EE2703 : Applied Programming Lab
Assignment 6
The Tubelight Simulation

Bachotti Sai Krishna Shanmukh
EE19B009

April 11, 2021

Introduction

In this assignment, we simulate a 1D model of Tubelight. The problem is to simulate and understand intensity pattern of emitted light and the parameters involved.

1 Modelling the problem

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 E0, 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 it's energy and the process starts again. At each time step mean number of electrons introduced at the cathode is N and the actual number of electrons in the tube is the integer part of a normally distributed random variable with standard deviation of σ

2 Input Parameters

The input arguments for the simulation are listed below:

- n : Spatial Grid Size (1 Dimensional Grid) Default value=100
- M : Number of electrons injected per turn Default value = 5

- p: Probability of Ionization Default value = 0.25
- nk: Number of turns in simulation Default value = 500
- u0: Threshold Velocity Default value = 5
- Msig: Standard deviation of electron distribution Default value: 2

The following code parses the command line for input arguments

```
def probability_check(x):
    """
    To check for allowed values of probability
    """
    try:
        p = float(x)
    except ValueError:
        raise argparse.ArgumentTypeError("Invalid input: %r is not a floating point value"%x)
    if p<0.0 or p>1.0 :
        raise argparse.ArgumentTypeError("%r is not a valid probability value. Must lie between [0,1]"%x)
    return p

def natural_check(x):
    """
    To check if input values are natural numbers
    """
    try:
        n = int(x)
    except ValueError:
        raise argparse.ArgumentTypeError("Invalid input:%r is not an integer."%x)
    if n<=0:
        raise argparse.ArgumentTypeError("%r is not a positive integer. Please enter a positive integer"%n)
    return n

def positive_check(x):
    """
    To check if input values are positive numbers
    """
    try:
        y = float(x)
    except ValueError:
        raise argparse.ArgumentErrorType("Invalid input:%r is not a floating point value"%x)
    if y<=0:
        raise argparse.ArgumentErrorType("%r is not a positive value. Please enter a positive value"%y)
    return y

```

```
#Collecting input by parsing command line arg
parser = argparse.ArgumentParser()
parser.add_argument("--n", type = natural_check, default = 100, help = 'Spatial Grid size')
parser.add_argument("--M", type = natural_check, default = 5, help = 'No. of electrons injected per turn')
parser.add_argument("--nk", type = natural_check, default = 500, help = 'No. of turns to simulate')
parser.add_argument("--u0", type = positive_check, default = 5, help = 'Threshold velocity')
parser.add_argument("--p", type = probability_check, default = 0.25, help = 'Probability that ionization will occur')
parser.add_argument("--Msig", type= positive_check, default = 2, help = 'Standard Deviation of Electron distribution')
args = vars(parser.parse_args())
n,M,nk,u0,p,Msig=(args['n'],args['M'],args['nk'],args['u0'], args['p'], args['Msig'])
```

3 The Simulation

The following code is used for simulation.
np.where() function is used to find out the indices of elements which match a given criterion.
All updates are done using vectorized operations wherever numpy arrays are used.

```
xx = np.zeros(n*M) # Initialisation of position array
u = np.zeros(n*M)  # velocity array
dx = np.zeros(n*M) # displacement array
I = []
X = []             # Empty lists for collecting Intensity, Position and Velocity data for all runs
V = []

for k in range(nk):      # Loop through each turn of simulation
    N = int(Msig*np.random.randn() + M)  # No. of electrons in the grid
    # Injection of e- at cathode
    start = np.where(xx==0)
    xx[start[0][:N]] = 1
    # Propagation : Update the prev position and velocity and compute
    ii = np.where(xx>0)
    X.extend(xx[ii].tolist())
    V.extend(u[ii].tolist())
    dx[ii] = u[ii] + 0.5
    u[ii] +=1
    xx[ii] += dx[ii]
    # Reset the status of e- which reached end of grid
    end = np.where(xx>=n)
    xx[end] = 0
    u[end] = 0
```

```
    dx[end] = 0
    # Threshold
    kk = np.where(u>= u0)
    ll = np.where(np.random.rand(len(kk[0]))<=p)
    kl = kk[0][ll]
    # position update of ionized e-
    # dt is uniformly sampled between [0,1] and
    # xx is recorrected for ionized e- by using dt
    dt = np.random.rand()
    xx[kl] = (xx[kl] - dx[kl]) + (u[kl]-1)*dt + 0.5*dt**2
    # Velocity update of ionized e-
    u[kl] = 0
    I.extend(xx[kl].tolist())
```

4 Histogram and Phase Space Plots

The lists X,V and I collect all the information about position, velocity and intensity in all iterations respectively.

```
#Plots
#e- density histogram
fig, ax = plt.subplots(num=0,figsize=(7, 5))
plt.hist(X,bins=n,edgecolor='black')
plt.xlabel('Position')
plt.ylabel('No. of electrons')
plt.title(r'Electron Density')
#Intensity histogram
fig, ax = plt.subplots(num=1,figsize=(7, 5))
count, bins, rect =plt.hist(I,bins=n,range=[0,n],edgecolor='black')
plt.xlabel('Position')
plt.ylabel('No. of ionized electrons')
plt.title(r'Intensity of Emitted Light')
#Electron Phase Space
fig, ax = plt.subplots(num=2,figsize=(7,7))
plt.plot(X,V,'rx',markersize =5)
plt.xlabel(r' $Position\longrightarrow$')
plt.ylabel(r' $Velocity\longrightarrow$')
plt.title('Electron Phase Space')
```

The following figures are plotted for default values. We use the plt.hist() function to plot histograms. It can be observed that there is zero intensity until a certain value, indicating a dark space. Even the phase space plot suggests that threshold velocity is not reached until a particular position, indicating no emissions in that region. The intensity of emitted light seems to be decaying as we move towards the anode end. This can be explained by the fact that fewer excited electrons can sustain till the end, hence lower number of emissions towards the anode end. As we move away from the cathode, the peaks in the intensity of the of emissions become more diffuse as zero energy location of different electrons is different.

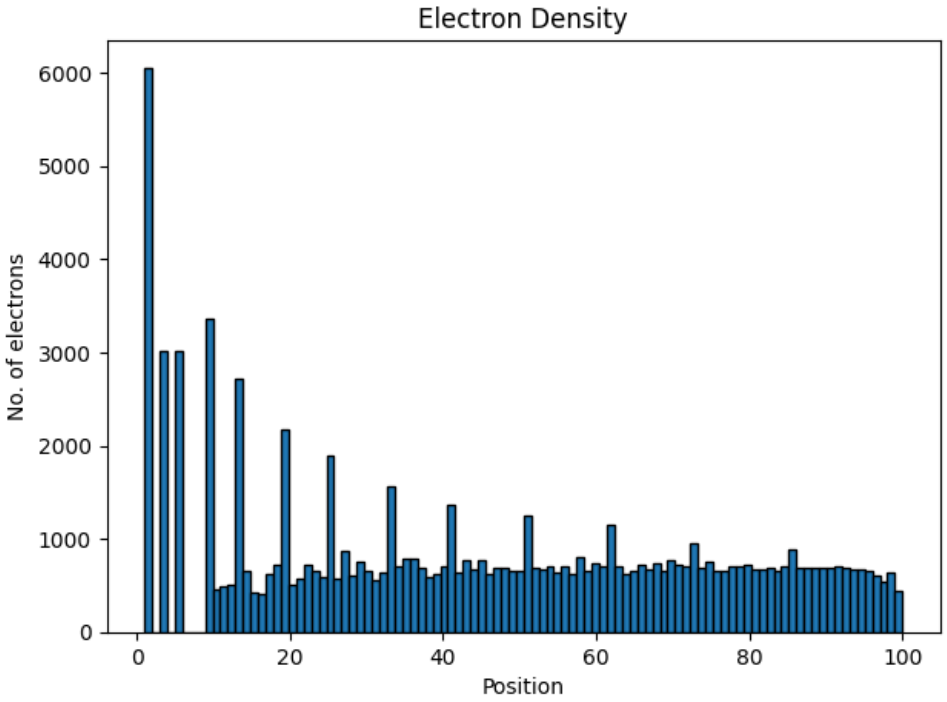

Figure 1: Histogram plot of Electron density

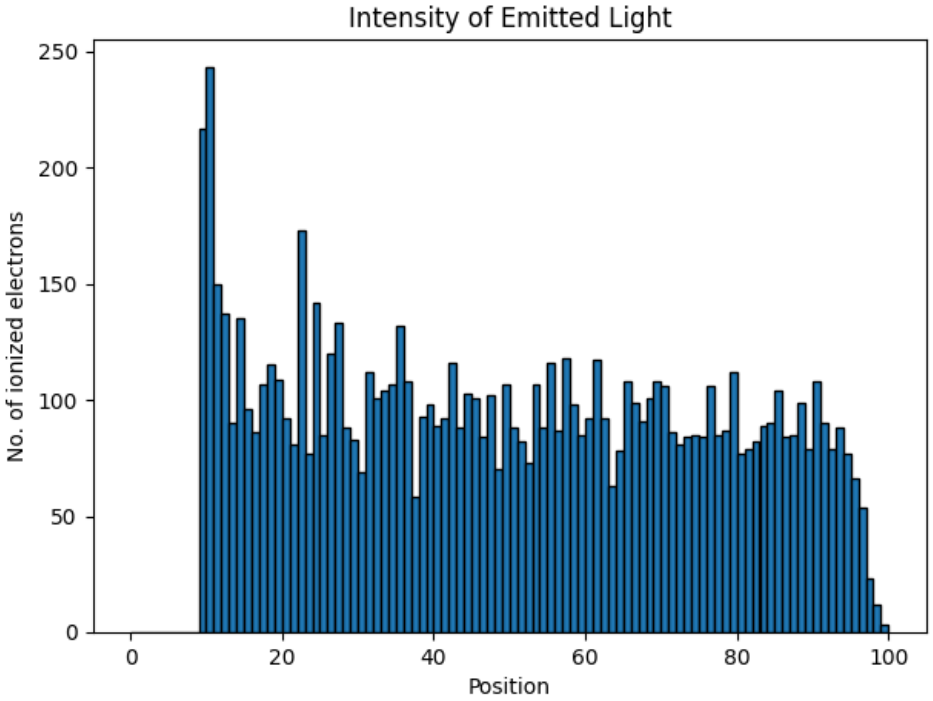

Figure 2: Histogram plot of Intensity

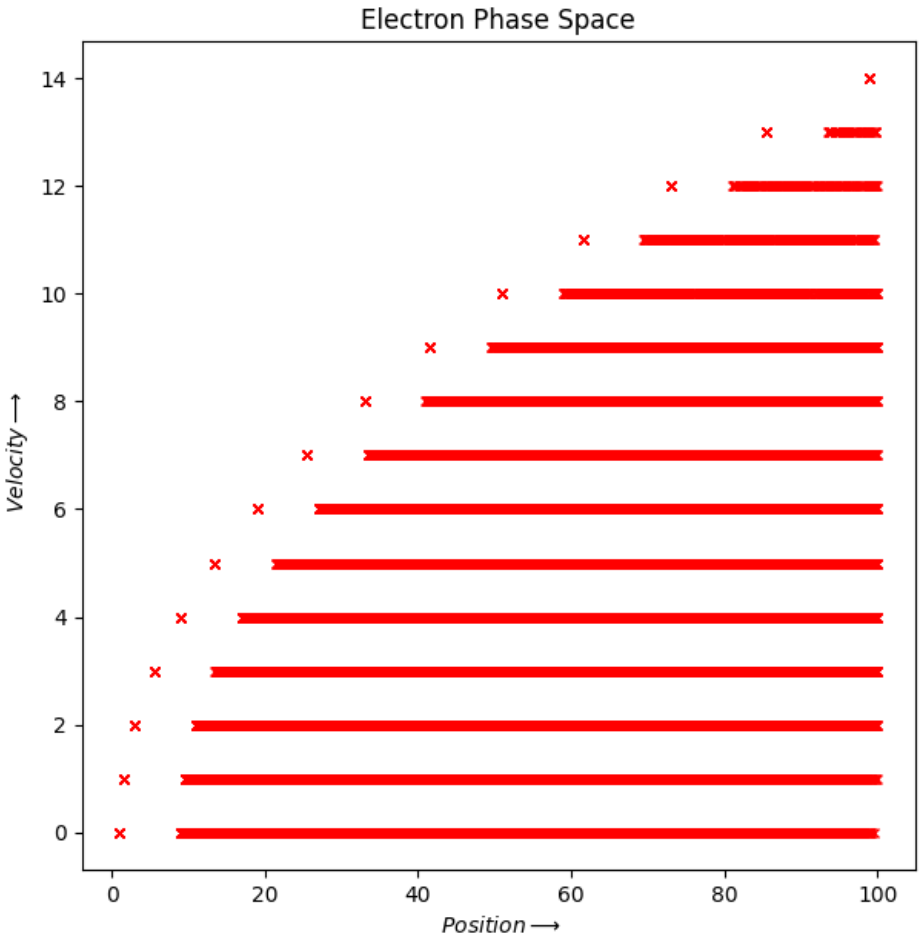

Figure 3: Phase space diagram

5 Intensity Data

The unpacked bins from histogram plot are vectorially operated to get the midpoints of bins. The code snippet and output for the default input arguments are presented below.

```
#intensity data
#using bins that are unpacked from plt.hist()
xpos=0.5*(bins[0:-1]+bins[1:])
with open('data.txt','w+') as f:
    print("Intensity Data",file=f)
    #tabulate helps in structuring the output data
    print(tabulate(np.stack((xpos,count)).T,["xpos","count"]),file=f)
print('Intensity data printed in data.txt file')
```

Intensity Data

xpos	count	xpos	count	xpos	count
0.5	0	35.5	132	70.5	106
1.5	0	36.5	108	71.5	86
2.5	0	37.5	58	72.5	81
3.5	0	38.5	93	73.5	84
4.5	0	39.5	98	74.5	85
5.5	0	40.5	89	75.5	84
6.5	0	41.5	92	76.5	106
7.5	0	42.5	116	77.5	85
8.5	0	43.5	88	78.5	87
9.5	217	44.5	103	79.5	112
10.5	243	45.5	101	80.5	77
11.5	150	46.5	84	81.5	79
12.5	137	47.5	102	82.5	82
13.5	90	48.5	70	83.5	89
14.5	135	49.5	107	84.5	90
15.5	96	50.5	88	85.5	104
16.5	86	51.5	82	86.5	84
17.5	107	52.5	73	87.5	85
18.5	115	53.5	107	88.5	99
19.5	109	54.5	88	89.5	79
20.5	92	55.5	116	90.5	108
21.5	81	56.5	87	91.5	90
22.5	173	57.5	118	92.5	79
23.5	77	58.5	98	93.5	88
24.5	142	59.5	85	94.5	77
25.5	85	60.5	92	95.5	66
26.5	120	61.5	117	96.5	54
27.5	133	62.5	92	97.5	23
28.5	88	63.5	63	98.5	12
29.5	83	64.5	78	99.5	3
30.5	69	65.5	108		
31.5	112	66.5	99		
32.5	101	67.5	91		
33.5	104	68.5	101		
34.5	107	69.5	108		

6 Dependence on Input Parameters

The simulation is set for different sets of parameters and the effect of threshold velocity, probability of collision is inferred

- u0 is set to 10 while remaining parameters are default values

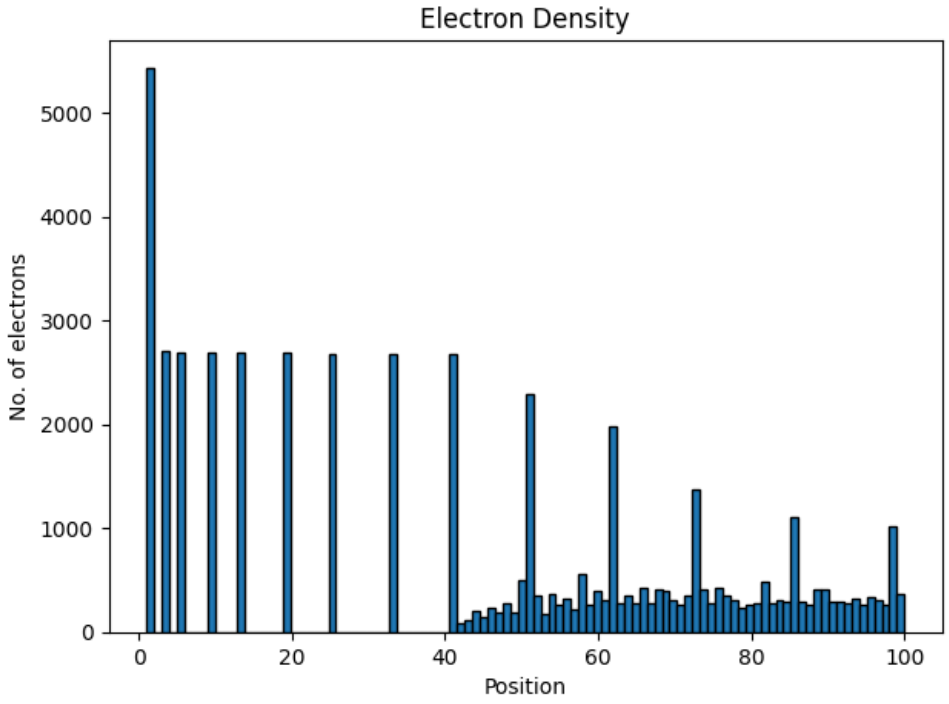

Figure 4: Histogram plot of Electron density for u0=10

The crowd of electrons is limited to specific locations until 40, whereas in the previous case this phenomenon was observed till position=10

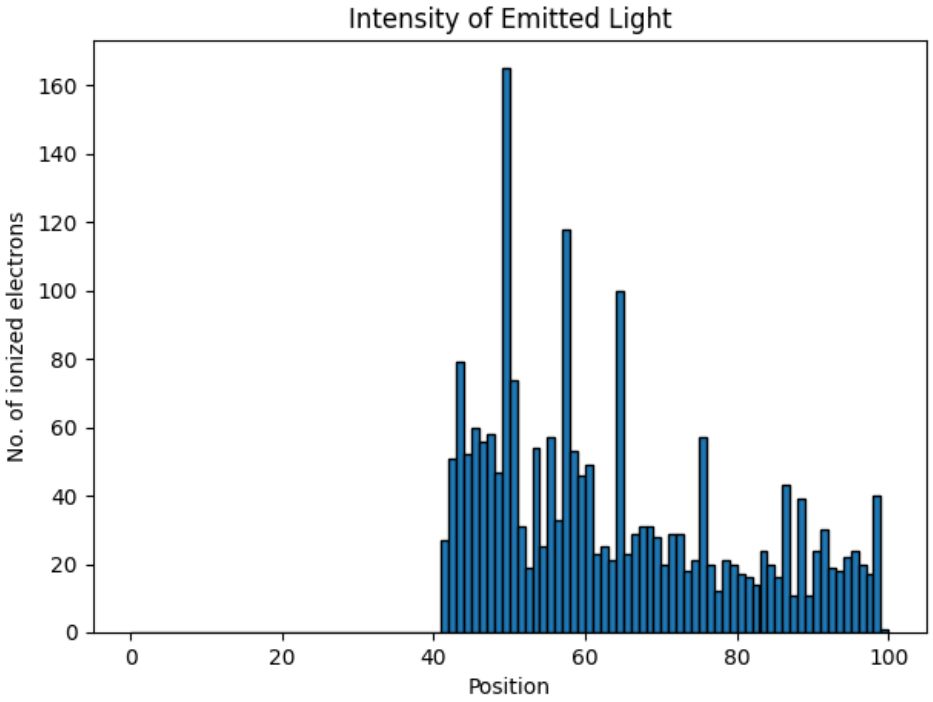

Figure 5: Histogram plot of Intensity for u0=10

The emission of light starts from a farther region from cathode and a larger dark space is observed due to increase in threshold velocity. Also, the frequency of ionized electrons oscillates faster w.r.t position compared to previous case.

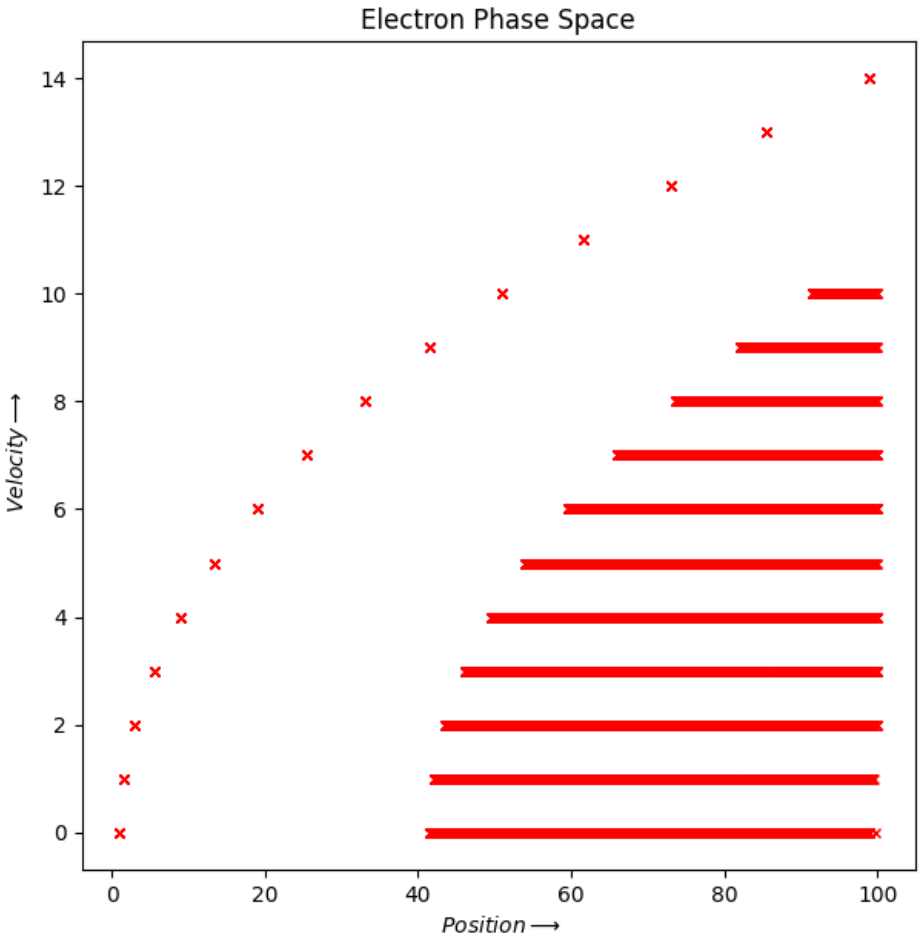

Figure 6: Phase space diagram for u0 =10

The phase trajectory of electrons which don't collide is invariant while the rest of them are shifted towards the anode side (right side).

- p is changed from 0.25 to 0.75 while the remaining parameters are default values

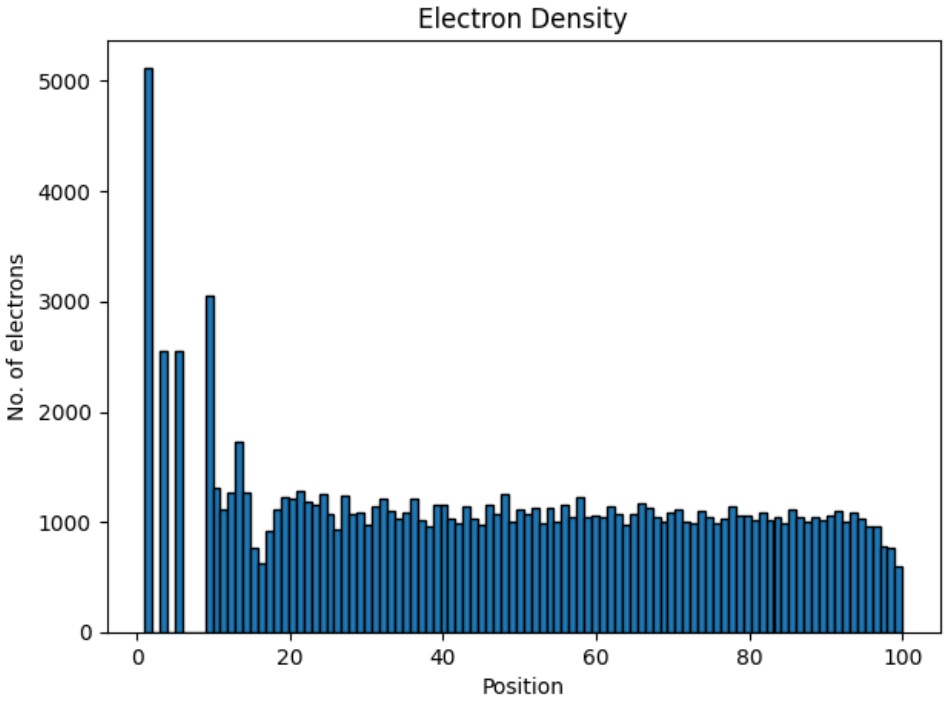

Figure 7: Histogram plot of Electron density for p = 0.75

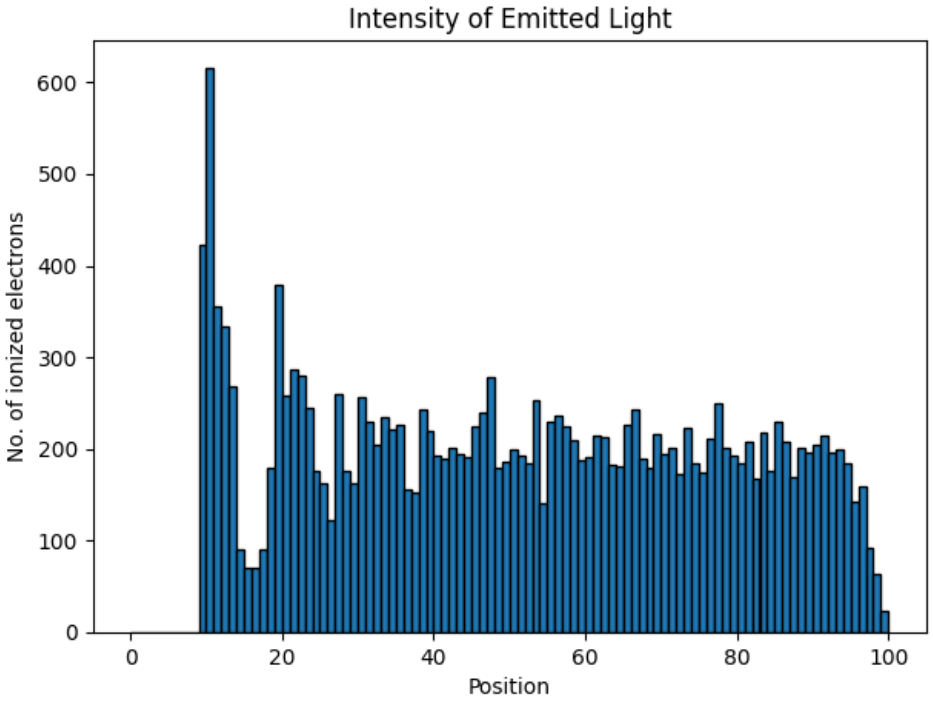

Figure 8: Histogram plot of Intensity for p = 0.75

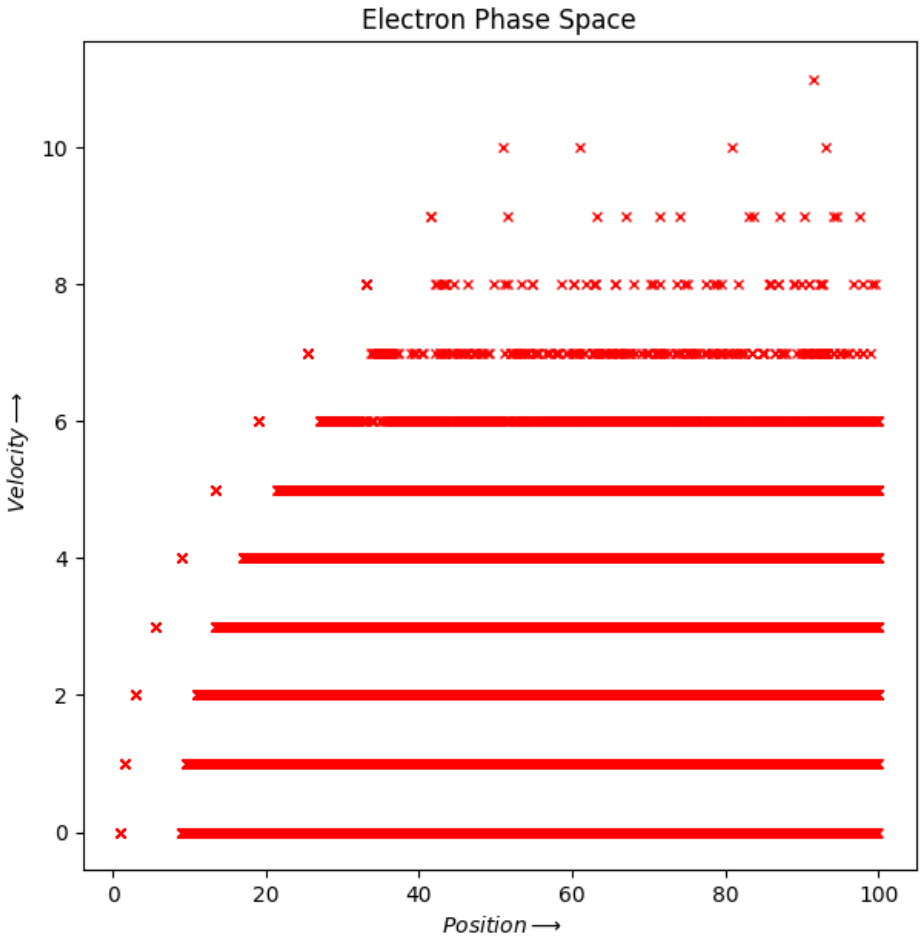

Figure 9: Phase space diagram for p = 0.75

The magnitude of intensity is high in case of greater probability. Also, a very sharp fall is found at the beginning of the spectrum, which can be treated as a dark space. The peaks in the electron density curve settle faster to steady value. The density of points in phase space for higher velocities has decreased as more no. of electrons are now colliding at a relatively lower velocities due to higher p.

7 Conclusion

The 1D Tubelight simulation can be performed in an iterative manner by simple functions and update of variables. The histograms and the phase space diagrams provide great information about the effect of various parameters on the behaviour of Tubelight. The intensity remains zero until electrons reach a threshold velocity. Lower threshold velocities are preferred so that dark spaces are minimal. Higher probability of electron collision is preferred as it results in increase in magnitude of intensity due to higher chance of collision. The fairly complex working

of a Tubelight is easily simulated with a few lines of code.