EE2703 : Applied Programming Lab
Assignment 6

P N Neelesh Sumedh
ee19b047

April 13, 2021

1 Introduction

In this assignment we will simulate a tubelight using python. 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_o, 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.

2 Assignment

2.1 Defining the Parameters

The parameters which control the run are as follows. These parameters are given by the user in the commandline else the program runs on the default values.

```
1) n : Spatial Grid Size
    input in commandline as " --n n_value ". Default is 100.
2) M : Number of electrons injected per turn
    input in commandline as "--M M_value". Default is 5.
3) nk : Number of turns to simulate
    input in commandline as "--nk nk_value". Default is 500.
4) u0 : Threshold Velocity
    input in commandline as "--u0 u0_value". Default is 5.0
5) p : Probability that ioniztion will occur
    input in commandline as "--p p_value". Default is 0.25.
6) Msig : Standard Deviation of number of electrons
```

```
input in commandline as "--Msig Msig_value". Default is 2.
```

The parameters are taken from the commandline given by the user. The code to update the parameter values is as follows.

```
parser = argparse.ArgumentParser()
parser.add_argument('--n', help='The spatial size of grid', type = int, default = 100)
parser.add_argument('--M', help = 'Number of electrons injected in one turn', type = int, default=5)
parser.add_argument('--nk', help = 'Number of turns to simulate', type = int, default = 500)
parser.add_argument('--u0', help = 'Threshold Velocity', type = float, default = 5.)
parser.add_argument('--p', help = 'Probability that ionization will occur', type = float, default = 0.5)
parser.add_argument('--Msig', help = 'Standard Diveation', type = float, default = 2)

args = parser.parse_args()
n = args.n
M = args.M
nk = args.nk
u0 = args.u0
p = args.p
Msig = args.Msig
```

Code lines which check for the correct range of values are also written. The Probability value should lie in the range [0,1]. The n, M, nk values should be positive integers.

2.2 Creating vectors

Vectors are created to store electron information. The dimension of the vectors is n*M. The vectors are used to store,

```
1) Electron Position stored as xx
2) Electron Velocity stored as u
3) Displacement in current turn stored as dx
```

We also require vectors to store the Intensity of light, position of electrons and their velocity after each iteration. To make updating of these simpler, we use list instead of numpy array.

```
I = []  # Intensity of emitted light
X = []  # Position of Electron
V = []  # Velocity of Electron
```

2.3 Performing the Iteration

The iteration is performed to simulate TubeLight. For Loop is written to perform the iteration. The forloop is as follows:

```
for k=1:nk:
   Electrons in Chamber
   Exiting Electrons
   Excited Electrons
   Intensity Update
   Entering Electrons
   Updating X and V
end
```

Each block in the for loop is explained in the following sections.

2.4 Electrons in Chamber

We find the positions where electrons are present in the present iteration. We take the length of the tubelight to be L. The position of the electron will be inbetween $0 < x < L$. We take L = n for this simulation. We assume that, when ever electron reaches $x = L$, we reset it to $x = 0$. When $x = 0$, the electron has not entered the chamber, when $0 < x < L$ the electron is in the chamber and when $x \geq L$ the electron has exited the chamber. After finding the positions of existing electrons, the position is updated using the equation

$$dx_i = u_i \Delta t + \frac{1}{2} a (\Delta t)^2 = u_i + 0.5$$

The velocity of those electrons is also increased. The equation used is

$$v_i = u_i + a\Delta t = u_i + a$$

It is increased by a amount. The acceleration for simplicity is taken to be 1 and Δt is also taken to be 1. This part is impletemented in the following code.

```
1   ii = np.where(xx>0)
2   dx[ii] = u[ii]+0.5
3   xx[ii] = xx[ii]+dx[ii]
4   u[ii] = u[ii]+1
```

2.5 Exiting Electrons

The electrons as they move, they reach the anode. The anode is at $x = L$. This implies that the electron is exiting. So the electrons which crossed anode are found out and the corresponding

index positions in the position, velocity vector are reset to zero. The displacement is also set to zero. This part is implemented as

```
    xx[np.where(xx>=n)] = 0
    u[np.where(xx>=n)] = 0
    dx[np.where(xx>=n)] = 0
```

2.6 Excited Electrons

The indices of electrons which are excited is found out and is stored in vector *kk*. Now, we find the indices among these which collide to give light. This is stored in another vector *kl*. The vector *ll* is an intermediate vector, not of much use except to find the electrons in *kk* which collide. The velocity of these electrons is now 0. Therefore, this has to be updated in the vector *u*. The collision could have occured at any time between $(k-1)\Delta t < t < k\Delta t$. To accomodate this, the position is also updated using

$$x_i \longleftarrow x_i - dx_i\rho$$

Here ρ is a random number between 0 and 1. The code for this block:

```
    kk = np.where(u>=u0)[0]
    ll = np.where(np.random.rand(len(kk))<=p)
    kl = kk[ll]
    u[kl] = 0
    xx[kl] = xx[kl] - dx[kl]*np.random.rand(len(kl))
```

2.7 Intensity Update

Now that we got the position of collisions with sufficient energy, we store the position of electron in the Intensity list to indicate that a photon has been emitted from that position.

```
    I.extend(xx[kl].tolist())
```

2.8 Entering Electrons

Now, we inject new electrons. The number of electrons that are injected is given by

$$m = randn() * Msig + M$$

These are the number of electrons that entered. Now, the data of these electrons is to be stored in data vectors at the indices where xx = 0. If the number of indices which has xx = 0 is less than m, then only these indices are updated. The code for this part. The position is initialized to 1. The velocity is initialized to 0.

```
    m = int(np.random.randn()*Msig+M)
    emptypos = np.where(xx==0)
    minindx = min(len(emptypos[0]),m)
    xx[emptypos[0][:minindx]] = 1
    u[emptypos[0][:minindx]] = 0
```

2.9 Updating X and V

Now the new position and velocity of electrons in the chamber are to be stored. These are stored in the lists X and V respectively.

```
    X.extend(xx[np.where(xx>0)].tolist())
    V.extend(u[np.where(xx>0)].tolist())
```

3 Electron Density, Light Intensity and Phase Space

3.1 Electron Density

The Electron Density is plot using the X list. To plot, *hist* is used. It counts the number of electrons at a position x and plots it.

```
    plt.figure(0)
    plt.hist(X, n, [0,n] , histtype='bar', edgecolor='black')
    plt.title('Electron Density')
    plt.show()
```

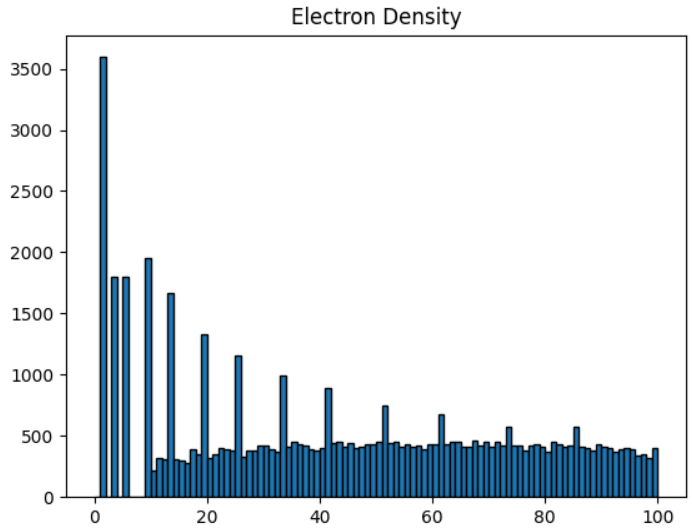

Figure 1: Electron Density

3.2 Light Intensity

The Light Intensity is plot using I list. The plotting is same as the electron density is plotted. Each electron collision is assumed to give only one photon. Therefore, more number of photons from a given position means more is the intensity.

```
    plt.figure(1)
    count, bins, _ = plt.hist(I,n, [0,n], histtype='bar', edgecolor='black')
    plt.title('Light Intensity')
    plt.show()
```

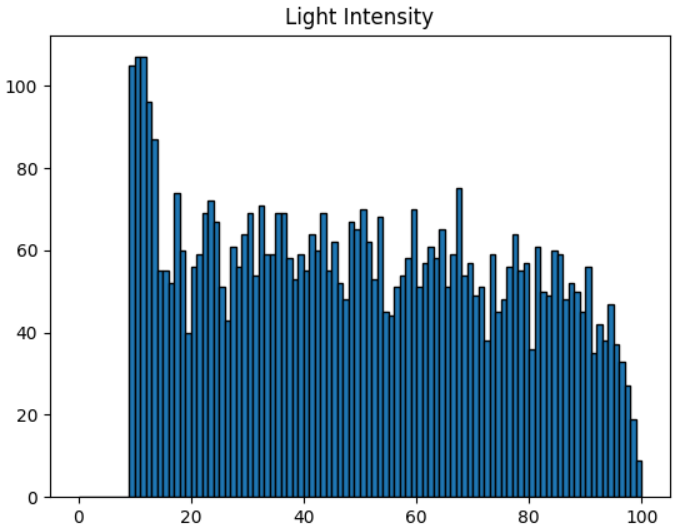

Figure 2: Light Intensity

3.3 Phase Space

Phase Space is the graph between position and velocity. This is plot using X and V lists.

```
    plt.figure(2)
    plt.plot(X,V,'bo',markersize = 1)
    plt.title('Electron Phase Space')
    plt.xlabel('X $\longrightarrow$')
    plt.ylabel('V $\longrightarrow$')
    plt.show()
```

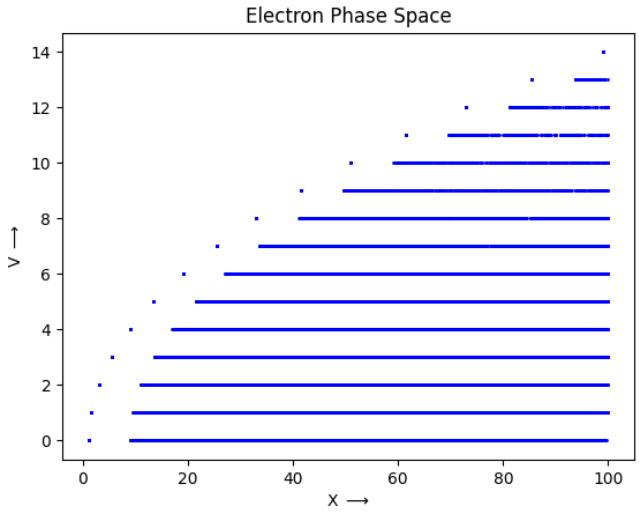

Figure 3: Electron Phase Space

In the phase space graph we can see that as electrons move along the length of the tubelight, the velocity of electron increases. High velocity electrons lie mostly near the anode end.

4 Intensity Table

The code to print the Intensity table is

```
    intensitycount = [[x,y] for x, y in zip(xpos, count)]

    print("Intensity Data : \n")
    print(tabulate((intensitycount), headers=['xpos', 'count'],tablefmt='github')
    )
```

To print the table in the output, we need to import tabulate library.

5 Conclusion

We have implemented a 1-Dimensional Tube Light using python code in this assignment. We observe that the electron density is almost uniformly spread out except at some places where there are spikes in the Electron Density graph. The electrons which are near to anode have high energy where as the electrons which are near the cathode have low energies. In the Figure 3 we can see that the threshold velocity is reached approximately only after x = 10. Therefore, if there is any collision with sufficient energy, then it should happen after x = 10 point. Thus we should have Light Intensity only after x = 10. Exactly this is the case, as we can see in the intensity graph. The intensity is high at x = 10 but decreases as x increases, this is because even though the electrons have sufficient energy to produce emission, as they move with high speeds as x increases, their chances of colliding decreases. Thus the intensity decreases. This is the implementation of light bulb using python.