Project 4: Financial Engineering from a Statistical Physics Approach

Michael Saybolt

5/7/2017

Contents

Abstract

This project explores the generation and simulation of an economic system. Financial transactions among agents are simulated using Monte Carlo methods. The resulting distribution of wealth is then studied. Various parameters can be tuned to create a system that accurately models real wealth distributions from various countries.

1 Introduction

The end goal is to analyze the distribution of wealth as a function of agents' money/income, m. The result should follow a Pareto distribution [1]. In addition, the model was made more realistic by following the work of Patriarca and collaborators, adding a quantity λ responsible for making the agents save a certain fraction of their money during a transaction [2]. Following the work of Goswami and Sen, the agents can be further enhanced to model tendencies to interact with other agents of similar income, as well as previous interactions[3].

2 Technical Details

2.1 General Considerations

In all levels of modeling, there are some basic considerations that may differ slightly from the papers. In general, the distribution of money follows

$$w_m \propto m^{-1-\alpha},$$

with $\alpha \in [1, 2]$[1].

A certain number of agents N are created for each Monte Carlo iteration, and they are randomly selected to perform transactions. Unlike the paper, they are each assigned some equal start money, m_0, instead of a random value from a uniform distribution. The choosing of the agents, however, is still done using a uniform distribution. `For` some number of Monte Carlo cycles, `for` some number of transactions, a pair of agents (i,j) are chosen by the uniform distribution, and exchange money. Money is conserved, thus

$$m_i + m_j = m'_i + m'_j. \tag{1}$$

The random number ϵ, generated from a uniform distribution, is used to determine the fraction of money that is changed. The money of agents' i and j become

$$m'_i = \epsilon(m_i + m_j),$$

leading to

$$m'_j = (1 - \epsilon)(m_i + m_j).$$

No agents can go into debt (only finitely smaller transactions), and due to the conservation law, the system relaxes towards a Gibbs distribution.

2.2 Parameter Choice

Parameters are chosen initially such that when the system reaches its termmination condition, there are not 'holes' in the data set. Without sufficient interactions, and more importantly enough Monte Carlo cycles to even out the dataset, there will be parts of the distribution of wealth with no agents in that section. Taking the log of the distribution of wealth will look uneven and be inaccurate as a result, when it should look relatively linear for the most basic economic model (no money saved or other caveats to interaction). Additionally, sufficient start money seemed to play a role (even though it seems it shouldn't, as all floats tested can be divided or multiplied many times themselves without significant accuracy loss), however there was not enough CPU time to verify this trend.

2.3 Termination Conditions

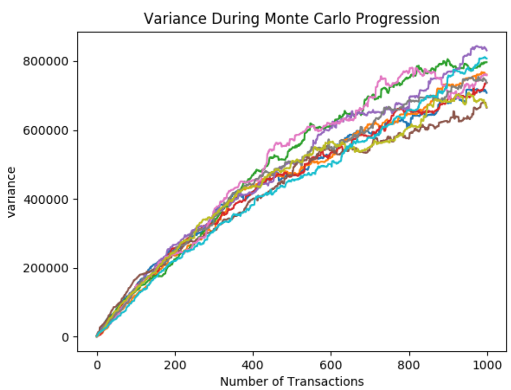
The termination condition is met when the system has reached some sort of equilibrium. This can be guessed by looking at the data and seeing when it quits changing much with additional transactions, or for a more quantatative approach, the variance amongst the agents can be observed as the transactions go on, which will stabilize when the system reaches equilibrium. Figures 1a, 1b, and 1c show the variance against number of transactions and 10 Monte Carlo cycles for clarity. Choosing `num_transactions` to ensure the system is at equilibrium is no longer a guessing game. This tends to be very predictable, and unless the number of agents `N` is changed, the system's convergence to equilibrium is easily predicted within an order of magnitude.

3 Big Data Challenges

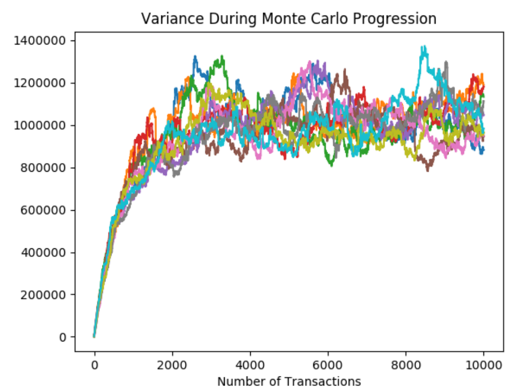
3.1 Dealing with Large Sets of Data

Unlike previous projects, Monte Carlo simulations involve a lot of CPU time to generate a sufficient pool of pseudo randomly generated data. While some general trends can be observed with less iterations for debugging, to truly check the details, sufficient numbers of experiments must be done.

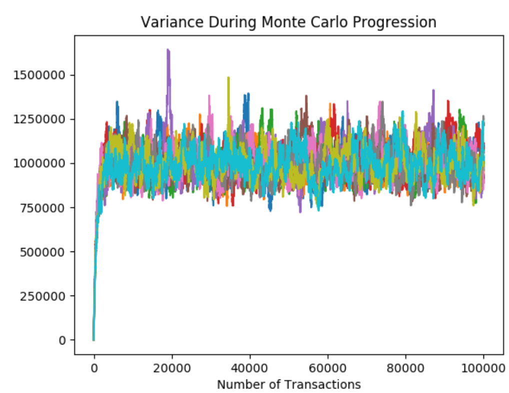
This prompted the need for modularizing the code, building a libray for commonly used functions, adding load and store functions, as well as a separate program `quikplot` to just plot data. Running through all that computation just to plot something and then discard it after making changes to the code is incredibly wasteful and inefficient, so after the cycles complete, it optionally writes to a file and/or plots, using functions contained in a shared financial library. This way, a separate plotting function can load the



(a) `num_transactions`$=10^3$



(b) `num_transactions`$=10^4$



(c) `num_transactions`$=10^5$

Figure 1: Variance of 10 MC cycles with `N` $= 500$

files into variables and call the exact same plotting function to recreate and display that environment.

3.2 Computation of Large Sets of Data

3.2.1 Parallelization Attempt

When dealing with such a CPU intensive task, parallelization immediately comes to mind. Unfortunately parallelizing the Monte Carlo cycles did not seem trivial, at least in the stage of the code that was first attempted to be parallelized. This seemed strange, as individual cycles do not depend on each other unless the termination condition relies on comparing other cycles and must index them numerically during computation.

3.2.2 Crude but Effective Method

Since CPU time is of essence near the end of the semester, utilizing as much of it as possible is still a top priority. While my 4 GHz antenna simulation targeted desktop can kick out more flops than the engineering building's compute servers core-per-core, the compute servers have many, many more cores. Just dying for a parallel implementation of this code. Since parameters needed to be varied for the extra details and caveats of the transactions, multiple python instances can be started separately, and then data combined at the end, programatically, but not truly 'parallel' code.

Of course care was taken to use the least loaded one, but running 30+ instances of Python on cores half as fast was still a much faster way to acquire large sets of reasonably accurate data!

4 Results

A formal discussion of the results of adjusting the model to more accurately represent real economic principals is presented.

4.1 Default

The 'default' model simply follows exponential decay as mentioned in the introduction [1]. Agents do not save any money, nor do they discriminate

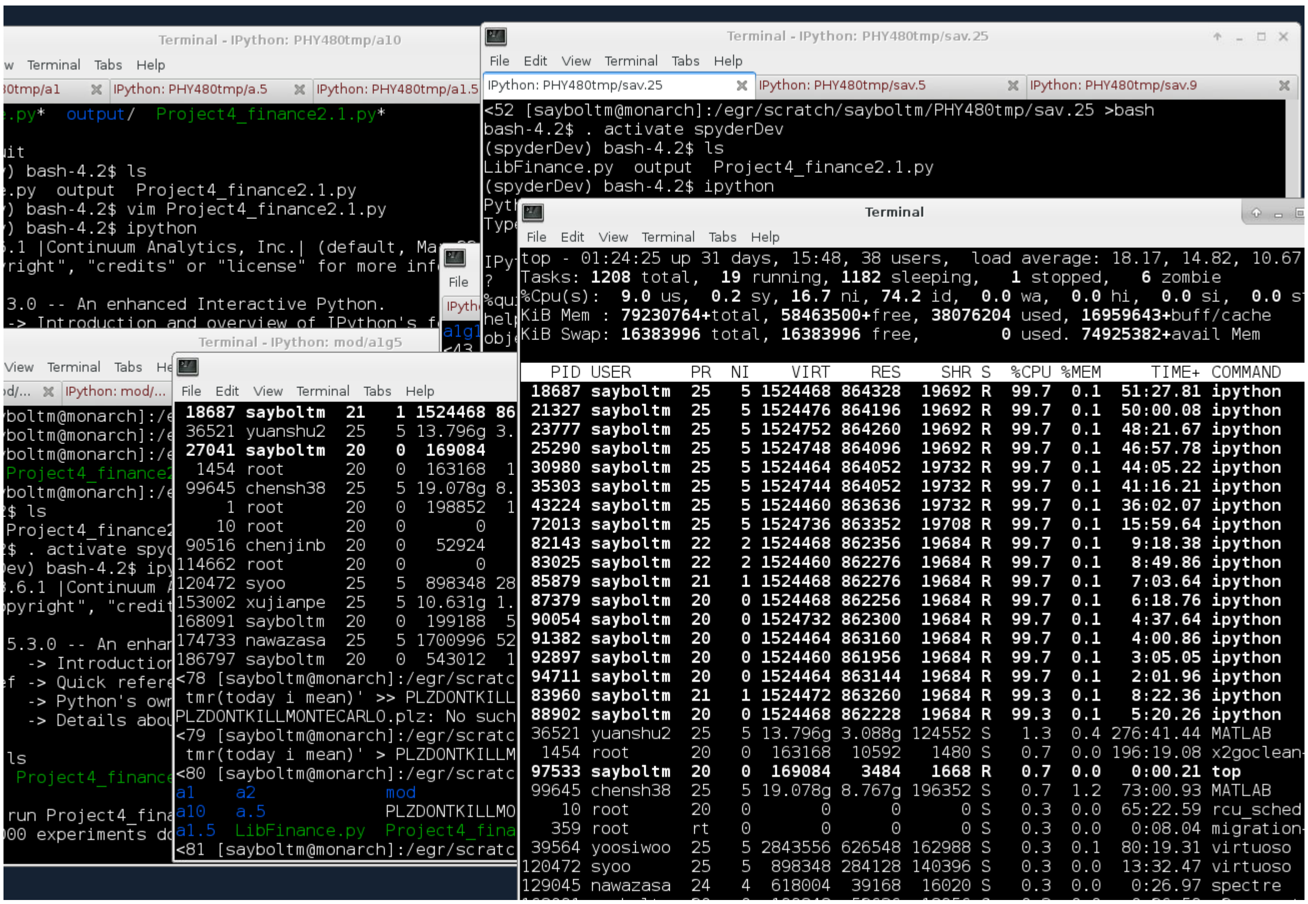
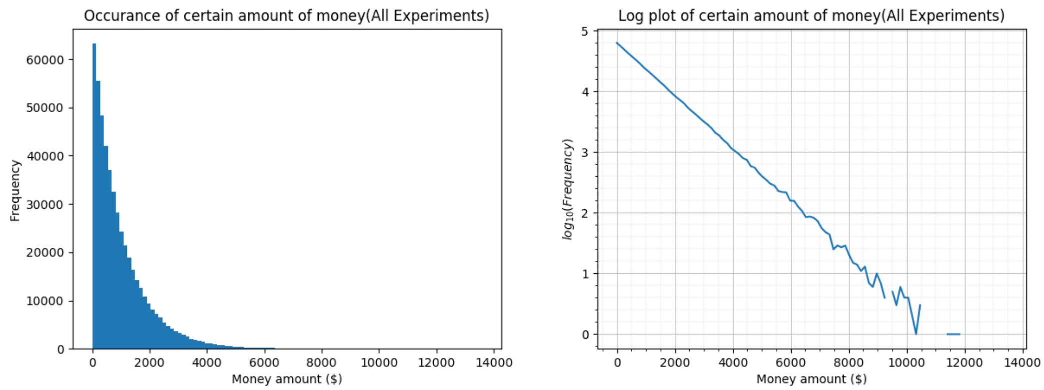


Figure 2: Taking over the egr compute servers



(a) Histogram: Gibbs distribution

(b) Log of the Gibbs distribution

Figure 3: The 'default' Gibbs distribution, N=500, `num_transactions`$=10^7$, `num_experiments`$=10^2$

based on wealth or previous transaction. There are many agents with little money, and very few with a lot of money.

Taking the log of this function results in a relatively straight line, so long as enough interactions take place. Despite the high number of transactions, the lower number of experiments results in poor convergence despite taking just as many flops as if `num_transactions`$=10^6$ and `num_experiments`$=10^3$.

4.2 Savings

To add realism or study a hypothetical economic systme, agents can save a fraction of their money, λ, in a transaction. While equation (1) for conservation of money still holds, the new money of the agents is modeled as follows

$$m_i' = \lambda m_i + \epsilon(1-\lambda)(m_i + m_j),$$

and

$$m_j' = \lambda m_j + (1-\epsilon)(1-\lambda)(m_i + m_j),$$

When agents save money, a strong 'middle class' is born. The more they save, the stronger this gets, and less agents have little to no money to deal with. The distribution goes from resembling a Gibbs distribution, to almost a perfect Gaussian. Results for saving a quarter, half and 90% of their money is shown by figures

wtf 4a, 4b, and 4c respectively.

Note the importance of having enough MC cycles to get an accurate representation of the correct distribution. Figure 4d shows the last MC cycle in this data set alone without 'averaging' them all together; the data is much more sporadic.

4.3 Similar Wealth

Agents can also be tailored to be more likely to interact with other agents of similar wealth, or geographic proximity, regardless of how much they save. This seems to help the lower and mid-income agents, and if the log is taken, somewhat resembles the plots in Goswami and Sen[3]. It seems there may be a bug since increasing α does not have as much effect on the curveature as expected.

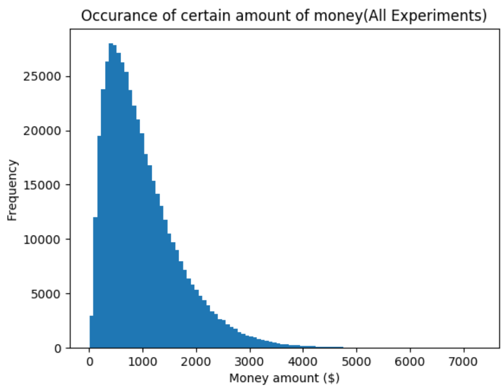
Taking it to the max to really see the influence, there is a difference with α=10, but it is not as much change as expected.

The figures are enlarged to show the x-axis scale, which does change considerably. The effect of larger α causing the agents to stick together results in a lower upper bound for maximum money.

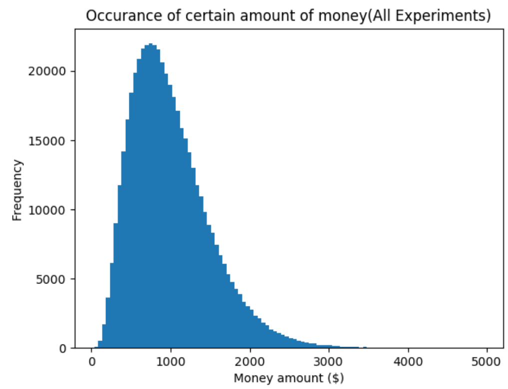
4.4 Adding Previous Interactions

People may have a tendency to want to continue to do business with previous clients, assuming it was a good transaction. Multiplying the probability of an interaction based on similar wealth, with an additional component containing a comparison to the maximum transactions between any agents, a confidence factor can be added in, increasing the likelhood of a transaction occuring the more previous transactions some agents have with each other.

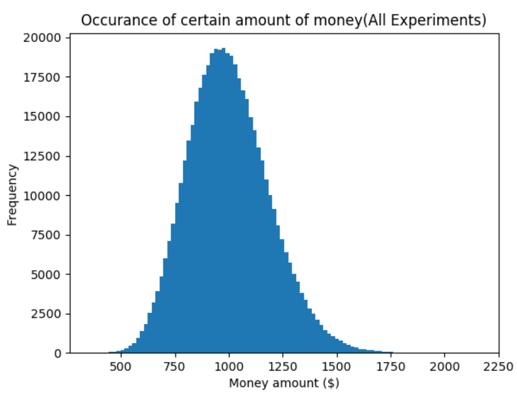
Looks like a bug where the probabilities are not scaled to each other properly. You see one effect clearly overtake the other. insert combined log and non log plots not sure why stick going up in the air. seen with default too (on an individual run). Guess is lack of interaction or low money exchange. Default averaged looks fine. Maybe need to establish criterion examining the variance and terminating then. interaction by chance or something.



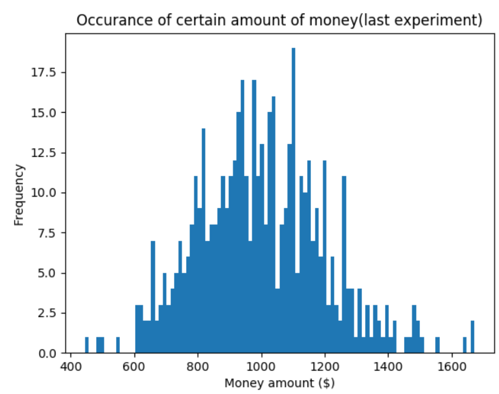
(a) λ=0.25, N=500



(b) λ=0.5, N=500



(c) λ=0.9, N=500



(d) λ=0.9, N=500, last MC cycle only

Figure 4: Distribution of wealth for `num_transactions`$=10^5$, `num_experiments`$=10^3$

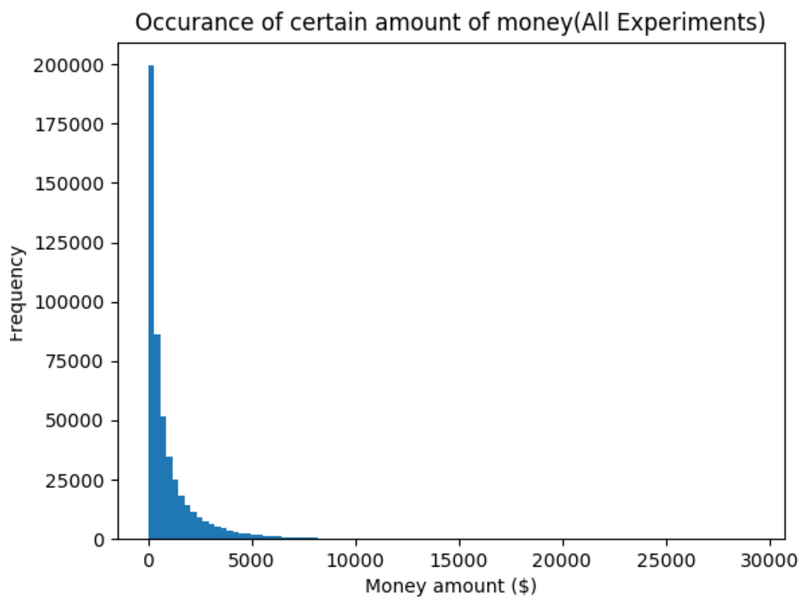


Figure 5: Distribution of wealth for α=0.5

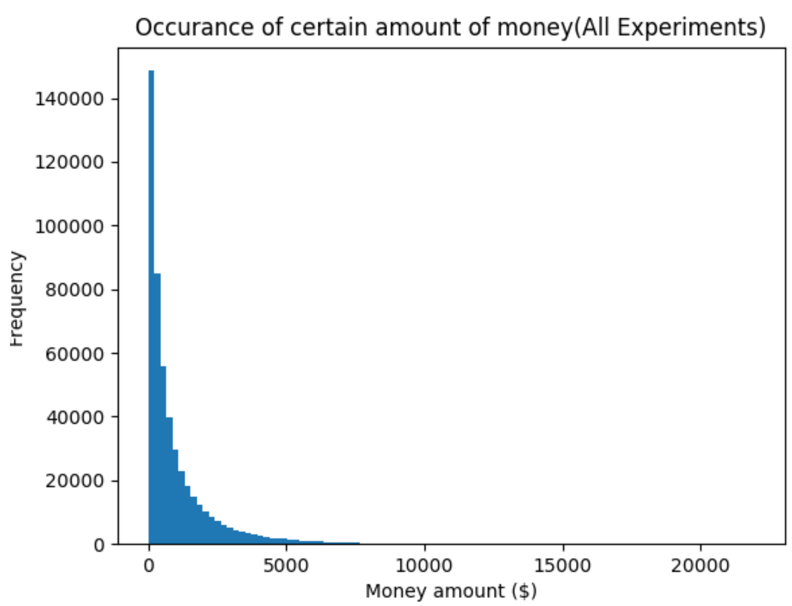


Figure 6: Distribution of wealth for α=1.0

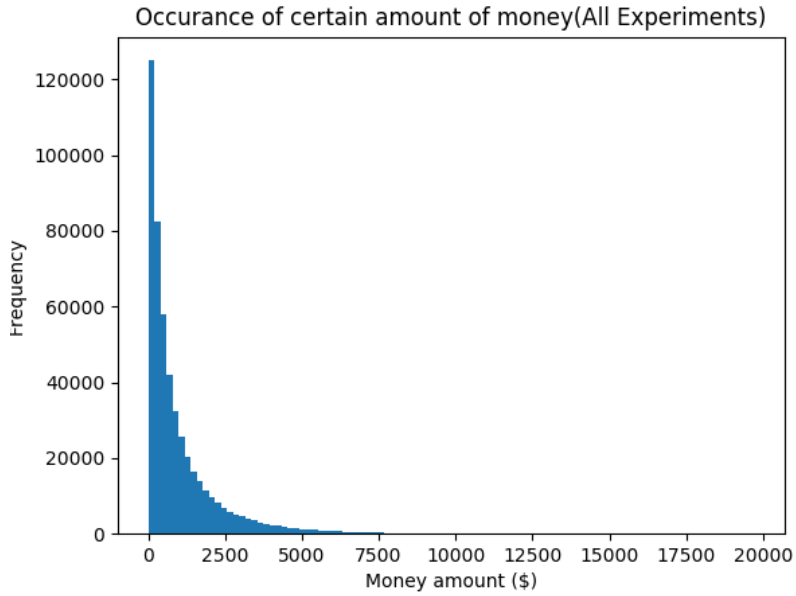


Figure 7: Distribution of wealth for α=1.5

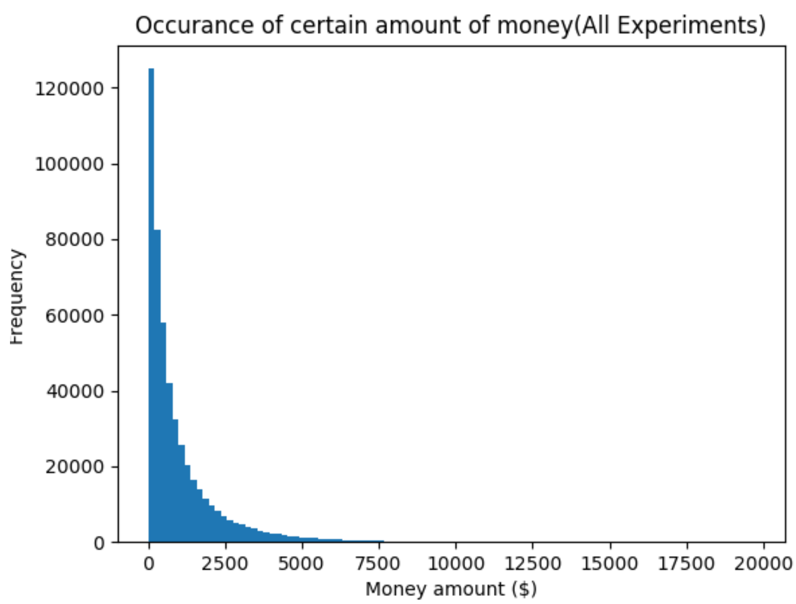


Figure 8: Distribution of wealth for α=2.0

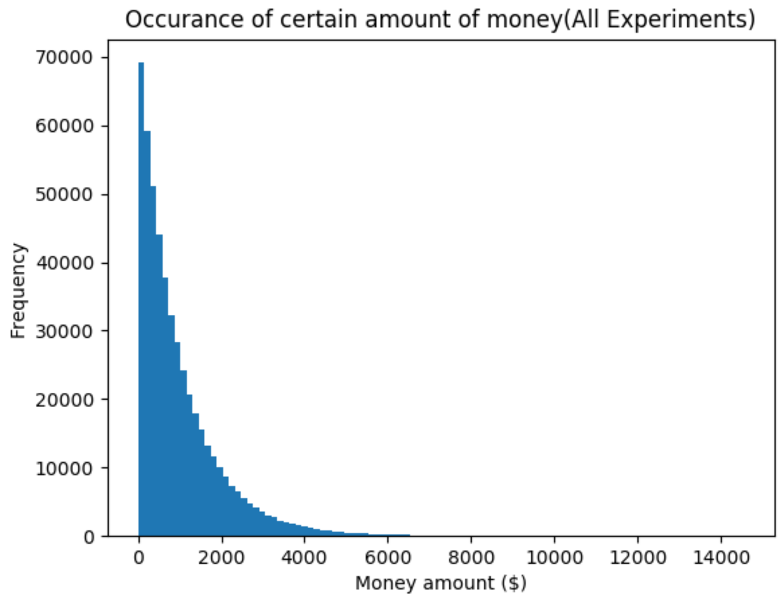


Figure 9: Distribution of wealth for α=10

5 Conclusions and Future Work

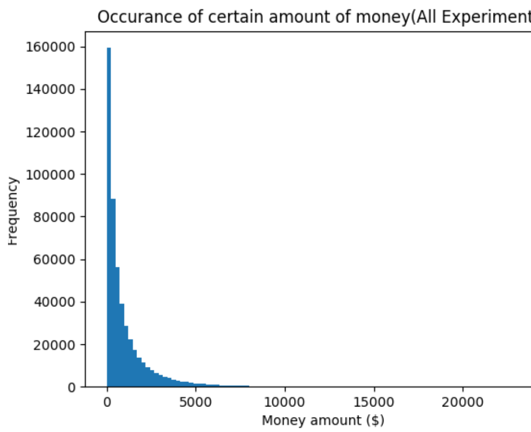
something

References

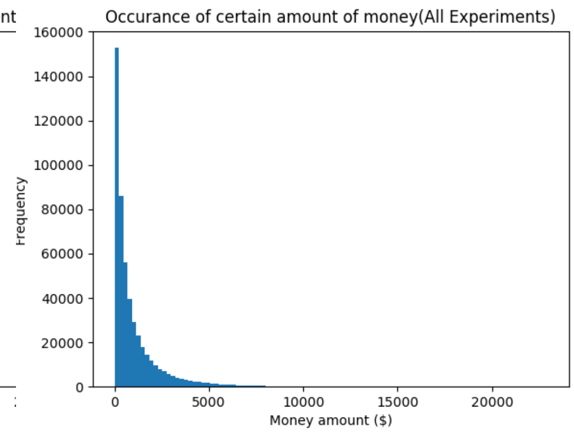
[1] V. Pareto, Cours d'economie politique, Lausanne, 1897.

[2] M. Patriarca, A. Chakraborti, K. Kaski, Physica A **340**, 334 (2004).

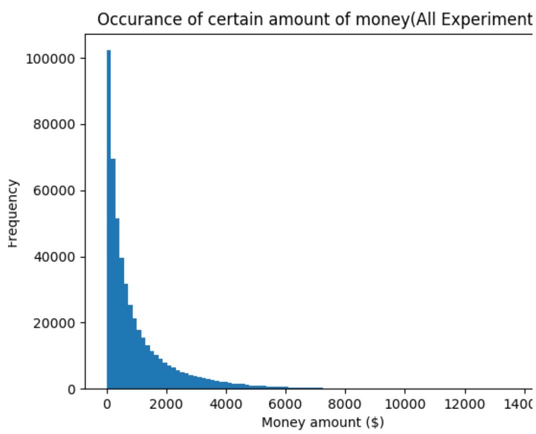
[3] S. Goswami and P. Sen, Physica A **415**, 514 (2014).



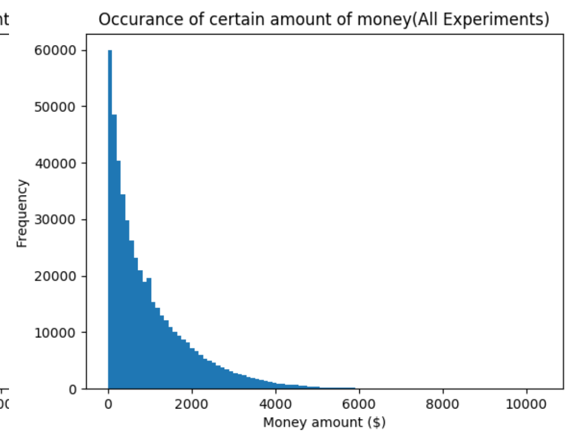
(a) α=1, γ=0



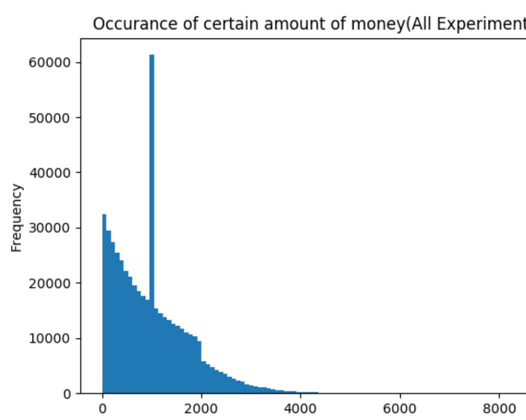
(b) α=1, γ=1



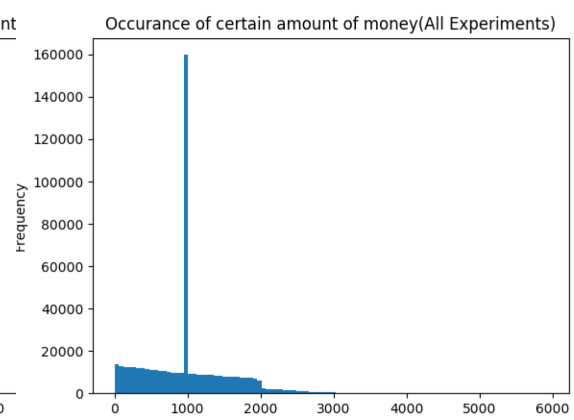
(c) α=1, γ=2



(d) α=1, γ=3

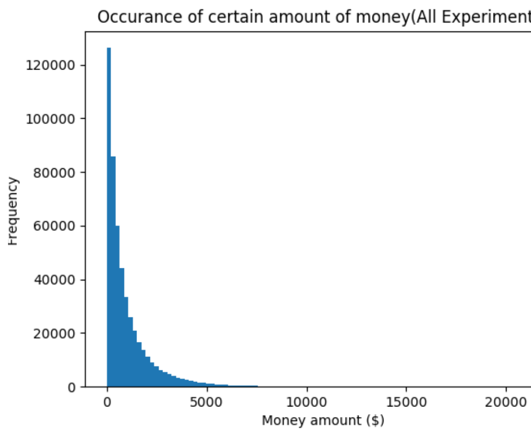


(e) α=1, γ=4

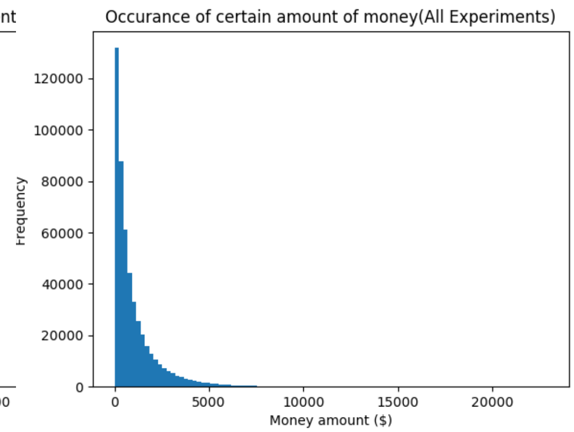


(f) α=1, γ=5

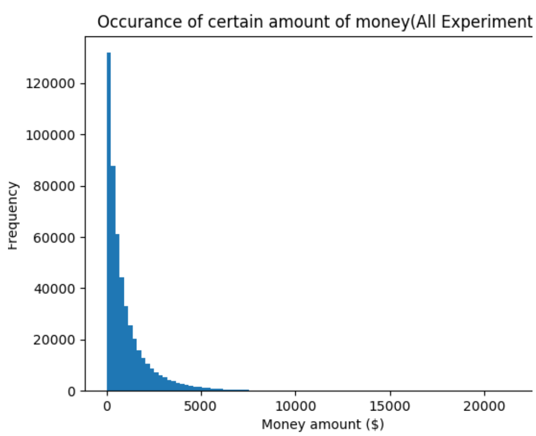
Figure 10: Distribution of wealth for α=1, N=500, `num_transactions`=10^6, `num_experiments`=10^3



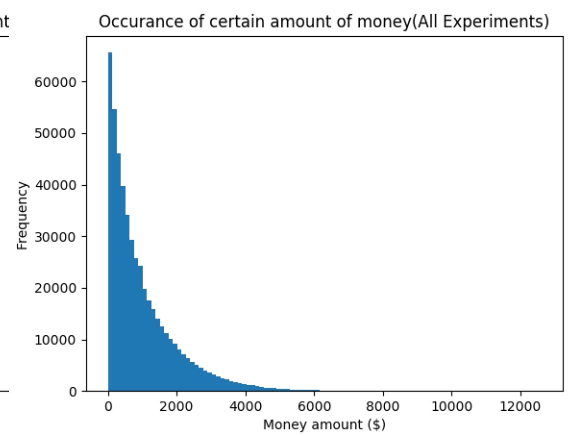
(a) α=2, γ=0



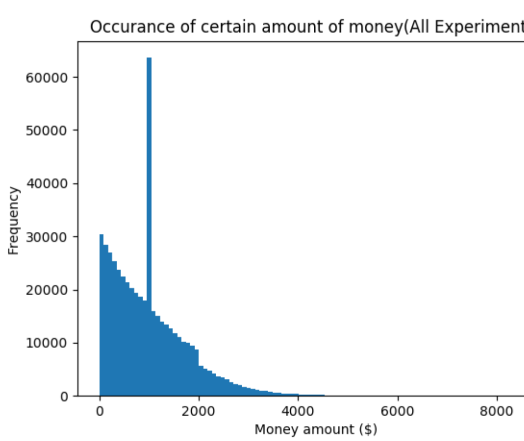
(b) α=2, γ=1



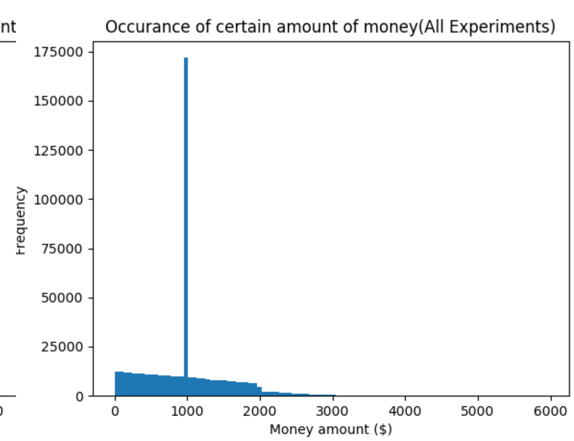
(c) α=2, γ=2



(d) α=2, γ=3



(e) α=2, γ=4



(f) α=2, γ=5

Figure 11: Distribution of wealth for α=2, N=500, `num_transactions`=10^6, `num_experiments`=10^3