EE2703 Assignment 6 - Tubelight Simulation

Nihal John George (EE19B131)

April 11, 2021

Abstract

In this assignment, A 1D tubelight was simulated and analysed using Numpy, Scipy, and Matplotlib in Python.
The model is based on electrons being injected from the cathode, accelerated by a field over a distance, and ionizing gas atoms with some probability. The code simulates the process over a number of timesteps, handling the regular injection of electrons, the updates of their kinematic variables, and the random ionization.

The detailed process and observations from each of the plots is given below.

1 Initial Setup

Firstly, the variables of the simulation $n, M, nk, u0, p$ (spatial grid size, injection rate, total timesteps, threshold velocity and ionization probability, respectively) are taken from the user, or set to their defaults if not provided. This is done using argparse library which can handle arguments in different order and do type checking, which would otherwise be cumbersome using sys.argv. An example of defining a user parameter in argparse is given below.

```
ap.add_argument('--n', default=100, type=int, help='Spatial Grid Size')
```

2 Simulation Model

In this model, we instantiate electrons in each timestep and perform operations on the active electrons. For storing the active electron properties, we use the vectors xx (position), dx (displacement), and u (velocity). All active electrons have nonzero position, 0 is used as a marker for free slots for electrons in our simulation. This is necessary since there is no particular order in which electrons ionize, and it becomes necessary to reset stale electrons to prevent the simulation from taking too much space.

To take note of intensity and position properties over time, we use lists X and V to hold properties of active electrons for each timestep. It accumulates over each timestep, thus storing a large number of electrons.

3 Update Loop

1. Alive Electrons: Just before the loop, we initialise ii, the vector containing positions of alive electrons. This allows us to find the alive electrons exactly once at the end of each iteration, as requested by the QP.

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

2. Acceleration Update: Then we update the kinematic variables due to field acceleration on the alive electrons.

```
dx[ii] = u[ii] + 0.5
xx[ii] += dx[ii]
u[ii] += 1
```

3. Anode Condition: The electrons which have hit the anode are reset to free slots using Numpy where() function.

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

4. Random Ionization: The alive electrons then ionized with probability p if they are energetic enough

```
high_el = np.array(np.where(u>=u0)[0])   # kk
ion_ind = np.array(np.where(np.random.rand(len(high_el))<=p)[0])     #ll
ion_el = high_el[ion_ind]   # kl
```

5. Random Position Reset: The ionized electron positions are reset to a distance randomly distributed between the two timesteps. The QP also asked to provide the correct code since the collision is uniformly distributed in time, but not space. Below is the derivation of the formula, it has been commented out in the code.

$$ds = ut + 0.5at^2 \qquad dx = u + 0.5$$

Now the fraction of dx actually travelled is given below, and the true position follows.

$$\frac{ds}{dx} = \frac{ut + 0.5t^2}{u + 0.5} \implies x_{true} = x_{calc} - dx(1 - \frac{ds}{dx})$$

```
u[ion_el] = 0
rho = np.random.rand(len(dx[ion_el]))
xx[ion_el] -= dx[ion_el]*rho

# Formula taking into account s=ut+0.5at^2 given below
# xx[ion_el] -= dx[ion_el]*(1-np.divide((u[ion_el]*rho) + 0.5*(rho**2), u[ion_el]+0.5)
```

6. Accumulate Intensity: We then store the intensities by assuming that each ionized electron gives out the same energy photon.

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

7. Injection: New electrons are injected by sampling from a normal distribution and rounding to an integer. They are assigned to free slots in the xx array, bounded by the number of free slots itself. The bound is implemented by list slicing.

```
m = max(int(np.random.randn()*Msig + M),0)
free = np.where(xx==0)
free = free[:m]
xx[free] = 1
```

8. Accumulate Kinematics: The position and velocity of alive electrons at the the end of this timestep are accumulated in X and V.

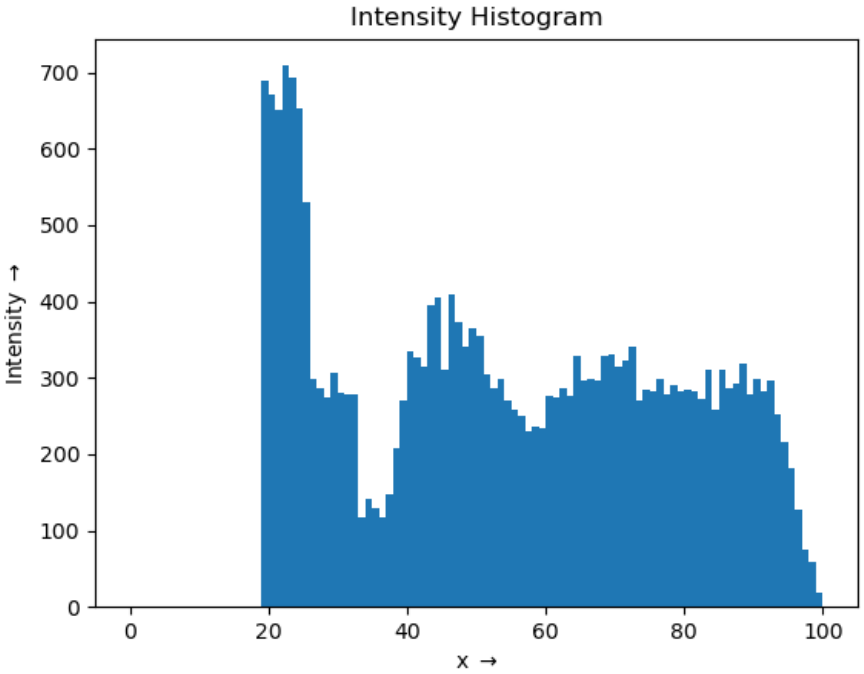
```
ii = np.where(xx>0)[0]
X.extend(xx[ii].tolist())
V.extend(u[ii].tolist())
```

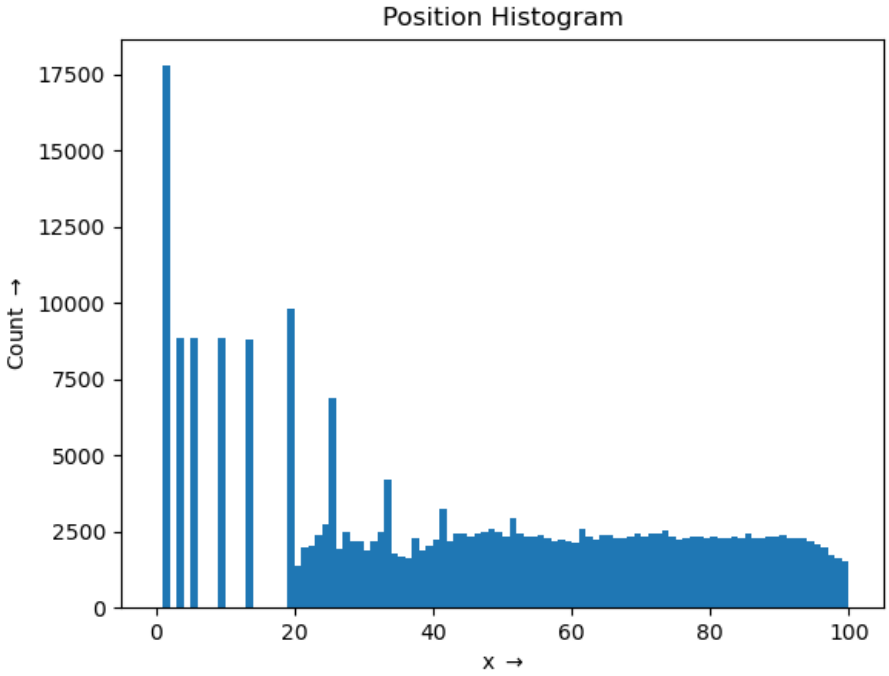
4 Visualizations

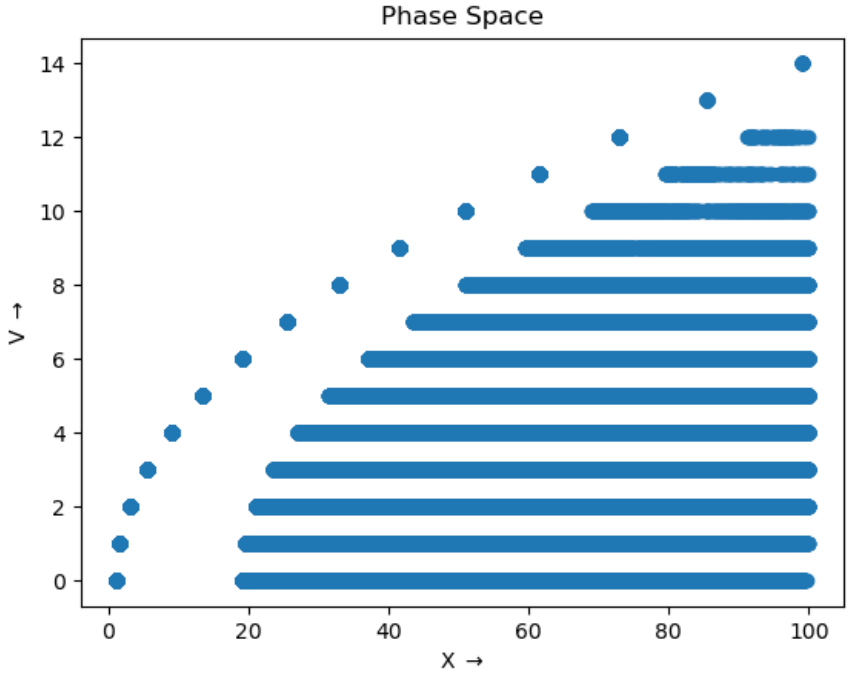
Histogram of position of electrons is plotted using plt.hist(). Custom bin edges are used since the default auto-adjusts to a different range.

```
bin_edges = [i for i in range(n+1)]
```

The intensity at different parts of the tubelight is also plotted similarly.
The phase plot is done using plt.scatter().
For $u0 = 7$ and $p = 0.5$, the graphs are shown below

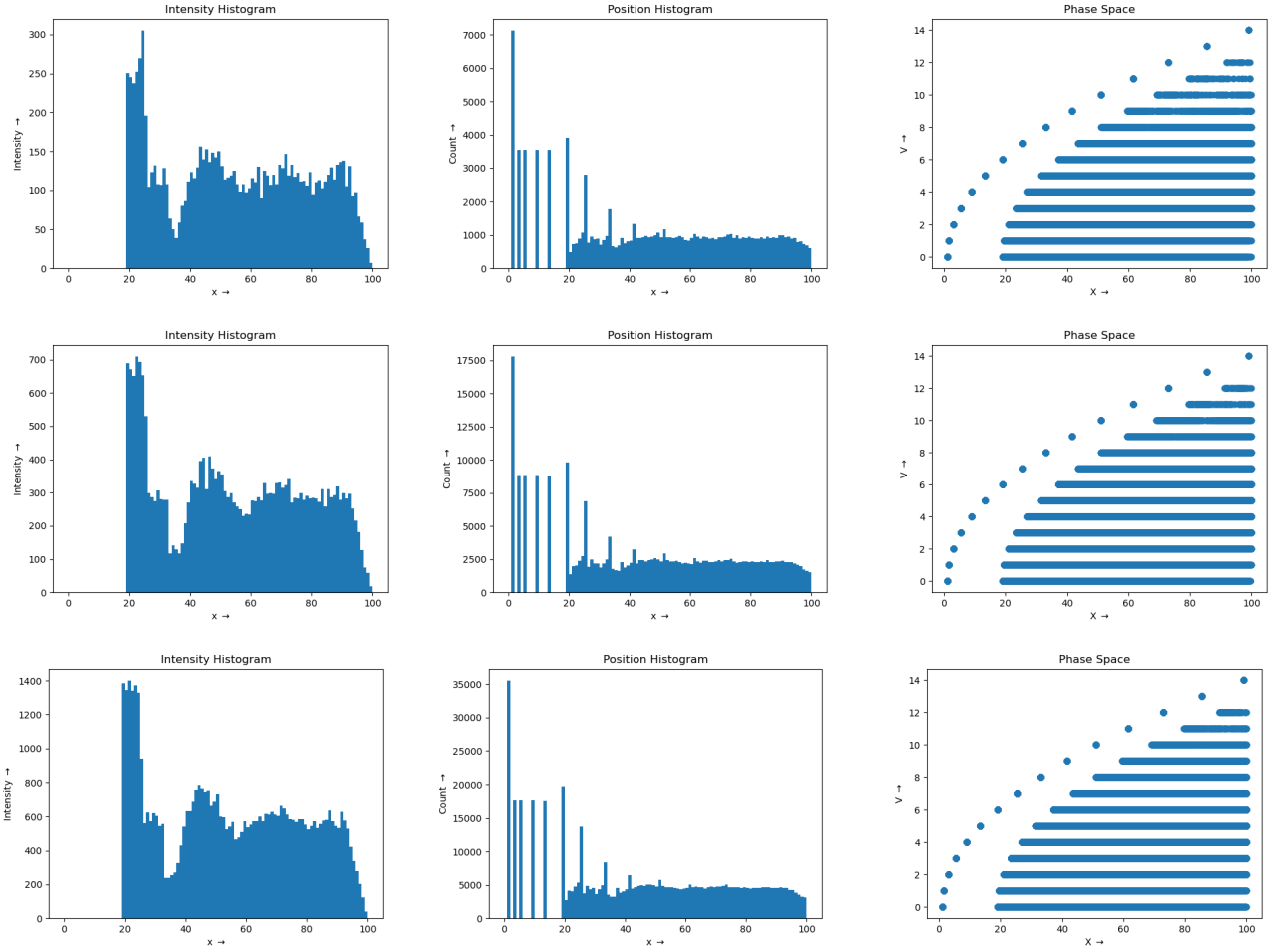






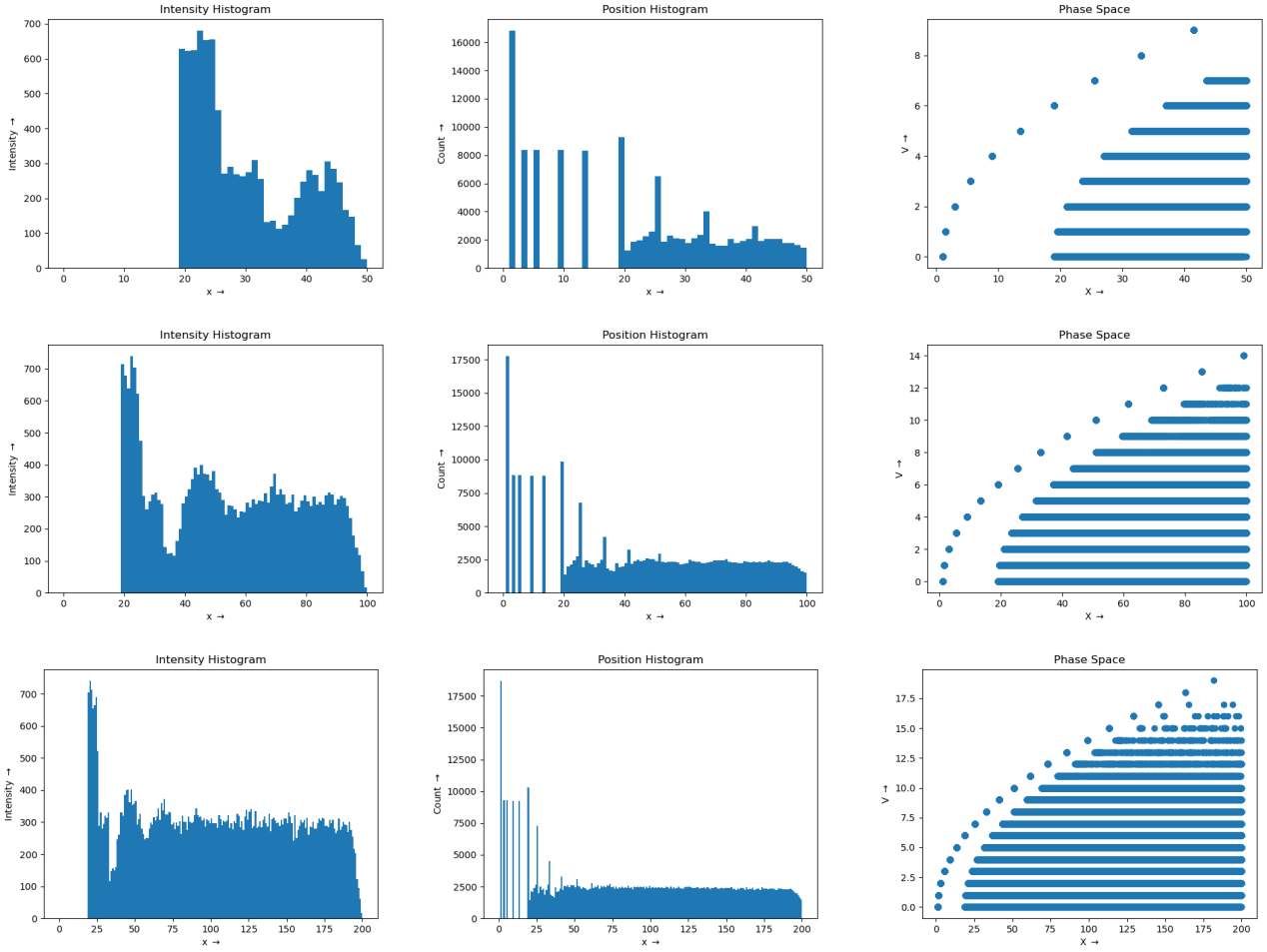
Each experiment was performed using the default parameters, and varying one parameter in both directions. Totally 6 experiments were performed.
Two of them ($Msig$ and nk) did not show any difference, and hence not plotted here. This is because the though the standard deviation $Msig$ adds some randomness, the mean remains same, which makes the mean injection over time constant.
Number of iterations nk just makes the simulation more precise, the actual curve shapes do not change unless nk is very small, in which case the transient state is prominent.
The remaining 4 experiments' plots are given in the next few pages. 3 values are used for each – lower, default, higher (in that order row-wise).

Experiment with M (Injection rate) - 2, 5, 8



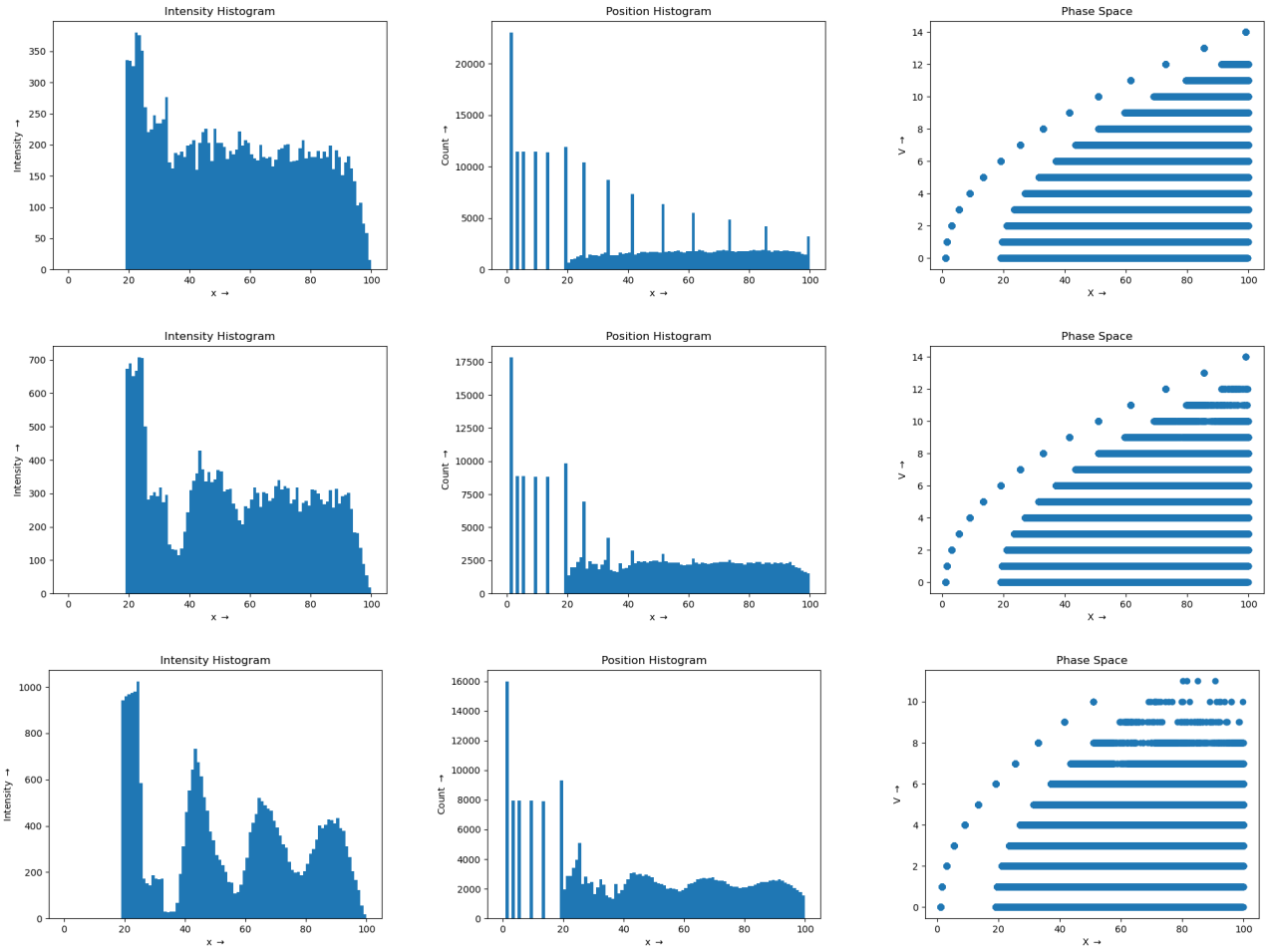
In this experiment, the curve shapes are fundamentally the same. The only difference is in scaling in number of electrons, which can be noticed from the y-axis of intensity and position histograms.

Experiment with n (Tube Length) - 50, 100, 200



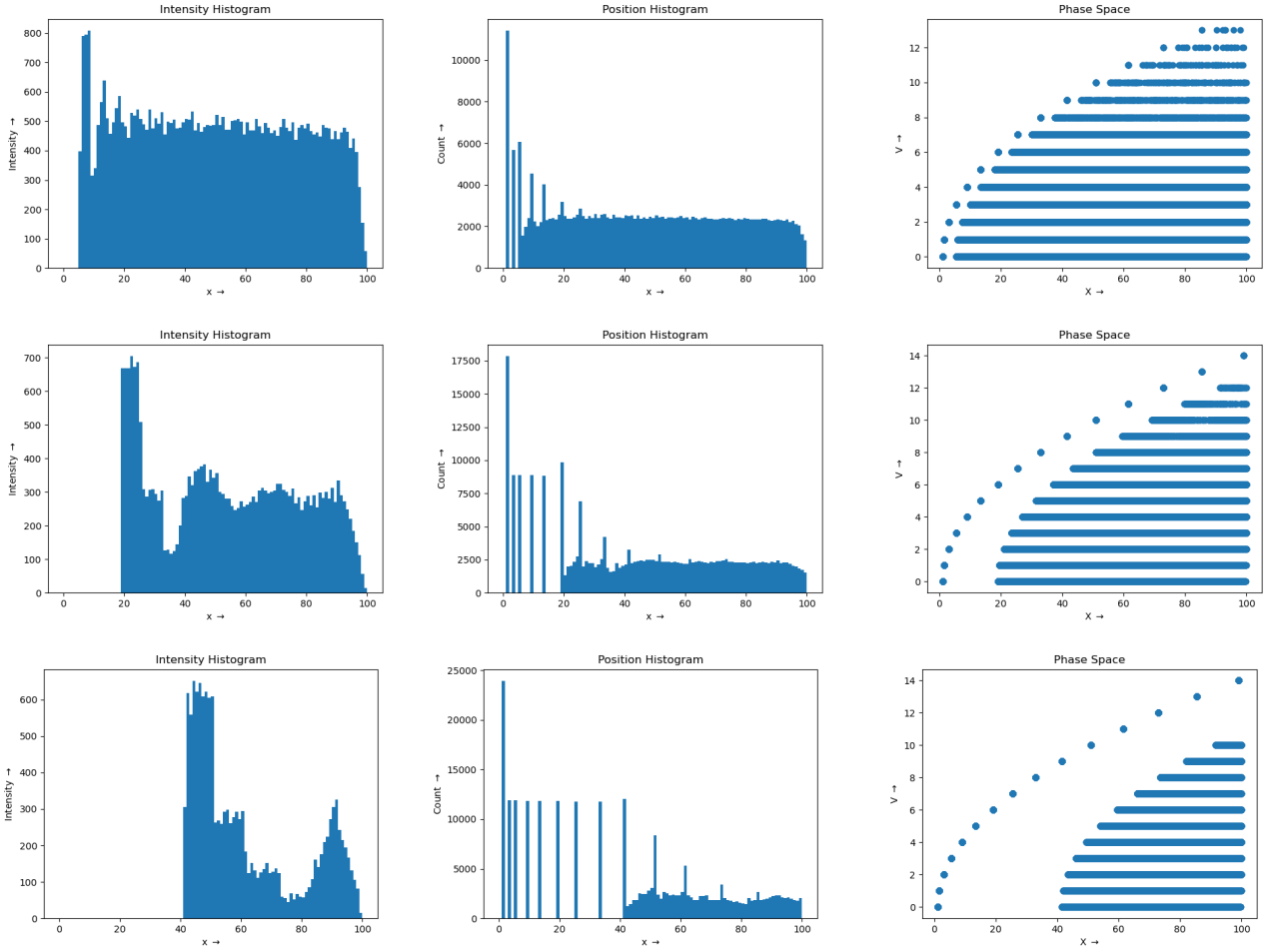
In this experiment, we notice that emission always starts at around x=20. We also notice that the curve structure does not get stretched with tube length, instead the plateau extends. So truncating the tube to smaller lengths has negligible effect on the curve shape at a particular x.

Experiment with p (Ionization Probability) - 0.2, 0.5, 0.8



When p is higher, we find more peaks, higher height of each of these peaks and lower height of the troughs. We can think of the case when p=1. In this case, since other parameters are fixed, ionization will always occur at fixed points on the x-axis – so very tall, sharp peaks. Now in the case of p=0.001, rarely does an ionization take place. So the electron may be located nearly anywhere on the x-axis before ionization.
So increasing p will sharpen the peaks.

Experiment with $u0$ (Threshold Velocity) - 4, 7, 10



Increasing $u0$ means increasing the distance over which an electron needs to gain speed before capably ionizing an atom. This is indicated by zero intensity in the leading region of each plot, which increases with $u0$.

Conclusion

This assignment explored the simulation of a 1D tubelight using Python.

- The intensity plots start with a zero intensity leading area, after which there is emission. The starting of the emission is always the tallest peak, and there are peaks which occur, decreasing in height, due to the Bernoulli probability distribution of emission (p for first emission, p^2 for 2nd emission and so on). The peaks are higher than the distribution near them, caused due to the randomness.
- The position plot has equal height peaks in the dark region since no emissions take place, so all snapshots of position take place according to $s = ut + 0.5at^2$ with integral values of t, hence the same positions for each electron in the dark region. In the emitting regions, again there are geometrically decreasing peaks, and a background distribution below these peaks.
- The phase plots had one set of dots on the left, and lines to the right of them. This set is of those points which never emit, and very less in number since the probability of never hitting decreases quickly. There are lines since there are only discrete increments of velocity that happen over complete seconds, so no other velocities than these are observed. However, a range of positions are seen, since the emission can happen anywhere after the threshold velocity is reached. Also, these lines never exist before a point, x=20 for the default values. This is because it is the first acceleration after being born, and emission can happen only after reaching thresold velocity.

After this, from the plots obtained, we observe that –

1. M: Increasing injection rate will uniformly increase intensity.
2. n: Increasing tube length will not change the plot on the original length (if field acceleration constant). The emission peaks are preserved at the same locations.
3. p: Increasing p sharpens the peaks, causing distinct dark and bright areas.
4. u0: Increasing u0 increases the dark leading region, and increases the peak separation.

In my opinion, a good tubelight has to be uniformly lit and bright enough. For this, a high M, low p, low u0 tubelight should be chosen.
Through this assignment, the student has been acquainted with the thought process of simulating models of physical systems over time in Numpy, and useful plotting functions in Matplotlib. A number of tricks using np.where() and conditional indexing were also learnt in the spirit of vectorization and quick runtime.