogram.

We convert to mid point values by,

```
xpos = 0.5(bins[0:-1] + bins[1:])
```

This averages the vector containing left positions of all the bins and the vector containing the right positions of all the bins. We only need to print out the two arrays in the following format:

Intensity Data xpos count

binval1 population1

binval2 population2

... ...

binvalN populationN

```
#Creating the table

x_pos = 0.5*(bins[0:-1] + bins[1:])
#converting to mid-point values

li = list()
print ("Intensity Data")
first_row = ["xpos", "Count"]
for pos, co in zip(x_pos,pop_count):
        l_temp = [pos,co]
        li.append(l_temp)
li.insert(0, first_row)
print(tabulate(li,tablefmt = 'psql', headers = "firstrow"))
```

Below is the intensity data.

```
+---------+---------+
|  xpos   |  Count  |
|---------+---------|
| 9.44935 |      71 |
```

54	10.3447
76	11.24
81	12.1353
74	13.0306
53	13.926
45	14.8213
41	15.7166
48	16.6119
54	17.5073
31	18.4026
45	19.2979
46	20.1932
43	21.0886
45	21.9839
58	22.8792
51	23.7745
49	24.6699
40	25.5652
43	26.4605
43	27.3558
35	28.2512
35	29.1465
43	30.0418

55	30.9371
46	31.8325
47	32.7278
42	33.6231
38	34.5184
48	35.4138
43	36.3091
49	37.2044
40	38.0997
44	38.9951
33	39.8904
48	40.7857
33	41.681
48	42.5764
40	43.4717
52	44.367
43	45.2623
36	46.1577
54	47.053
46	47.9483
41	48.8436
40	49.739
29	50.6343

36	51.5296
44	52.4249
43	53.3203
41	54.2156
34	55.1109
41	56.0062
48	56.9016
37	57.7969
50	58.6922
36	59.5875
48	60.4829
43	61.3782
32	62.2735
26	63.1688
37	64.0642
38	64.9595
31	65.8548
44	66.7501
35	67.6454
52	68.5408
40	69.4361
43	70.3314
38	71.2267

30	72.1221
31	73.0174
46	73.9127
27	74.808
37	75.7034
37	76.5987
41	77.494
47	78.3893
46	79.2847
35	80.18
31	81.0753
42	81.9706
50	82.866
42	83.7613
38	84.6566
39	85.5519
39	86.4473
43	87.3426
29	88.2379
31	89.1332
36	90.0286
35	90.9239
26	91.8192

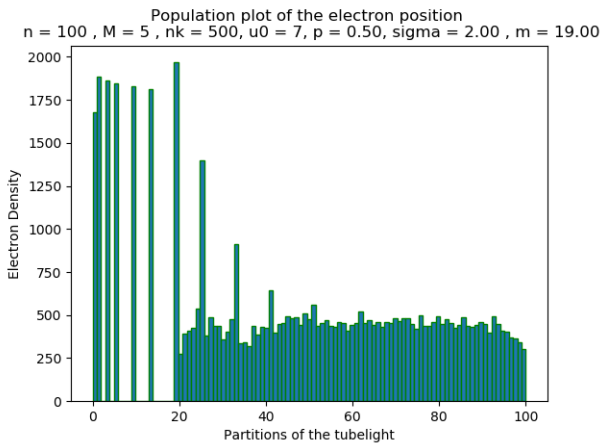
39	92.7145
45	93.6099
29	94.5052
41	95.4005
23	96.2958
24	97.1912
10	98.0865
6	98.9818

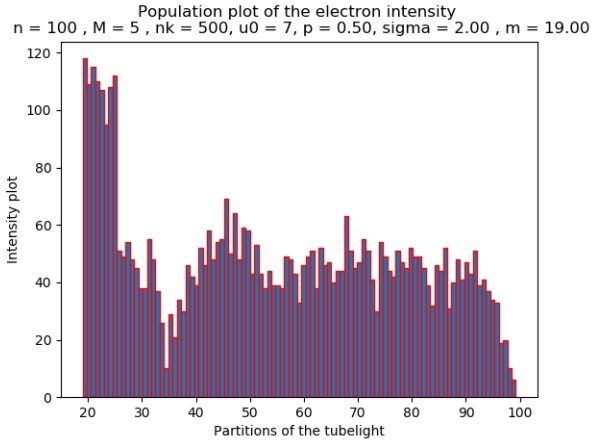
2.7 Obtaining the previous plots for changed values of threshold velocity and p :

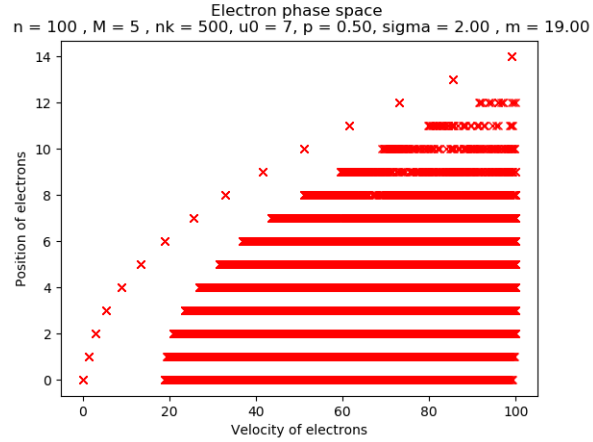
Now, threshold velocity,

$u_0 = 7$

$p = 0.5$







The region upto 20 is where electrons are building up their energy. Beyond that is a region where the emission decays, representing the fewer energetic electrons that reached there before colliding. At 40 is the next peak. But this is a diffuse peak since the zero energy location of different electrons is different.