PHY 4004: Experiment 3

Monte-Carlo Simulation for Population Growth including Non-linear Saturation Effect

Statistical Mechanics with Applications to Mathematical Finance

Omar Gabr
Baruch College
EMPLID: 23513476
October 29, 2021

1 Theory

1.1 Abstract

In this report, I will be computing the statistics for population growth using the Monte Carlo method in Python. I will first begin by introducing the mathematics behind population growth and the Monte Carlo method, then I will analyze the data through a Python simulation, and finally compute for the average evolution and standard deviation of the population growth.

1.2 Physics

It has been shown by French mathematician, Pierre Francois Verhulst, that the saturation of a population at N_{max} is

$$\frac{dN}{dt} = rN\left(1 - \frac{N}{N_{max}}\right)$$

such that N is the current population, r is the growth rate, and t is the time. This is otherwise known as the logistic equation.

We can find a solution to our differential equation as the following:

$$N(t) = \frac{N_0 \cdot e^{rt}}{\frac{N_0}{N_{max}} \cdot e^{rt} - \frac{N_0}{N_{max}} + 1}$$

such that $N = N_0$. This solution to our logistic equation will help serve as our analytical solution when comparing it to the solution we get from the Monte-Carlo method.

1.3 Monte Carlo Method

The Monte Carlo method is used to model the probability of different outcomes in a process that cannot easily be predicted. This method relies on random sampling to obtain data. In our context, the Monte Carlo simulation can help predict possible growth rates of populations over time. I will be using this method to approximate the average evolution and standard deviation of the probability distribution.

We know that the size of the time-step should be smaller than $\frac{1}{r}$. This means that we can get our time-step, h, as a factor of our growth rate.

This is the psuedo code to implement the Monte Carlo method:

1. probability = growth rate * time-step
2. for each trial in N simulations:
   if the random value is less than the computed probability, then
   increment the counter variable
3. once done iterating the simulations, set the population as the current population
4. append population to the array

2 Data and Analysis

```
# initial conditions
N_0 = 2          # initial population
N_max = 10**4    # saturation growth
growth = 0.5     # growth rate
t = 4.0/growth   # time

# time-step per growth rate
# h << 1
h = 0.01 / growth

# simulations
simulations = 4 * 10**3
# array of N values
N_array = []
```

The code above presents the initial conditions of our population growth with the following values:

1. N_0 is the initial population of 2 people
2. N_{max} is the saturated population at 10,000 people
3. $growth$ is the growth rate at 0.5 per year
4. t is the time of our program defined as $\frac{4}{growth}$

```
for r in range(simulations):
    N = N_0
    t = 0.
    while (t < 4.0/growth):
        N_i = N
        for i in range(N):
            # probability to win
            probability = growth * h
            if (random.random() < probability):
                N_i += 1
            # probability to lose
            probability = growth * h * N / N_max
            if (random.random() < probability):
                    N_i -= 1
        t += h
        N = N_i
    N_array.append(N)
```

The above code is the implementation of the Monte-Carlo method as explained in the introduction. This program helps compute a probability distribution for prediction of our population growth over time.
More specifically, since we are calculating the probability per time-step, I want to iterate over N simulations and increment the counter variable if the random number generator is less than the probability per time-step. This will be considered as the probability to win.
On the other hand, to calculate the probability of loss, then we calculate the probability per time-step times the ratio of the population to saturation. Then I will decrement the probability from the array.
After calculating the probabilities, I will end it with adding the time-step to the total time while it is less than $\frac{4}{r}$ and append the current population to our array.

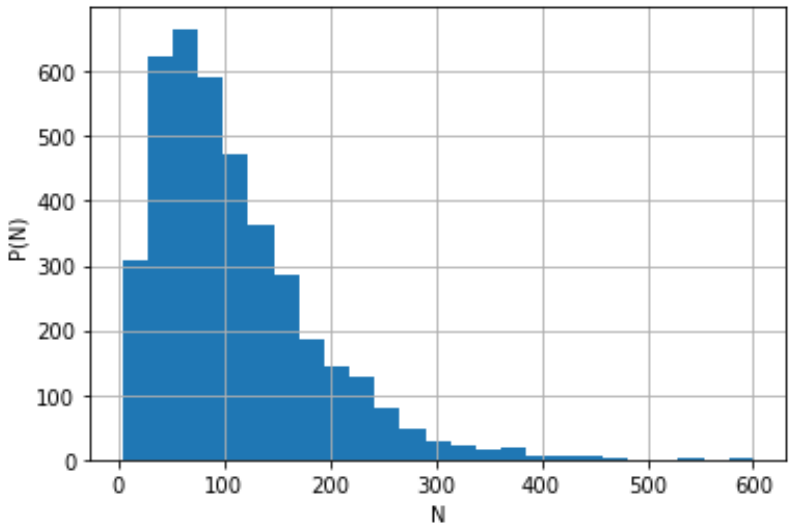
```
# statistics
# average evolution <N(t)>
N_average = N_0 * math.exp(t*growth) / (1 + N_0/N_max * math.exp(t*growth))
s = np.std(N_array) # standard deviation for each N per time-step

# statistics output
print("Average evolution: N(t*growth=4.0) = ", N_average)
print("<N(t)> = ", np.mean(N_array), " +/- ", s/math.sqrt(simulations))
print("standard deviation (sigma) = ", s)
```

In the code above, $N_{average}$ is the solution to our analytical equation introduced earlier. I also included the average and standard deviation of the population growth from the Monte-Carlo method to compare both results.

```
In [5]: runcell(0, '/Users/O.G/Desktop/MC_populationgrowth.py'
Average evolution: N(t*growth=4.0) =  109.09054074093541
<N(t)> =  107.98775  +/-  1.1901255816023681
standard deviation (sigma) =  75.27015078992136
```

When executing the program where $t = 4 \cdot \frac{1}{r} = 8$, these are the resulting statistics:

1. The average evolution is approximately 109 people per the growth rate. This is the analytical solution of our population growth.
2. The mean, using the Monte-Carlo method, approximates the population to 108 ± 1 people.
3. The standard deviation of our mean is approximately 75 people.



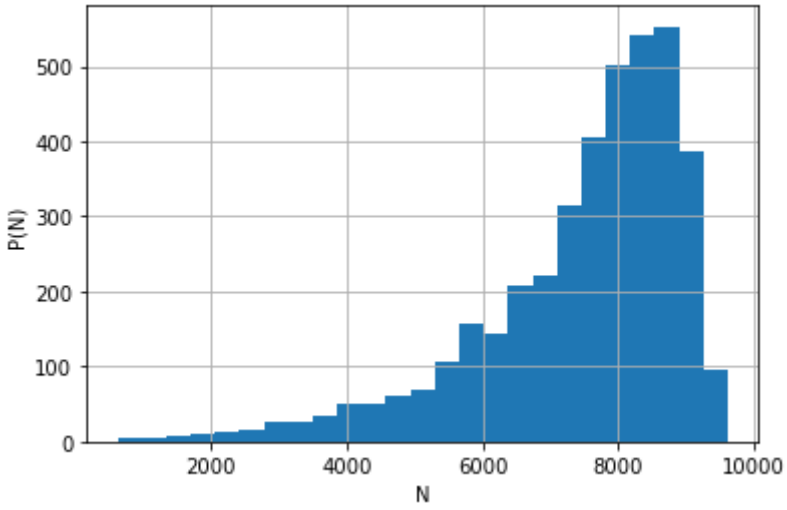
The figure above displays the probability distribution of our population growth where the time $t = \frac{4}{r}$ such that r is the growth rate. This histogram reflects the population growth over 8 years.

Now we will look at a different time-scale when $t = \frac{10}{r}$.

```
In [1]: runcell(0, '/Users/O.G/Desktop/MC_populationgrowth.py')
Average evolution: N(t*growth=10.0) =  8164.991854851616
<N(t)> =  7467.6325  +/-  23.89247785100862
standard deviation (sigma) =  1511.0929790862474
```

When executing the program where $t = 10 \cdot \frac{1}{r} = 20$, these are the resulting statistics:

1. The average evolution is approximately 8165 people per the growth rate. This is the analytical solution of our population growth.
2. The mean, using the Monte-Carlo method, approximates the population to 7468 ± 24 people.
3. The standard deviation of our mean is approximately 1511 people.



Similarly to the histogram above, I also modeled the probability distribution for when $t = \frac{10}{r}$. This distribution reflects the population growth over 20 years.

3 Conclusion

As we can infer from our probability distributions using $t = \frac{4}{r}$ and $t = \frac{10}{r}$, that the population grows exponentially as time increases.
We can also see that the Monte-Carlo method becomes less accurate over time. One may argue that the time-step used could have been smaller, although the simulation of such small values would take too long for my software to process. As mentioned before, the Monte-Carlo is good for prediction, and so we do not expect to get accurate results necessarily.