PROJECT 5: SIMPLE STOCK MARKET MODEL
FYS4150 - COMPUTATIONAL PHYSICS

MARKUS LEIRA ASPRUSTEN   MAREN RASMUSSEN   METIN SAN

December 14, 2018

HTTPS://GITHUB.COM/METINSA/FYS4150/TREE/MASTER/PROJECT_5

ABSTRACT. This article uses a Monte Carlo model to simulate a simple stock market. It simulates transactions between random agents, and gradually introduces the following modifications: saving (λ), a tendency to that only agents of similar wealth preform transactions (α) and lastly a higher likelihood that agents that have preformed transactions in the past will do so again (γ).

The act of saving was found to increase the equality in the wealth distribution, while both a higher probability of performing transactions with agents of similar wealth and agents with a history of previous transactions was found to contribute to increased inequality in the wealth distribution.

1. INTRODUCTION

The primary objective of this study is to consider wealth distribution through a simple stock market model. We will combine concepts from statistical physics and macroeconomics which naturally places us in the field of econophysics. This is an interdisciplinary field where theories and methods originally developed by physicists are applied to solve economy related problems. These methods are often of stochastic nature such as the Monte Carlo method which will be our main algorithm. We will use a simple agent based model where we allow agents to carry out financial transactions with one another, similar to the previous marketing models by Goswami and Sen (2014)[1] and Patriarca et al. (2004)[3].

The report will consist of a theory section which briefly motivates the use of a simple agent model in addition to introducing the physics and mathematics which the model is based upon. We follow this up with a methods section which shows the functionality of the program and further discusses its features and implementation in detail. Thereafter we present the results of the study together with a discussion around them. The study is then finalized with a conclusion and future developments section where we discuss the weaknesses of the model and suggest improvements.

The main program modelling the stock market is written in `C++`, while the obtained results are analyzed through `Python`. The source code for the program is publicly available and can be found on our GitHub.

2. THEORY

2.1. **Pareto distribution.** Distribution of wealth has been studied for a long time. Already in 1987, Vilfredo Pareto proposed that the wealth distribution in a population could be modeled as a power law

$$w_m \propto m^{-1-\nu}, \tag{1}$$

with $\nu \in [1,2]$ and m being the money. The Pareto distribution results in the famous "80-20 rule" which states that 80% of the wealth in a society is held by only a mere 20% of its population. This a very successful model, especially in the higher ends of the distribution which has been shown by empirical studies.

2.2. **A simple model.** Most macroeconomic models which describe the operation of an economy usually model the economic income as a function of multiple variables. A good example is the popular theoretical IS-LM Keynesian model in which the income is the sum of consumer spending, investment, government spending and net export. In this study however, our main interest lies in the behaviour of the wealth distribution among financial agents. Such a wealth distribution can be achieved even in the absence of a regulating government. We will also assume that we are dealing with a closed economy, meaning that there are no financial net export. The resulting model from the above assumptions is a considerably simplified one where the income of one agent simply is the spending of another. A more mathematical formulation reads

$$Y_i = C_j, \tag{2}$$

where Y_i is the income of agent i and C_j is the consumption of agent j.

2.3. **Financial transactions.** In order to model a wealth distribution, we need to simulate a large set of financial transaction between the agents. We assume that our model consists of N financial agents who exchange money in pairs (i, j). We further assume that all agents start off with the same amount of money $m_0 > 0$. At a given time in our simulation, a transaction will take place between two arbitrarily chosen agents i and j. Agent i will have money m_i prior to the transaction and m'_i post transaction. Similarly for agent j, we have $m_j \to m'_j$. During such a transaction the total money will be conserved such that

$$m_i + m_j = m'_i + m'_j. \tag{3}$$

For a given transaction, the amount of money transferred is determined by a random reassignment factor ϵ, such that

$$m'_i = \epsilon(m_i + m_j), \tag{4}$$

$$m'_j = (1-\epsilon)(m_i + m_j), \tag{5}$$

where we have assumed that agent i is the one obtaining money. Another way to formulate equations (4) and (5) is

$$m'_i = m_i + \Delta m, \tag{6}$$

$$m'_j = m_j - \Delta m. \tag{7}$$

These sets of equations can the be solved for Δm, resulting in

$$\Delta m = \epsilon m_j - (1-\epsilon) m_i. \tag{8}$$

A consequence of the conservation law in (3) is that the system will relax toward an equilibrium given by a Gibbs distribution

$$w_m = \beta e^{-\beta m}, \tag{9}$$

where $\beta = \langle m \rangle^{-1}$ is the inverse average money and $\langle m \rangle = \sum_i m_i / N = m_0$. Such a distribution implies that after the system reaches equilibrium most of the agents will be left with a lower wealth compared to what they had to begin with, while the number of rich agents exponentially decrease. Another property of the Gibbs distribution is that its logarithm produces a linear equation

$$\ln w_n = \ln \beta - \beta m. \tag{10}$$

2.4. **Saving criterion.** We can add to the complexity of our model by introducing a saving criterion. In a more realistic scenario, it would be highly unlikely that agents would be willing to trade away all of their money during a trade. We will therefore introduce the saving fraction λ. This allows the agents to save a fraction λ of their wealth before they perform a transaction. We still require the conservation law in (3) to hold, however, the money traded during a transaction is now $(1-\lambda)(m_i+m_j)$. The post transaction wealth of agents i and j is then given by

$$m_i' = \lambda m_i + \epsilon(1-\lambda)(m_i+m_j), \tag{11}$$

$$m_j' = \lambda m_j + (1-\epsilon)(1-\lambda)(m_i+m_j). \tag{12}$$

We can then find an expression for the change in money Δm similar to what we did for the case without the saving. The change in money when a fraction of the wealth is saved is then

$$\Delta m = (1-\lambda)(\epsilon m_j - (1-\epsilon)m_i). \tag{13}$$

The introduction of a saving fraction affect the equilibrium distribution of our model, and we no longer expect to observe a Gibbs distribution [3] (unless we set $\lambda = 0$). Instead we expect to see a more complex curve which depend on the saving fraction. A parametrization can be extracted by fitting the results of numerical simulations. This is done by Patriarca and collaborators [3], where they introduce the reduced variable

$$x = \frac{m}{\langle m \rangle}, \tag{14}$$

which is the agents money normalized with the average money. They also introduce the parameter

$$n(\lambda) = 1 + \frac{3\lambda}{1-\lambda}. \tag{15}$$

The wealth distribution is then given by the well-fitted function

$$P_n(x) = a_n x^{n-1} e^{-nx}, \tag{16}$$

where the prefactor a_n can be shown to be

$$a_n = \frac{n^n}{\Gamma(n)}, \tag{17}$$

where $\Gamma(n)$ is the Gamma function. This parametrization is valid for any λ.

2.5. **Likelihood.** It is known that agents has certain preferences for potential trading partners. These preferences are often based on social class, economic position, connections and other factors. We will therefore introduce a likelihood p_{ij} to our model which will govern the probability of a transaction taking place between two agents. We will consider two likelihood models which are based on the work of Goswami and Sen [1].

2.5.1. *Economic status.* We assume that an essential factor is the similarity in wealth between the agents. In other words, the two agents should have similar economic status as it is unlikely that a rich agent is to perform a financial transaction with a poor agent. This is taken into account by the following likelihood function

$$p_{ij} \propto |m_i - m_j|^{-\alpha}, \tag{18}$$

where the parameter $\alpha > 0$ governs the strength of likelihood.

2.5.2. *Networking.* The likelihood can further be complicated by considering the transactions history of the individual agent. Agents will usually develop connections and networks with other agent through out their careers and are therefore more likely to perform financial transactions within these networks. We can therefore modify (18) to read

$$p_{ij} \propto |m_i - m_j|^{-\alpha}(c_{ij} + 1)^{\gamma}, \tag{19}$$

where c_{ij} represents the number of previous interactions that have taken place between agents i and j. This additional term account for an increase in likelihood if the two agents have traded before. We note that setting $\alpha = 0$ and $\gamma = 0$ represents a model where all agents are allowed to trade irrespective of social class and savings.

3. Algorithm & Implementation

We have chosen to go with an object oriented approach when creating the program. This is strictly not necessary as the code required to simulate the model is fairly brief and simple. However, object orientation allow us to further improve upon the model in the future if desired. It also results in a systematic code with high clarity. The complete program consist of 3 c++ files: `main.cpp`, `functions.cpp` and `functions.h`. The model consists mainly of the class `StockMarketModel` which again consists of the 3 subfunctions: `Trade`, `Simulate` and `DumpToFile`. We will now briefly outline the purpose of the first two functions.

3.1. The `Trade()` function.

This is the core function of the class. Its functionality is to perform a given number of transactions between the agents. This is done through a simple Monte Carlo algorithm which begins by picking two arbitrary agents. What follows is a logical operator which acts as the Monte Carlo acceptance rate based on the likelihood. The acceptance criteria is of importance to our model as it dictates the resulting distribution. It is therefore listed below in listing 1.

```
    while ((agent_i == agent_j) || (((pow(fabs(m_i - m_j), -alpha))
  * (pow(C_ij + 1, gamma))) < RNG_real(generator)))
    {
       // Picking two new agents
       agent_i = RNG_int(generator);
       agent_j = RNG_int(generator);

       // Extracting their wealth
       m_i = agents(agent_i);
       m_j = agents(agent_j);

       // Update the current number of transactions
       C_ij = C(agent_i, agent_j);

    }
```

Listing 1. Monte Carlo acceptance criteria.

The logical operator starts by checking if the two trade candidates are the same, that is $i = j$. If this condition is evaluated as **true** we immediately proceed by picking two new agents without calculating the likelihood. If two different agents are found, the program continues by computing the likelihood p_{ij}. The likelihood is then compared to a random number. If p_{ij} turns out to be smaller than this random number, the condition is evaluated as **true**, meaning that the agents are denied the trade. Two new agents are then chosen and the process is repeated.

The while loop only breaks if both Boolean expressions are evaluated as **false**, in which case the agents are allowed to trade.

What follows is then the transaction itself which is done by computing Δm and following equations (6) and (7). This only requires a few lines of code, as seen in the implementation in listing 2.

```
        // Finding a random monetary value exchanged during transaction
        double epsilon = RNG_real(generator);

        // Computing the traded money delta_m
        double delta_m = (1-lambda)*(epsilon*m_j - (1 - epsilon)*m_i);

        // Updating the new wealth of both agents
        agents(agent_i) += delta_m;
        agents(agent_j) -= delta_m;

        // Update the number of transactions
        C(agent_i, agent_j) += 1;
        C(agent_j, agent_i) += 1;
```

Listing 2. The transaction process.

3.2. **The `Simulate()` function.**

This function is responsible for performing a given number of Monte Carlo simulations making use of the `Trade()` function. For each simulation, the end wealth distribution of the agents are sorted and stored. After all the simulations are done, the wealth distributions are averaged and written to a file using the `DumpToFile` function. The implementation is seen below in listing 3

```
    // Performing a given number of simulations
    for (int i = 0; i < simulations; i++)
    {
        // Activating the Trade function
        Trade();

        // Sorting the agents array and adding it to the total
        total_average_agents += arma::sort(agents);
    }

    // Averaging over all simulations and saving to file
    total_average_agents /= simulations;
    DumpToFile();
```

Listing 3. Performing multiple simulations.

3.3. **Equilibrium estimate.**

Wealth distributions such as the one we are working with often end up in a steady state after a set number of transactions. The exact number of transactions required to reach this equilibrium is unknown an often varies from simulation to simulation. After the steady state is reached, further transactions between agents will often result in negligible changes to the distribution. It is therefore wise to implement a functionality which stops the simulation once a equilibrium situation has been reached. Such a functionality leads to significant improvement in terms of computational time as we might heavily reduce the total amount of transactions needed per simulation.

There are no single definitive way of making such an estimate, but most equilibrium criteria in Monte Carlo simulation are usually variance based. We have chosen a similar approach where we analyze the variance in the distribution at given transaction intervals in a simulation. We start by sectioning the total number of transaction in a single simulation into intervals of 10^4 transactions. After every such interval, the average variance over the interval is computed. This variance is then compared to the average variance of the previous interval. If the variance has changed by less than some desired percentage, we assume that equilibrium is reached and stop the current simulation. The implementation can be seen in listing 4.

```
      // Enters loop at the end of a transaction interval
      if (i % transaction_interval == 0)
      {
         // Averaging the variance over the interval
         averaged_variance = variance/transaction_interval;

         // Checking if variance changed by less than 0.5%
         if ((fabs(previous_averaged_variance - averaged_variance) /
   (previous_averaged_variance)) < 0.005)
         {
            // Equilibrium has been reached.
            break;
         }

         else
         {
            // The averaged variance is stored for later use
            previous_averaged_variance = averaged_variance;
         }

         // Reseting the variance
         variance = 0;
```

Listing 4. Estimating the equilibrium wealth distribution .

We found that a variance difference of 0.5% was a good pick as it most likely guarantees that the distribution is in equilibrium.

3.4. Running the simulation.

The execution of the program is done through the `main.cpp` file. It takes in the following argument: A Filename, saving fraction λ, parameter α, parameter γ, number of simulations, number of transactions, and the initial and average wealth m_0. All arguments, except the filename, are optional as standard values are provided by the program.

3.5. Parallelization.

To save time in calculations, the program can easily be parallelized, as the results we are interested in are the averages of several discrete simulations. Because of this, we simply have to distribute the calculations to several threads. There are several ways to do this. In our case we chose to use OpenMP as it does not require a lot of change from the serial code written. The goal is the to distribute all simulations as equal as possible to all available threads.

```
#pragma omp for // Automatically distributing the indices of the
    for-loop to threads
  for (int i = 0; i < simulations; i++)
   {
    // Activating the Trade function
    Trade(agents, C); // Sending agents and C as pointers
    // Sorting the agents array and adding it to the total
    total_average_agents_per_thread.col(thread_num) +=
    arma::sort(agents);
   }
```

```
#pragma omp barrier // Barrier to make sure all threads are finished
    before average is calculated
#pragma omp master // Only run by master thread
  {
   // Calculating average from all threads
   total_average_agents = sum(total_average_agents_per_thread , 1);
   // Saving the final results by dumping them to a file
   total_average_agents /= simulations;
   DumpToFile();
```

Listing 5. Running the code from listing 3 in parallel.

OpenMP has a simple way to do this, and in theory only requires two–three extra lines of code, as the module provides commands to distribute all values of a for-loop to all available threads automatically, as seen in listing 5. The problem also has the very convenient property of distributing nicely between the threads so that all threads are more or less utilized to the max at all times. This basically means that that the computational time drops approximately as t/n, where t is the computational time on one thread, and n is the number of threads available for parallelization. I.e. increasing the number of threads will drastically reduce the computational time used. This approximation only works with comparable CPUs. The parallelized version of the code can be found under the *Parallel*- branch on our GitHub.

3.6. **Data analysis.** To analyze the data from the simulations, we use some simple python codes. The results from the simulations are given as a average wealth for specific agents. We are interested in the distribution of wealth across the agents. This can be found by using the numpy function `np.histogram` with the `density` variable set to `True` to normalize the result. This function basically just sums up the number of values which are within certain bins and normalizes the result. It is then a simple matter to plot or fit the distribution with other functions. The implementation of this can be seen in several files in the `tools` folder on our GitHub.

4. Results and discussion

4.1. **Initial distribution.** Before adding any acceptance criteria or saving to the simulations, we ran a simulation with uniformly random transactions. The results of this can be seen in figure 1, where we have plotted the average wealth distribution after 10^4 simulations of 10^7 transactions with $m_0 = 100$. This gives us more or less a straight line before hitting zero at about 6-7 times m_0 when we plot it with a logarithmic y-axis. This can be fitted to $w(m) = 0.01e^{-0.01m}$ which can be though of as a Gibbs measure (Boltzmann distribution in physics), as a Gibbs distribution is on the form of $f(x) \propto e^{-ax}$ as seen in equation 9.

4.2. **Saving criterion.** Figure 2 shows how the wealth is distributed after equilibrium for different saving fractions λ, using 1000 interacting agents, 10^7 transactions and a total of 10^4 simulations. All the agents started with a wealth of $m_0 = 100$. The dots shows the numerical results, while the solid lines shows the fitting functions found by Patriarca and collaborators [3], defined in equation 16. We can clearly see that our results matches the results Patriarca and collaborators got. Note that Patriarca and collaborators have used percentage on the y-axis, while we have used decimal, and that we have used different initial wealths m_0.

We can also see that when the agents are not allowed to save, i.e. $\lambda = 0$, we obtain the Gibbs distribution as we also got in figure 1, where most of the agents end

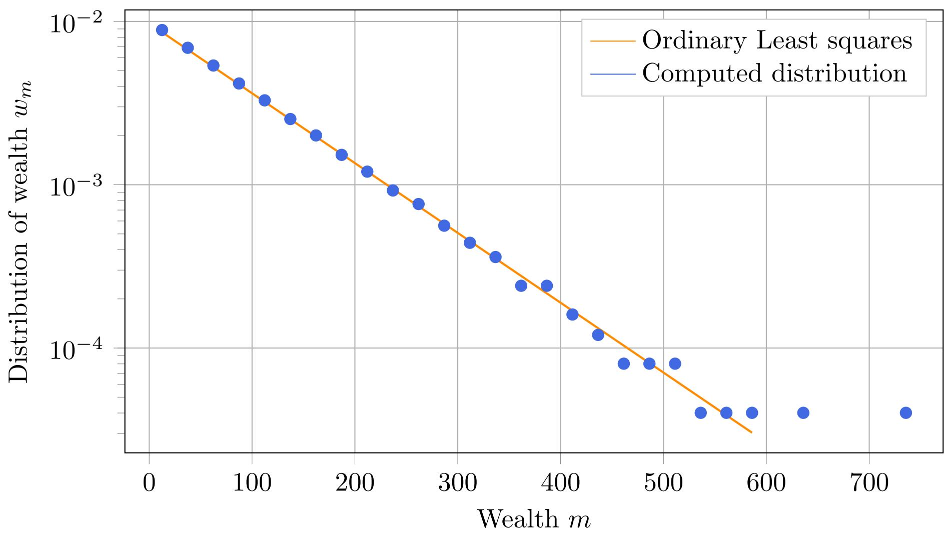

FIGURE 1. Wealth distribution after 10^4 simulations with 10^7 transactions as a semi-logarithmic plot, without savings or acceptance criteria. The orange line has been fitted to the exponential function $w(m) = 0.01e^{-0.01m}$.

up with very little money. If we let the agents save 25 % of their wealth, we see that the distribution changes. Now most of the agents end up with a wealth of $m = 50$, which corresponds to half of their initial wealth. When the saving fraction increases further, we can see that the peak is approaching the initial wealth of $m = m_0 = 1000$. This is obviously because the agents now only trade with 10 % of their money, and none of them will end up with a very small or very large amount of money. The distribution also looks more and more like a Gaussian distribution as the saving fraction increases.

Figure 3 shows the same, but now using logarithmic axis. Again we can see a plot matching well with what Patriarca and collaborators have gotten.

In section 2.1 we mentioned that the high end of wealth distributions tend to follow the Pareto distribution. Figure 4 shows an attempt to fit our data to the Pareto power law given in equation 1. We can see that when no saving is allowed, figure 4a, we get a quite good match with the Pareto power law using $\nu = 2$ giving $w_m \propto m^{-3}$. For comparison the Pareto power law for $\nu = 1.5$ giving $w_m \propto m^{-2.5}$ is also plotted, but we can see that this does not fit with the same precision as it decreases to slow. With a saving fraction of $\lambda = 0.5$ on the other hand, as shown in figure 4b, the data can not be fitted with a Pareto power law, as it is not steep enough for $\nu = 2$. But by instead allowing an even higher exponent and let $w_m \propto m^{-4}$ we can fit the data quite well.

4.3. **Nearest neighbour interactions.**

After implementing nearest neighbour transactions, where agents with similar wealth are more likely to carry out a transaction, we got the distributions outlined in figures 5 and 6 after equilibrium was reached. The simulations was done with 5000 agents to get a higher resolution in the tail. The results for 500 and 1000 agents is approximately identical to the figures provided, with the exception of the back end of the tails.

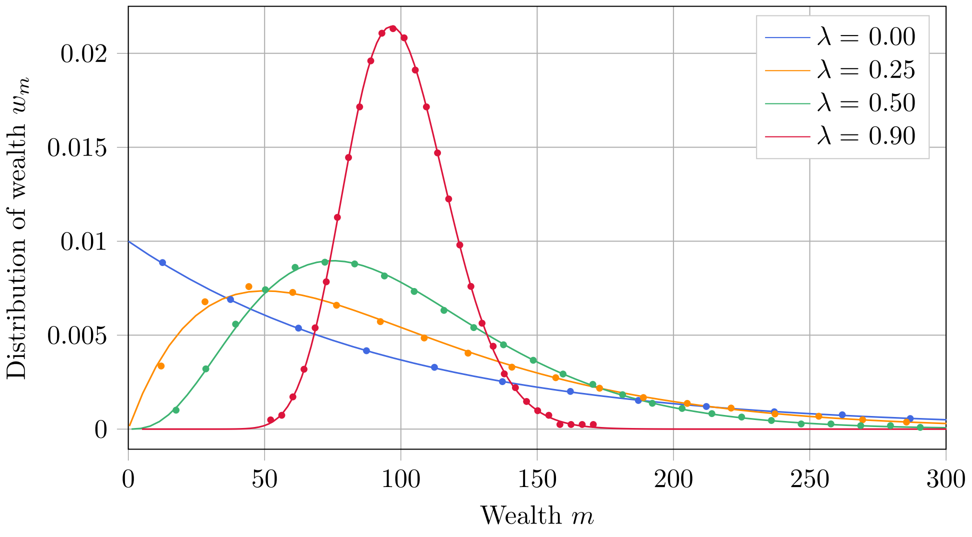

Figure 2. The equilibrium distribution of wealth for different saving fractions λ using 1000 interacting agents with initial wealth of $m_0 = 100$, 10^7 transactions and a total of 10^4 simulations. The dots shows the numerical results, while the solid lines shows the fitting functions found by Patriarca and collaborators [3], defined in equation 16.

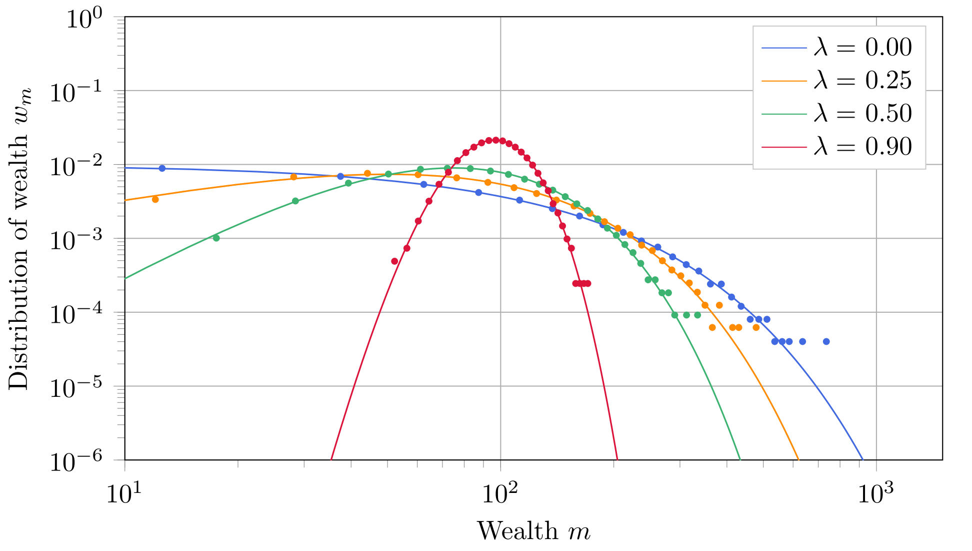

Figure 3. The equilibrium distribution of wealth for different saving fractions λ using 1000 interacting agents with initial wealth of $m_0 = 100$, 10^7 transactions and a total of 10^4 simulations, plotted with logarithmic axis. The dots shows the numerical results, while the solid lines shows the fitting functions found by Patriarca and collaborators [3], defined in equation 16.

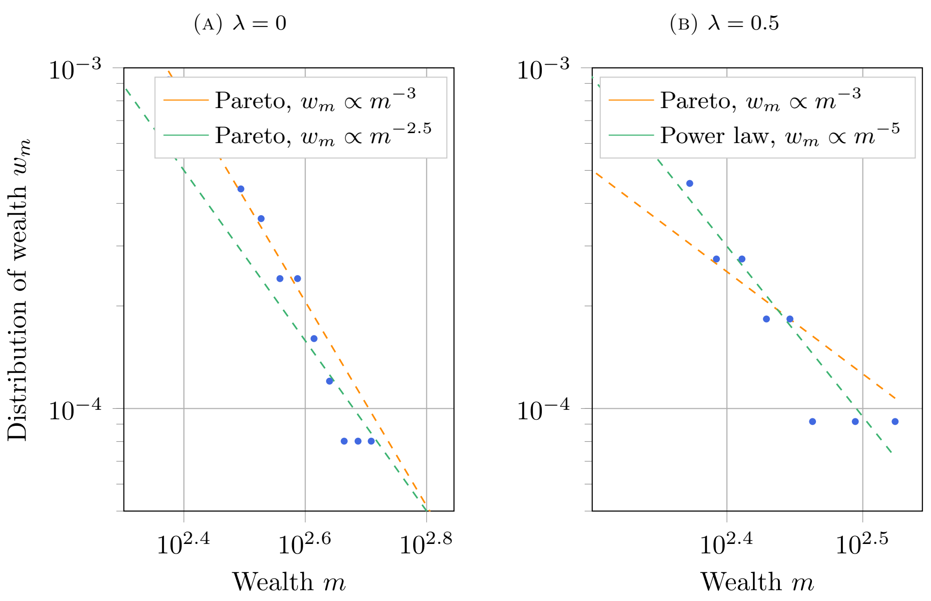

FIGURE 4. The high end of the distribution fitted with the Pareto power law given in equation 1 for different values of ν. In figure 4a the data without any saving is given by the blue dots, the orange line shows a Pareto power law given by $w_m = 1.3 \cdot 10^4 \cdot m^{-3}$ and the green line shows a Pareto power law given by $w_m = 5 \cdot 10^2 \cdot m^{-2.5}$. In figure 4b the data is again given by the blue dots, but now using a saving fraction of $\lambda = 0.5$. The orange line shows a Pareto distribution given by $w_m = 4 \cdot 10^3 \cdot m^{-3}$, while the green line shows a power law given by $w_m = 3 \cdot 10^8 \cdot m^{-5}$.

As seen in figure 5a, a higher preference for agents to transact only with agents with similar wealth increases the amount of agents with very low wealth, and may induce more inequality in the system. Introducing saving, as seen in figure 6b, affects the distribution only to a certain degree. With a lower α (i.e. higher chance of someone with different wealth to preform a transaction), we see a reduction in the number of agents with very low wealth, and a reduction in the wealth of the richest agents. A higher preference to only preform transactions with agents of similar wealth seems to negate this effect, as the distributions for both $\lambda = 0$ and $\lambda = 0.3$ are more or less identical for $\alpha \gg 1$.

Our findings are very similar to the results from Patriarca et al.[3], and figure 1 in their article is more or less identical with our figure 5a showing the same distributions.

We can see in figure 6a that a Pareto distribution extracts the essential properties of the tail of the distribution by following the main changes in the distributions. This makes a Pareto distribution look like a good approximation for the tail of the wealth distributions. The raw data is, however, very variable and it is therefore a little difficult to judge the accuracy of the fitted function correctly. This seems to be the case with saving as well, as seen in figure 6b.

4.4. **Former transactions.** Figure 7 shows the equilibrium distribution of wealth when we have also included that there is a higher likelihood for interactions between

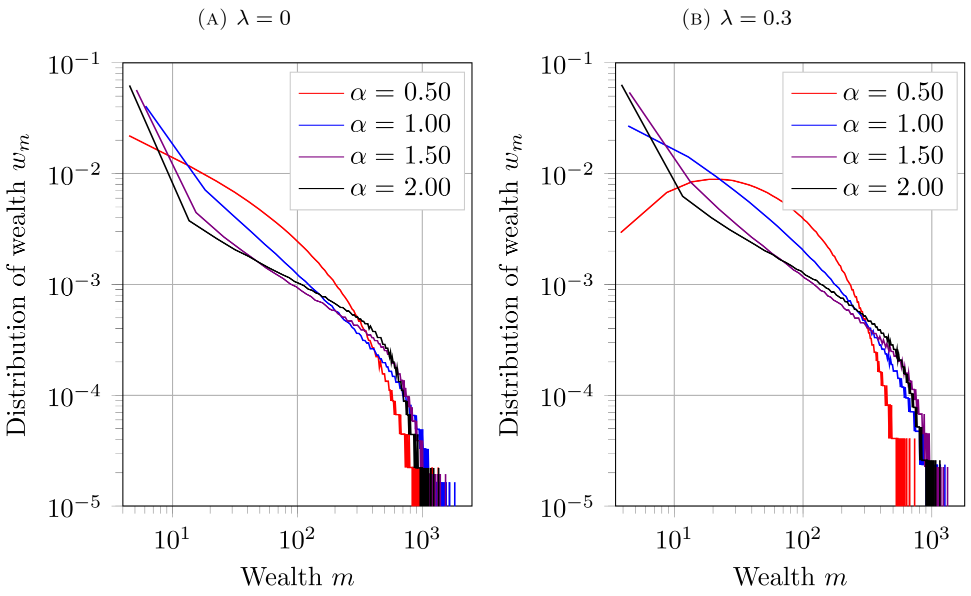

FIGURE 5. The equilibrium distribution of wealth for different α-exponents using 5000 interacting agents with initial wealth of $m_0 = 100$, 10^7 transactions and a total of 10^4 simulations. In figure 5a $\lambda = 0$, and in figure 5b $\lambda = 0.3$, i.e. 30 % saving.

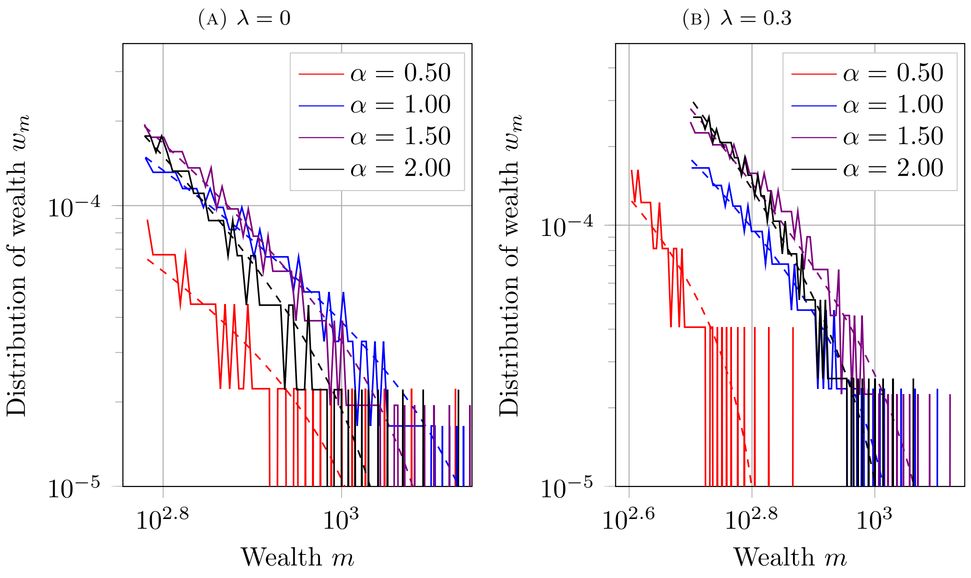

FIGURE 6. The tail of the distributions seen in figure 5, with a Pareto distribution fitted their respective data as a dotted line with the same colour. In figure 6a $\lambda = 0$, and in figure 6b $\lambda = 0.3$, i.e. 30 % saving.

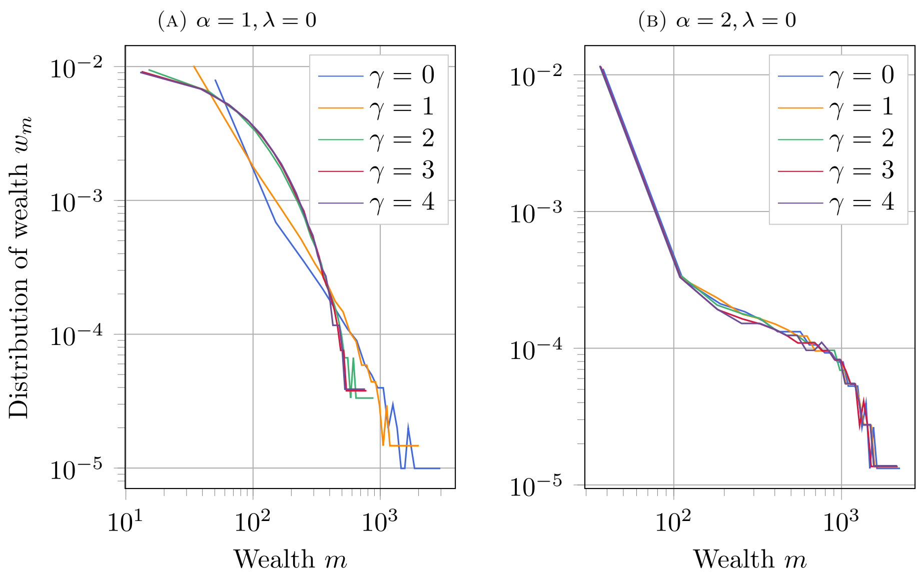

FIGURE 7. The equilibrium distribution of wealth for different γ-exponents using 1000 interacting agents with initial wealth of $m_0 = 100$, 10^7 transactions and a total of 10^4 simulations. In figure 7a $\alpha = 1$, and in figure 7b $\alpha = 2$. In both figures no saving is allowed, i.e. $\lambda = 0$.

agents that have performed similar transactions earlier. Both of the subfigures shows the distributions with different γ-exponents. It is also included that the agents have to be financially close to interact. 1000 interacting agents with initial wealth of $m_0 = 100$, 10^7 transactions and a total of 10^4 simulations was used as initial conditions. The agents were not allowed to save, i.e. $\lambda = 0$. In figure 7a $\alpha = 1$, and in figure 7b $\alpha = 2$, meaning that they have different strengths on the criterion that agents have to be financially close to interact.

In figure 7a we can see that lower γ-exponents gives more linear distributions, while higher γ-exponents gives more curved distributions. This is because a higher γ-exponent strengthens the criterion saying that agents that have interacted is more likely to interact again.

In figure 7b it seems like an α-value of 2 makes the likelihood for interactions between financially close agents dominate, so agents that are not financially close will almost not interact at all. This means that the richest agents will only interact with the other rich agents, while the poorer agents will only interact with other poorer agents. Because of this, the effect from a higher probability of performing a transaction between agents with a history of former transactions, is negligible since a higher α already causes the same effect. This means that all γ-values results in similar distributions.

By comparing our result with the results Goswami and Sen got in their report [1], we can see that our results have some similarities, but they are still quite different. The curved part of out figure 7b, can for example not bee seen in their results. But because of lack of information about what parameters they have used running their simulations, we can not expect to get exactly equal answers.

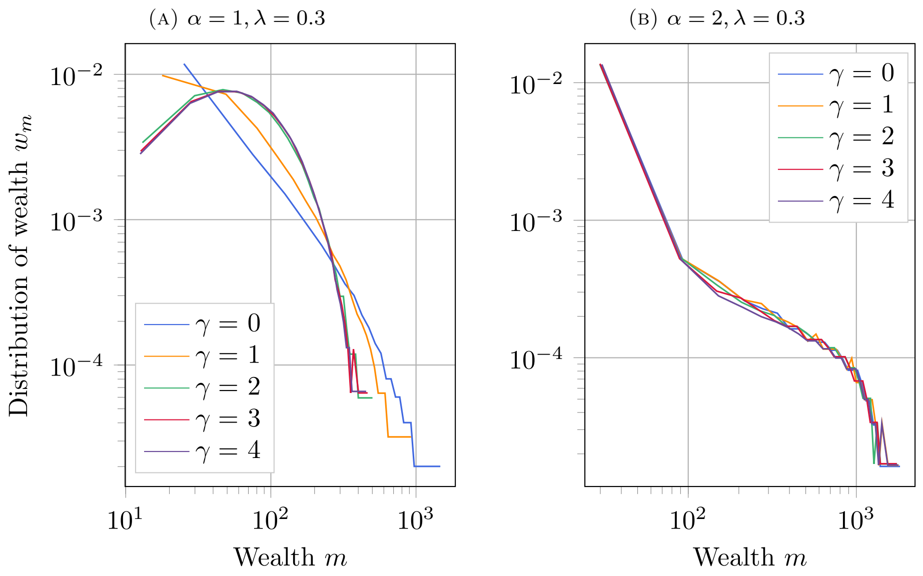

FIGURE 8. The equilibrium distribution of wealth for different γ-exponents using 1000 interacting agents with initial wealth of $m_0 = 100$, 10^7 transactions and a total of 10^4 simulations. In figure 8a $\alpha = 1$, and in figure 8b $\alpha = 2$. In both figures the agents are allowed to save 30 % of their wealth, i. e. $\lambda = 0.3$.

Figure 8 shows the same, but now allowing the agents to save 30 % of their wealth. By comparing figure 8a and figure 7a, we can see that by allowing the agents to save money, less agents end up with a very small amount of wealth in the equilibrium distribution. This is also something we discussed in section 4.2 and 4.3.

5. CONCLUSION

We found that the simple model without any criteria follows the Gibbs distribution, as seen in figure 1. This is exactly as expected from section 2.3, and indicates that our model works as expected.

The tendency to only preform transactions with agents with similar wealth may be a leading cause of unequal wealth distributions, as seen in figure 5 where a higher α denotes a higher tendency to only preform transactions with agents of similar wealth. Even though saving (λ) has a mitigating effect on this result for lower α, we found that at $\alpha \gg 1$ this no longer has an effect. In the same manner, allowing a higher probability for transactions between agents with a history of former transactions will negate the effects of saving and result in a distribution where there are a lot of agents with very low wealth, and a few agents with very high wealth, as seen in figures 7 and 8. This indicates that stimulating the market for transactions between agents regardless of wealth and transaction history may have a positive effect on wealth inequality.

Future work

As a future project, the validity of these results should be tested against observation from the real economy. This could highlight both negative and positive attributes of the current model, and also outline possible improvements to the model. One such improvement may be the inclusion of inflation, as total wealth is not a conserved quantity in reality.

The model also does not take purchasing power into account, e.g. even though the poorest agent may have a factor 10^{-3} less wealth than the richest agent, the poorest agent may well have enough to live a more than comfortable life. A model implementing purchasing power may yield interesting results that are more applicable to real life scenarios.

References

[1] Sanchari Goswami and Parongama Sen. Agent based models for wealth distribution with preference in interaction. *Physica A: Statistical Mechanics and its Applications*, 415(C):514–524, 2014.

[2] Morten Hjorth-Jensen. Overview of couse material: Computational physics. https://compphysics.github.io/ComputationalPhysics/doc/web/course, 2018. Accessed 2018.12.10.

[3] Marco Patriarca, Anirban Chakraborti, and Kimmo Kaski. Gibbs versus non-gibbs distributions in money dynamics. *Physica A: Statistical Mechanics and its Applications*, 340(1):334–339, 2004.