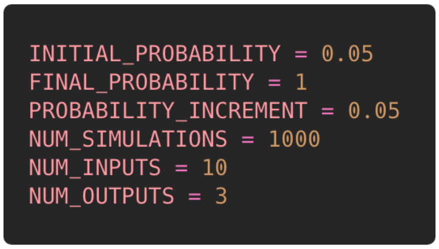
Project #1:
Switching Element Problem

Mankirat Gulati
ID: 111161128
CSE 346

Source Code

This program was written in **Python 3** and uses the libraries, **Numpy** to calculate the Bernoulli trials and **matplotlib** to plot the graphs.

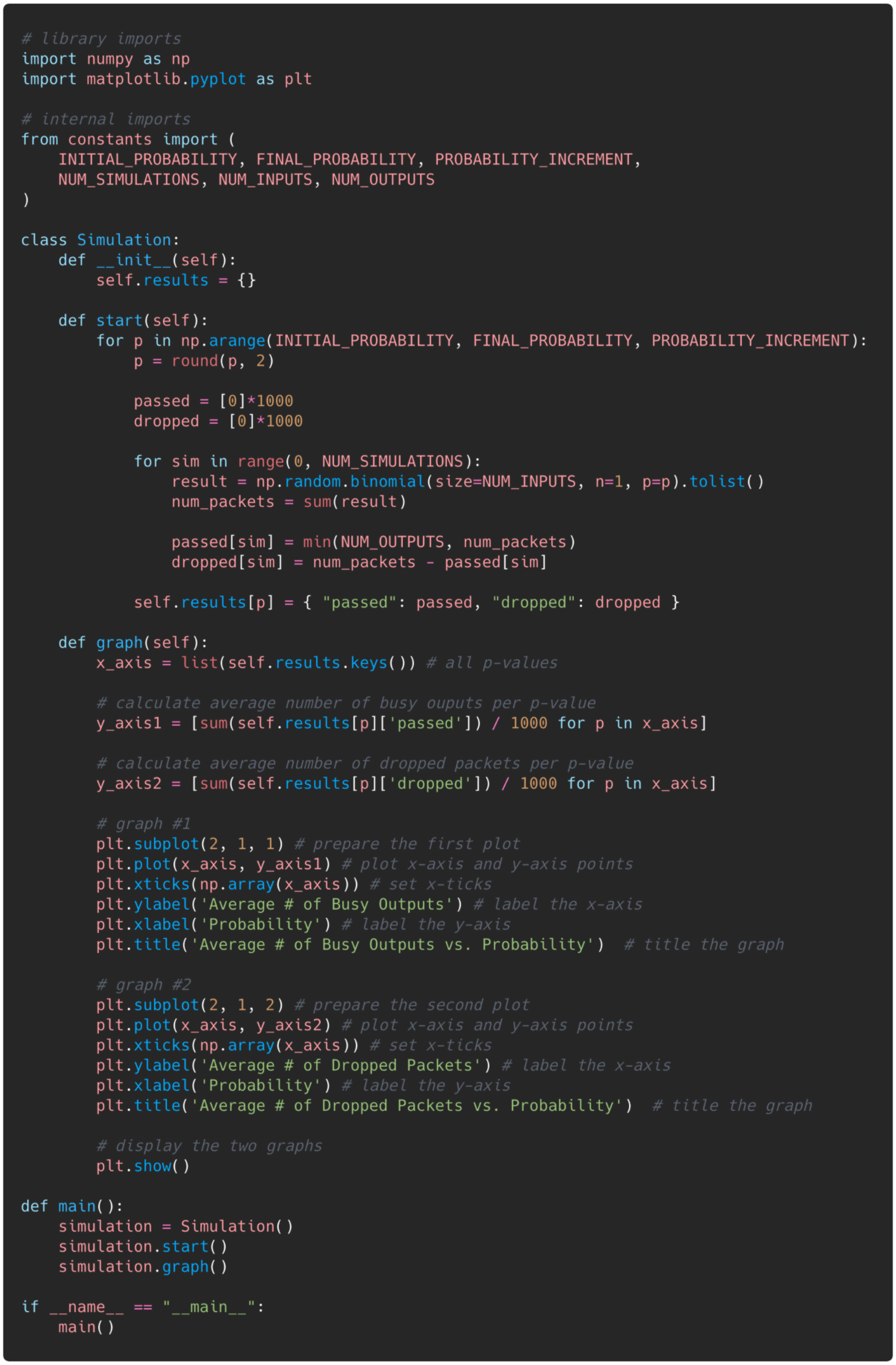
constants.py



```
INITIAL_PROBABILITY = 0.05
FINAL_PROBABILITY = 1
PROBABILITY_INCREMENT = 0.05
NUM_SIMULATIONS = 1000
NUM_INPUTS = 10
NUM_OUTPUTS = 3
```

File containing important constants for the program to work.

simulation.py



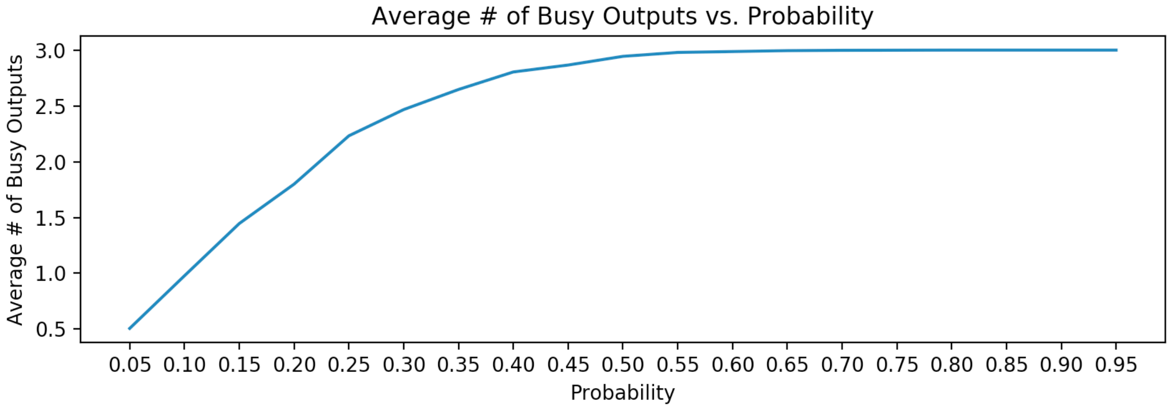
```python
# library imports
import numpy as np
import matplotlib.pyplot as plt

# internal imports
from constants import (
    INITIAL_PROBABILITY, FINAL_PROBABILITY, PROBABILITY_INCREMENT,
    NUM_SIMULATIONS, NUM_INPUTS, NUM_OUTPUTS
)

class Simulation:
    def __init__(self):
        self.results = {}

    def start(self):
        for p in np.arange(INITIAL_PROBABILITY, FINAL_PROBABILITY, PROBABILITY_INCREMENT):
            p = round(p, 2)

            passed = [0]*1000
            dropped = [0]*1000

            for sim in range(0, NUM_SIMULATIONS):
                result = np.random.binomial(size=NUM_INPUTS, n=1, p=p).tolist()
                num_packets = sum(result)

                passed[sim] = min(NUM_OUTPUTS, num_packets)
                dropped[sim] = num_packets - passed[sim]

            self.results[p] = { "passed": passed, "dropped": dropped }

    def graph(self):
        x_axis = list(self.results.keys()) # all p-values

        # calculate average number of busy ouputs per p-value
        y_axis1 = [sum(self.results[p]['passed']) / 1000 for p in x_axis]

        # calculate average number of dropped packets per p-value
        y_axis2 = [sum(self.results[p]['dropped']) / 1000 for p in x_axis]

        # graph #1
        plt.subplot(2, 1, 1) # prepare the first plot
        plt.plot(x_axis, y_axis1) # plot x-axis and y-axis points
        plt.xticks(np.array(x_axis)) # set x-ticks
        plt.ylabel('Average # of Busy Outputs') # label the x-axis
        plt.xlabel('Probability') # label the y-axis
        plt.title('Average # of Busy Outputs vs. Probability')  # title the graph

        # graph #2
        plt.subplot(2, 1, 2) # prepare the second plot
        plt.plot(x_axis, y_axis2) # plot x-axis and y-axis points
        plt.xticks(np.array(x_axis)) # set x-ticks
        plt.ylabel('Average # of Dropped Packets') # label the x-axis
        plt.xlabel('Probability') # label the y-axis
        plt.title('Average # of Dropped Packets vs. Probability')  # title the graph

        # display the two graphs
        plt.show()

def main():
    simulation = Simulation()
    simulation.start()
    simulation.graph()

if __name__ == "__main__":
    main()
```

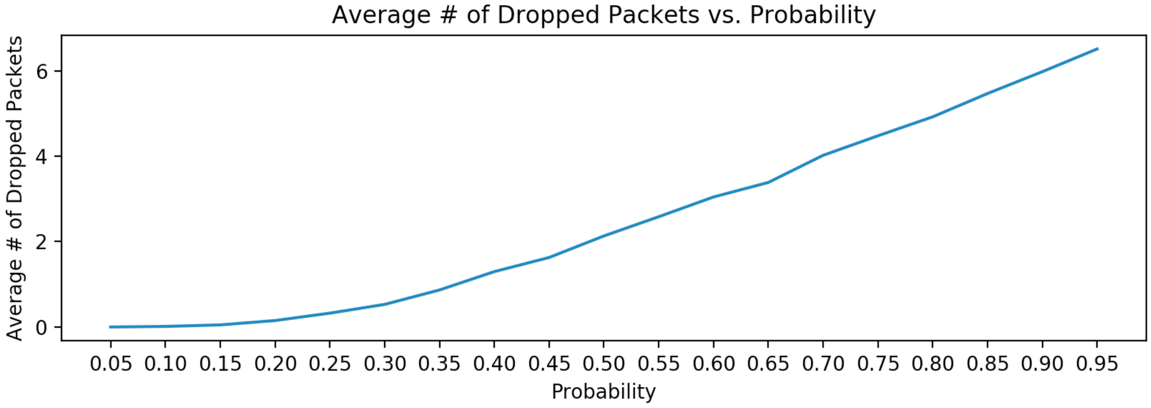
The full source code that handles the simulation and plotting of graphs.

Program Algorithm

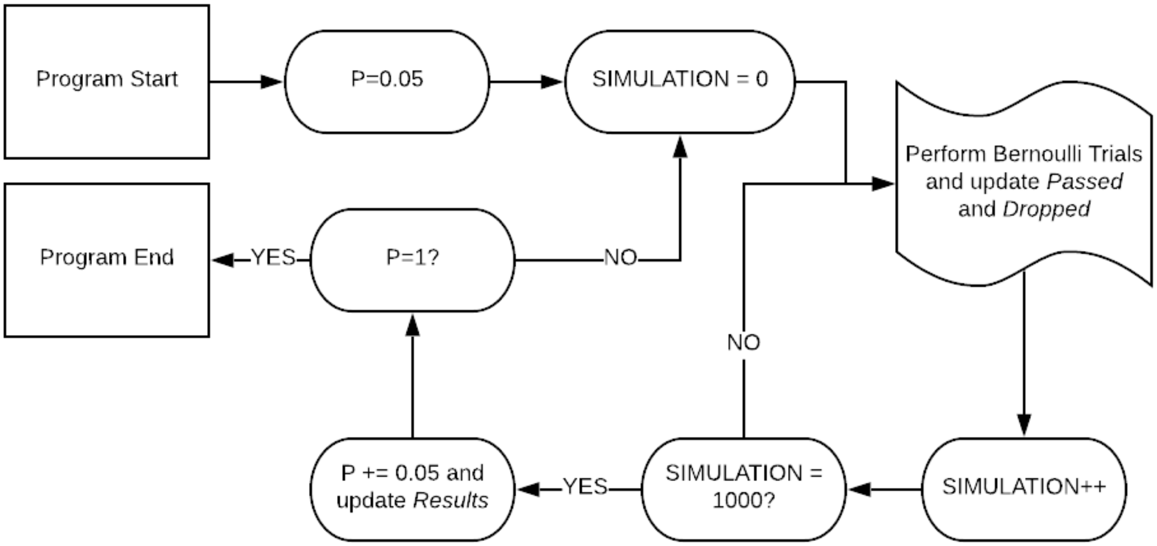
- For each value of p in $[0.05, 1)$ or $[\mathrm{INITIAL\_PROBABILITY}, \mathrm{FINAL\_PROBABILITY})$, I create two arrays, *passed* and *dropped* which contain the number of packets that went through and the number of packets that were dropped, respectively.
- I loop from 0 to the 1000 $(NUM\_SIMULATIONS)$ and perform 10 (NUM_INPUTS) Bernoulli trials for each simulation.
- I get the number of packets sent by summing up the 1s in the *result* array.
- The *passed* array is updated with the number of packets sent or 3 $(NUM\_OUTPUTS)$ if $num\_packets > 3$.
- The *dropped* array is updated with the number of packets minus the number of passed packets.
- The *results* dictionary is updated the probability p as the key and the two arrays, *passed* and *dropped* as values.

Graphs





Flow Chart


Program Start
P=0.05
SIMULATION = 0
Perform Bernoulli Trials and update *Passed* and *Dropped*
Program End
YES
P=1?
NO
NO
P += 0.05 and update *Results*
YES
SIMULATION = 1000?
SIMULATION++

Conclusion

- Looking at first graph (Busy Outputs vs. Probability), we can see it first started out as an increasing linear relationship until around $p = 0.55$. From that probability onwards, the graph levels out and any probability above that points to an average of 3 busy outputs. With the second graph, (Packets Dropped vs. Probability), we can see a fairly consistent increasing linear relationship. This makes sense because the switch can only accept 3 packets at a time. If we increase the probability, we increase the number of packets that arrive. With more arriving packets, we have more dropped packets since the switch can only accept 3 at any time. Performance wise, this switch is fairly inefficient considering it can only handle a maximum of 3 packets at a time.