FINANCIAL ENGINEERING

A PRESENTATION ON PROJECT 4 AND A REFLECTION ON PHY480

OVERVIEW OF PRESENTATION

- Overview of Project 4 as Well as Why I Chose This Topic
- The Work We Did With Project 4
- Reflections on PHY480 In General

WHY DID I CHOOSE FINANCIAL ENGINEER?

WHY DID I CHOOSE FINANCIAL ENGINEERING?

- Future Career Interests
- It Served As A Nice Break From Typical Curriculum
- Not Computationally Challenging but A Nice Thought Exercise
- Group Dynamics

WHAT IS THE IDEA BEHIND PROJECT 4

- "Econophysics"
- Two People, i and j, Meet Under Certain Circumstances
- Whether Or Not They Exchange Money Depends on What We Looked At
- Random Number Generation

WHAT DID WE ACTUALLY DO?

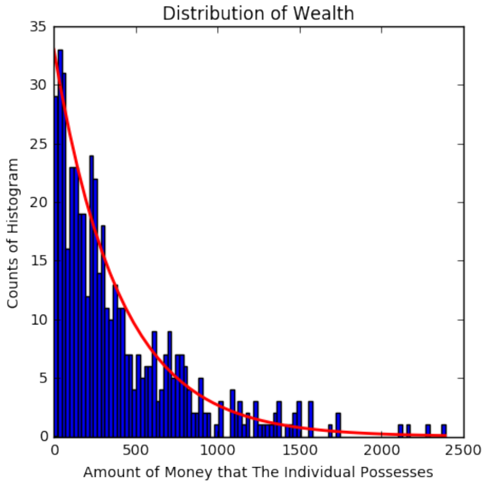
LET'S ACTUALLY GET IN TO PROJECT 4 DETAILS

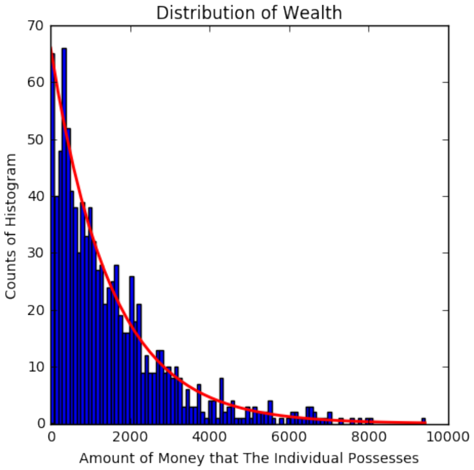
WHAT DID WE ACTUALLY DO?

- Model Basic Financial Interactions Between People
- Introduce Some ”Real World” Parameters
- Think

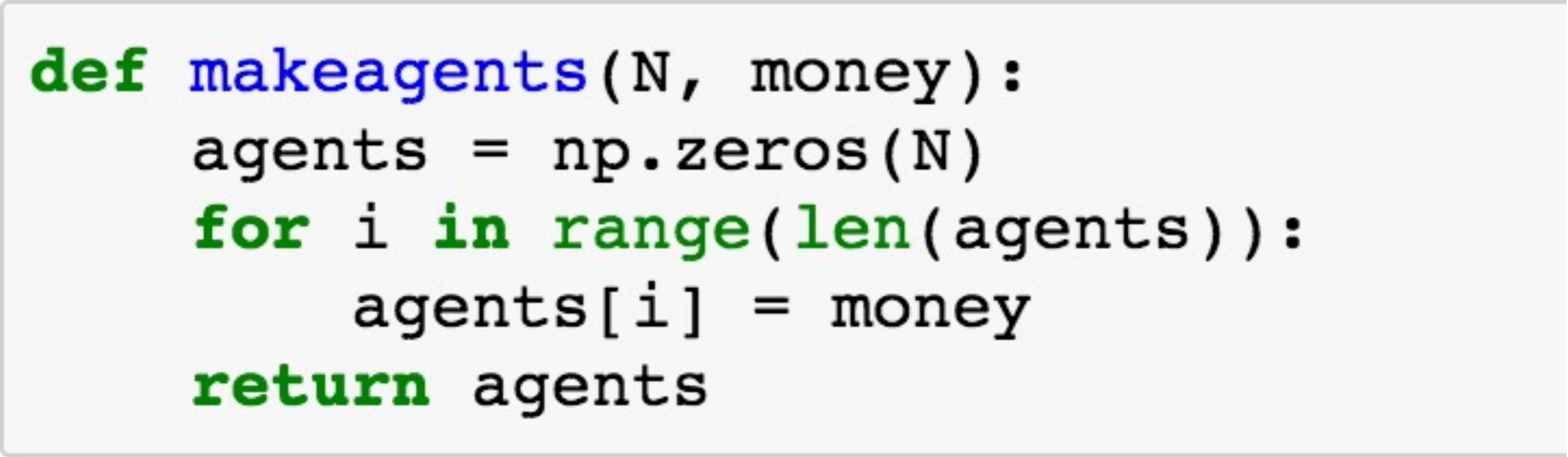
PART ONE: BASIC TRANSACTIONS

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



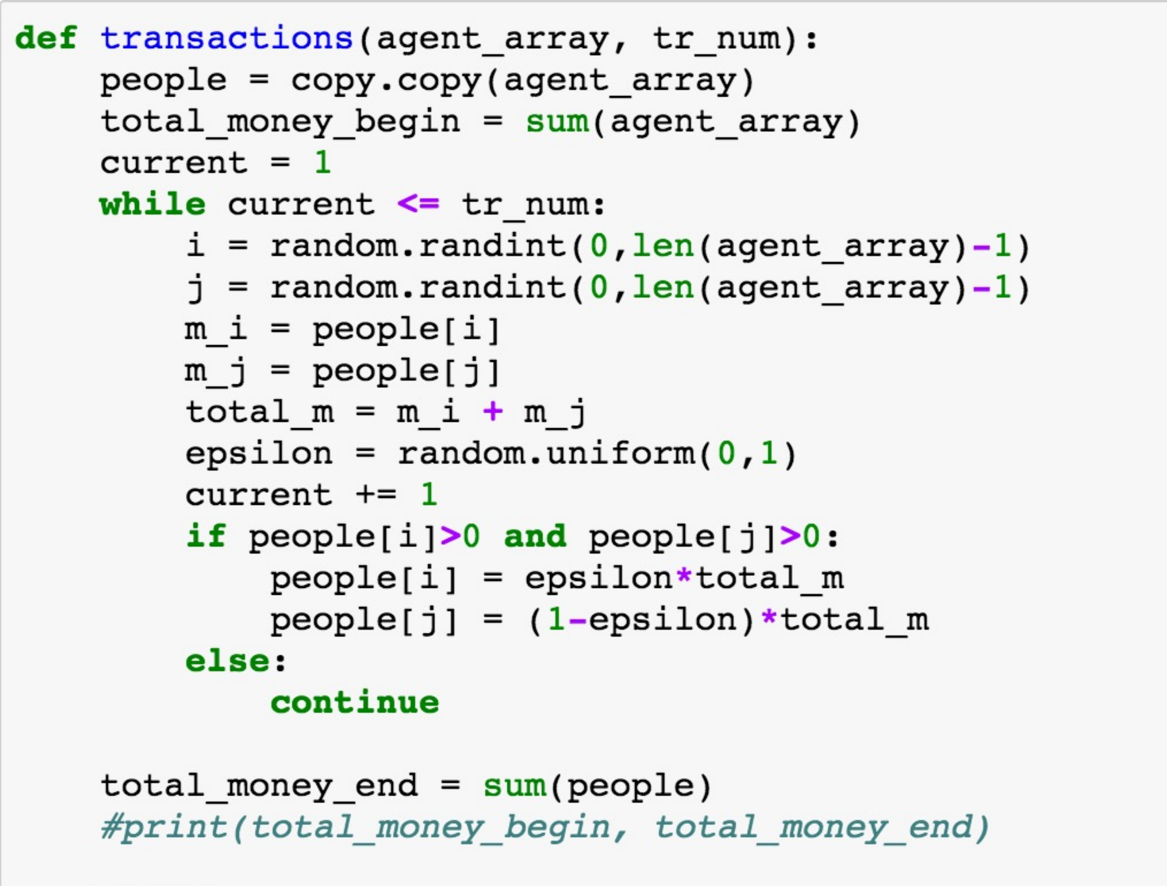


THE CODE



```
def makeagents(N, money):
    agents = np.zeros(N)
    for i in range(len(agents)):
        agents[i] = money
    return agents
```

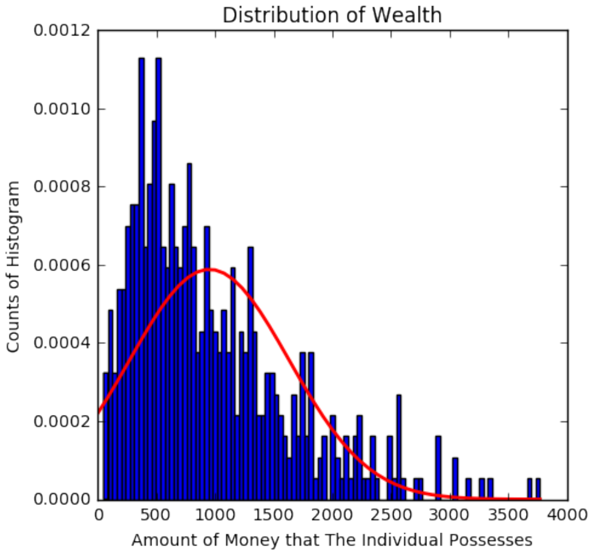
THE CODE



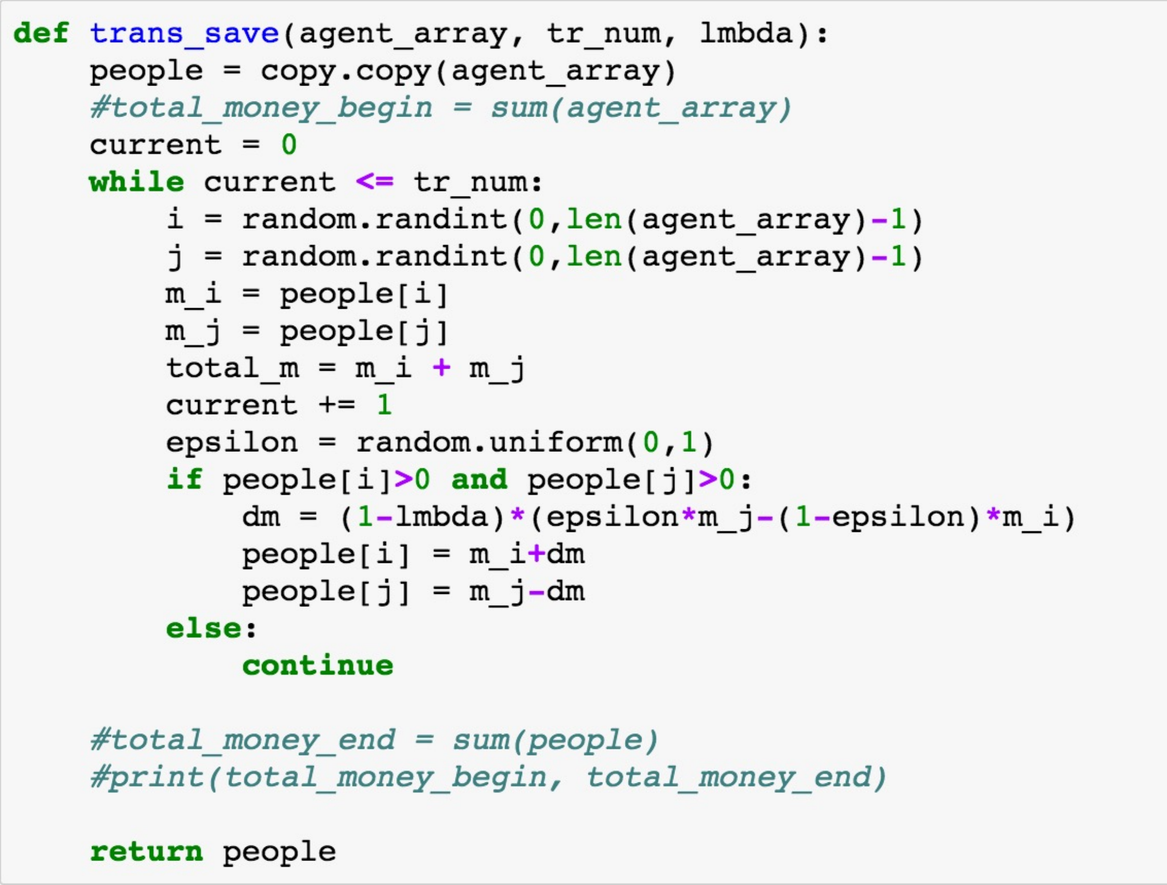
```python
def transactions(agent_array, tr_num):
    people = copy.copy(agent_array)
    total_money_begin = sum(agent_array)
    current = 1
    while current <= tr_num:
        i = random.randint(0,len(agent_array)-1)
        j = random.randint(0,len(agent_array)-1)
        m_i = people[i]
        m_j = people[j]
        total_m = m_i + m_j
        epsilon = random.uniform(0,1)
        current += 1
        if people[i]>0 and people[j]>0:
            people[i] = epsilon*total_m
            people[j] = (1-epsilon)*total_m
        else:
            continue

    total_money_end = sum(people)
    #print(total_money_begin, total_money_end)
```

PART TWO: SAVINGS

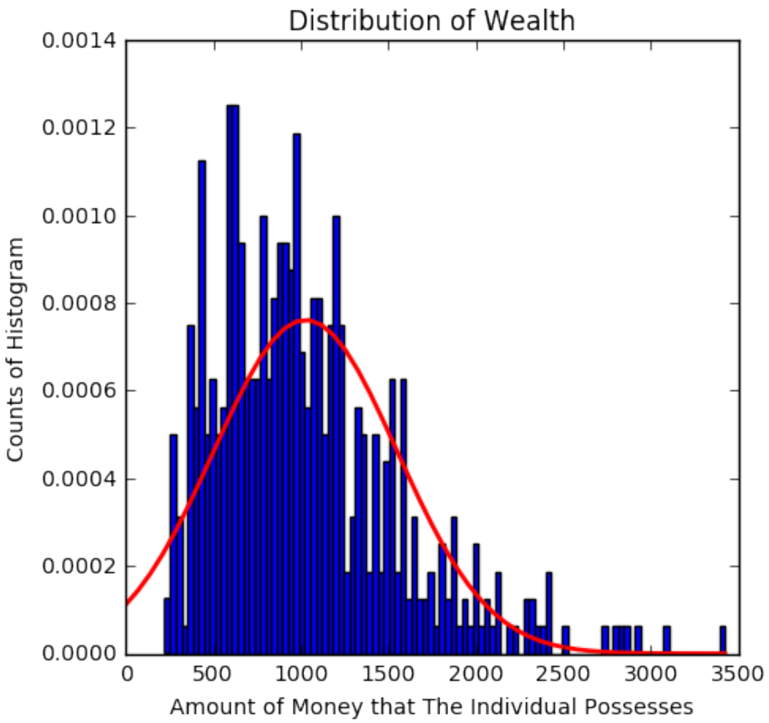
$$\delta m = (1-\lambda)(\epsilon m_j - (1-\epsilon)m_i)$$
$$m_i' = m_i + \delta m$$
$$m_j' = m_j - \delta m$$



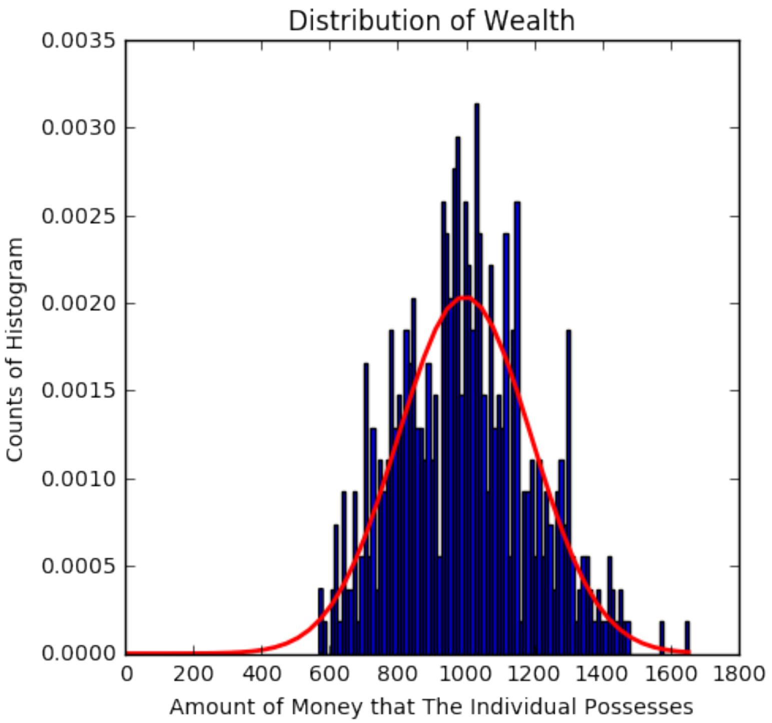
THE CODE



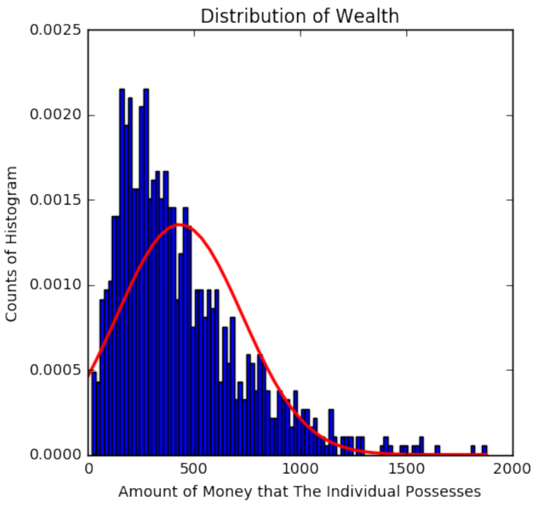
```python
def trans_save(agent_array, tr_num, lmbda):
    people = copy.copy(agent_array)
    #total_money_begin = sum(agent_array)
    current = 0
    while current <= tr_num:
        i = random.randint(0,len(agent_array)-1)
        j = random.randint(0,len(agent_array)-1)
        m_i = people[i]
        m_j = people[j]
        total_m = m_i + m_j
        current += 1
        epsilon = random.uniform(0,1)
        if people[i]>0 and people[j]>0:
            dm = (1-lmbda)*(epsilon*m_j-(1-epsilon)*m_i)
            people[i] = m_i+dm
            people[j] = m_j-dm
        else:
            continue

    #total_money_end = sum(people)
    #print(total_money_begin, total_money_end)

    return people
```

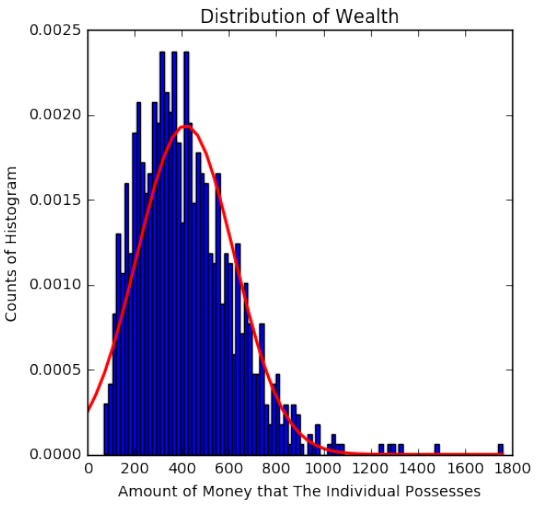


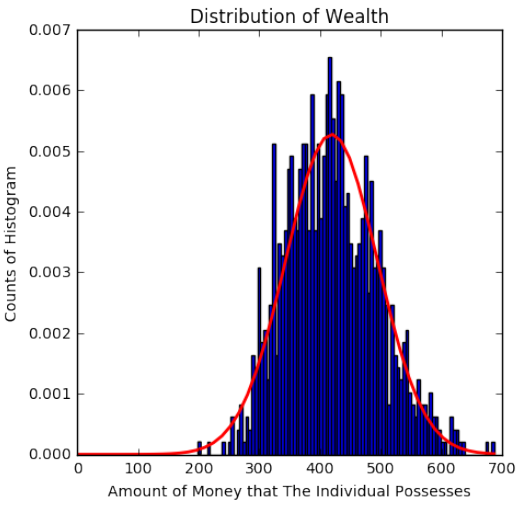
Distribution of Wealth



Distribution of Wealth


Distribution of Wealth
Counts of Histogram
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
0
500
1000
1500
2000
Amount of Money that The Individual Possesses


Distribution of Wealth
Counts of Histogram
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
0
200
400
600
800
1000
1200
1400
1600
1800
Amount of Money that The Individual Possesses


Distribution of Wealth
Counts of Histogram
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
0
100
200
300
400
500
600
700
Amount of Money that The Individual Possesses

SO FAR SO GOOD

- So Far We Were Able to Model Basics Transactions and Savings
- Stop to Think About the Results
- What’s Next?

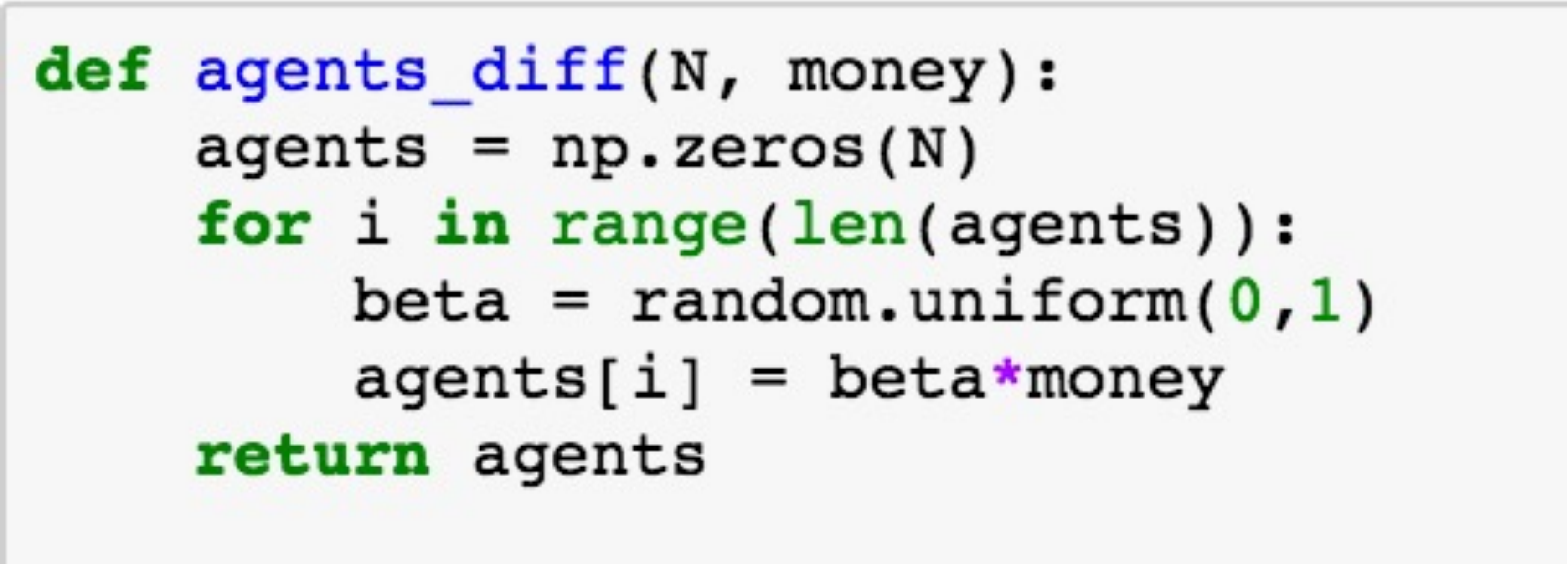
PART THREE: NEAREST NEIGHBORS

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

HOW THIS WORKS

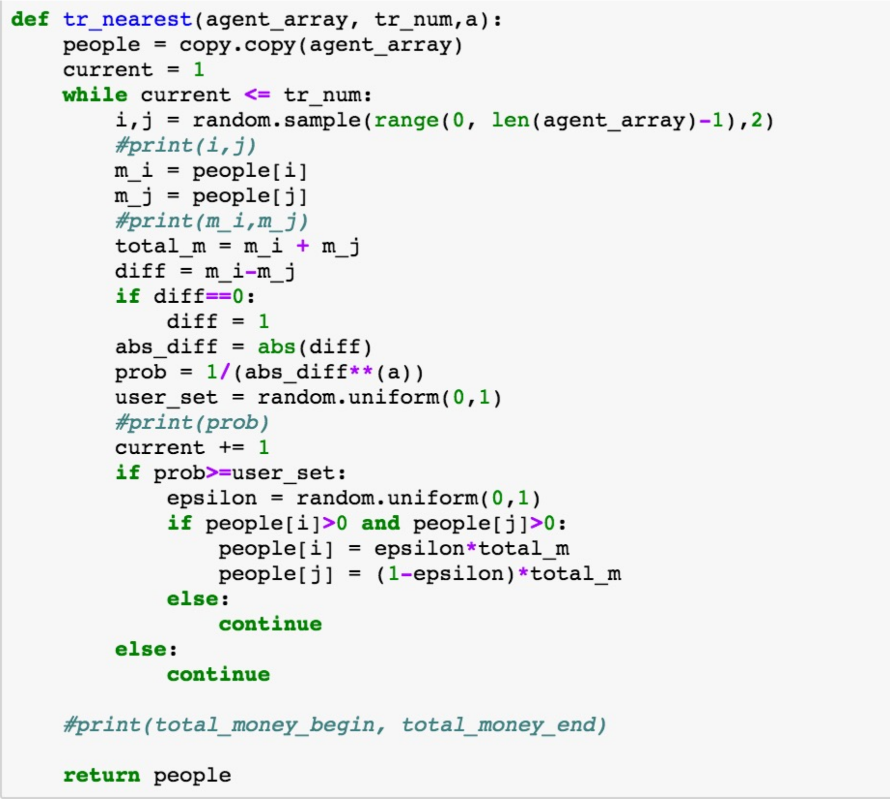
- Everyone Starts with Some Amount of Money (Why Can't It Be Uniform Anymore?)
- We Randomly Sample Two Indeces i,j (random.sample)
- Calculate the Difference in Their Wealth
  - If m_i = m_j, Then the Transaction Happens
- Compare to "user_set" (random.uniform)
  - If $p_{ij} \propto |m_i - m_j|^{-\alpha}$ >user_set, Then the Transaction Occurs as With The Base Model

MAKE SOME AGENTS



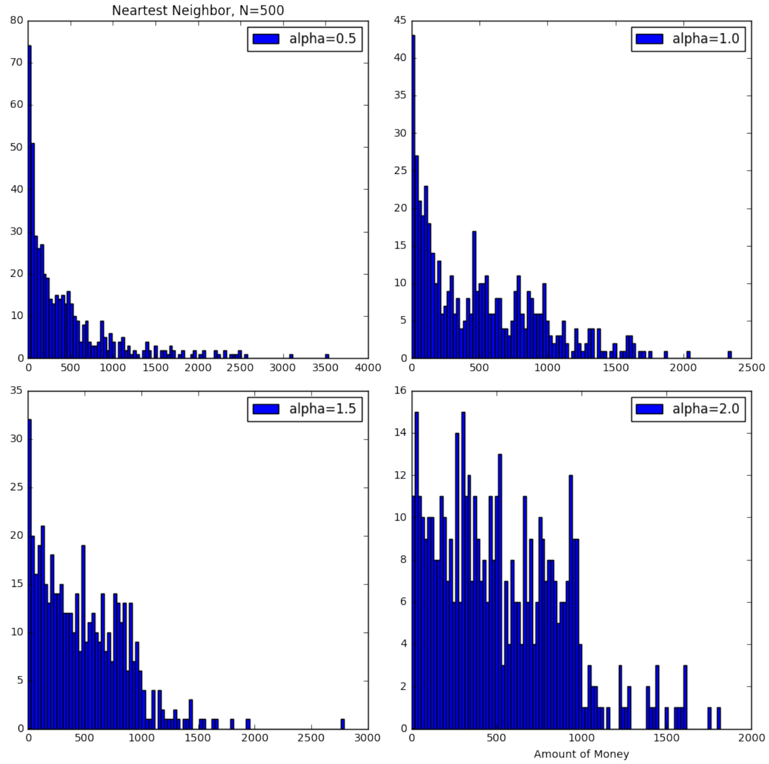
```python
def agents_diff(N, money):
    agents = np.zeros(N)
    for i in range(len(agents)):
        beta = random.uniform(0,1)
        agents[i] = beta*money
    return agents
```

THE CODE

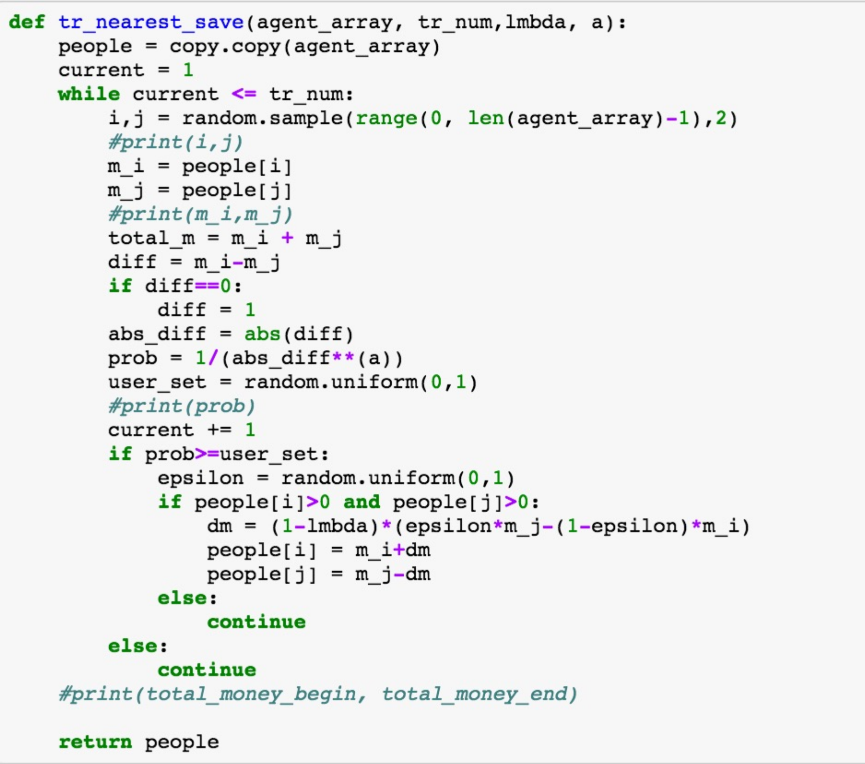


```
def tr_nearest(agent_array, tr_num,a):
    people = copy.copy(agent_array)
    current = 1
    while current <= tr_num:
        i,j = random.sample(range(0, len(agent_array)-1),2)
        #print(i,j)
        m_i = people[i]
        m_j = people[j]
        #print(m_i,m_j)
        total_m = m_i + m_j
        diff = m_i-m_j
        if diff==0:
            diff = 1
        abs_diff = abs(diff)
        prob = 1/(abs_diff**(a))
        user_set = random.uniform(0,1)
        #print(prob)
        current += 1
        if prob>=user_set:
            epsilon = random.uniform(0,1)
            if people[i]>0 and people[j]>0:
                people[i] = epsilon*total_m
                people[j] = (1-epsilon)*total_m
            else:
                continue
        else:
            continue

    #print(total_money_begin, total_money_end)

    return people
```

THE RESULTS FOR N=500, NO SAVINGS

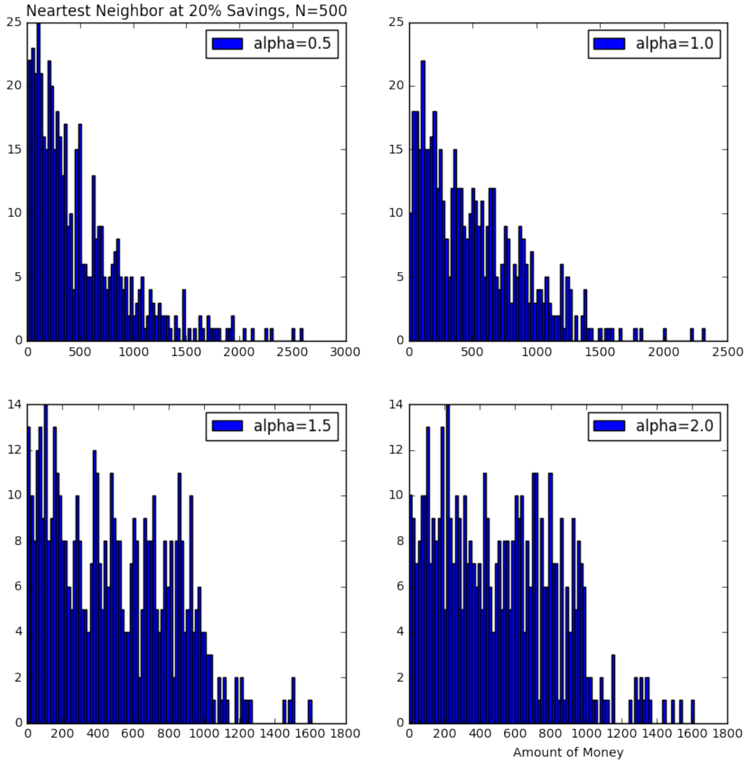


THE CODE

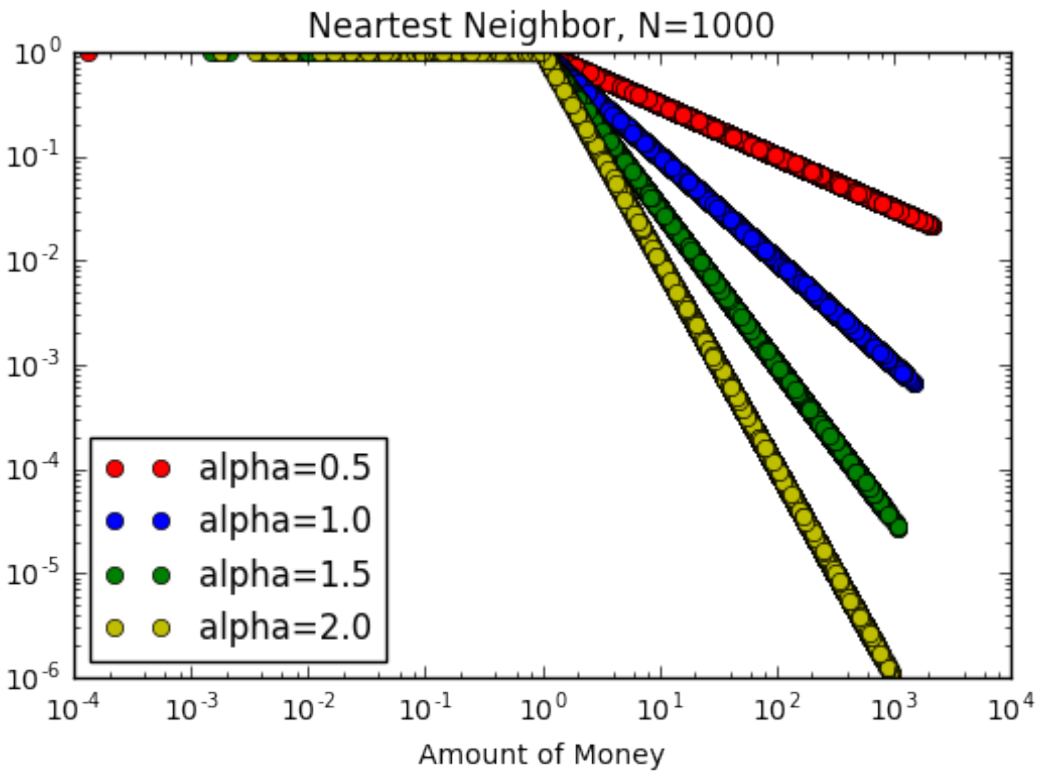


```
def tr_nearest_save(agent_array, tr_num,lmbda, a):
    people = copy.copy(agent_array)
    current = 1
    while current <= tr_num:
        i,j = random.sample(range(0, len(agent_array)-1),2)
        #print(i,j)
        m_i = people[i]
        m_j = people[j]
        #print(m_i,m_j)
        total_m = m_i + m_j
        diff = m_i-m_j
        if diff==0:
            diff = 1
        abs_diff = abs(diff)
        prob = 1/(abs_diff**(a))
        user_set = random.uniform(0,1)
        #print(prob)
        current += 1
        if prob>=user_set:
            epsilon = random.uniform(0,1)
            if people[i]>0 and people[j]>0:
                dm = (1-lmbda)*(epsilon*m_j-(1-epsilon)*m_i)
                people[i] = m_i+dm
                people[j] = m_j-dm
            else:
                continue
        else:
            continue
    #print(total_money_begin, total_money_end)

    return people
```

THE RESULTS FOR N=500, 20% SAVINGS



GOSWAMI AND SEN



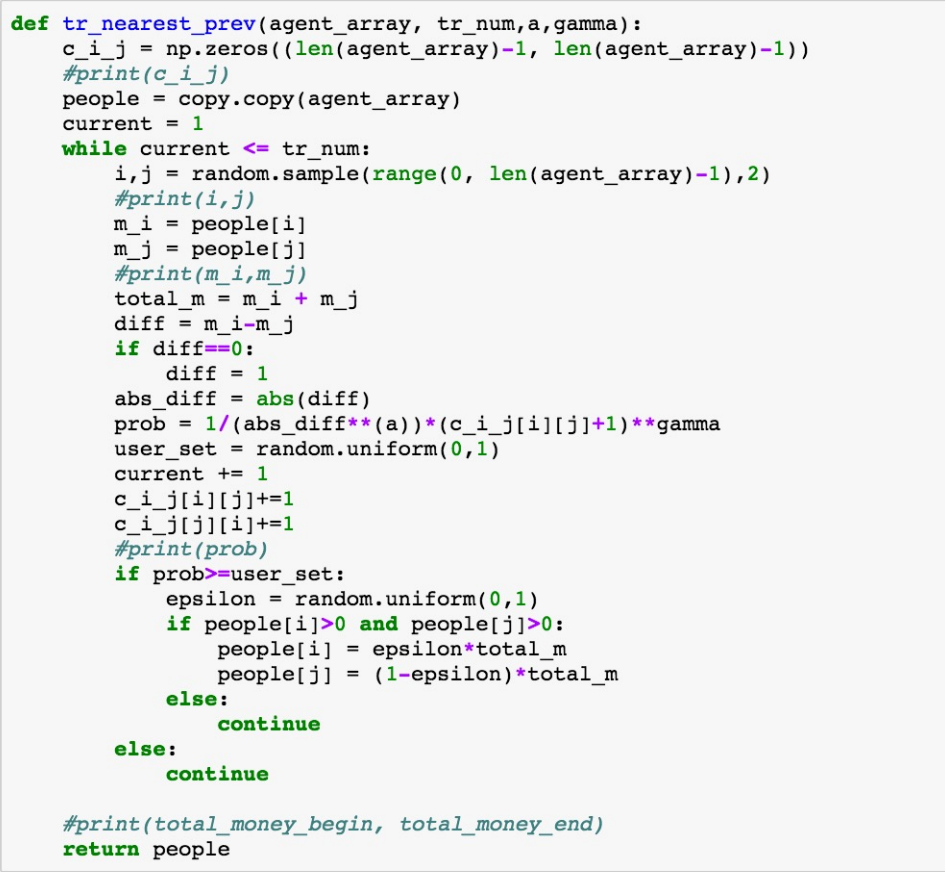
IT GETS SAUCY: PREVIOUS INTERACTIONS

- Now We Take One Extra Factor Into Consideration
- If People Have Interacted Before (Previous Transactions), Then More Likely to Exchange Money

THE MATHEMATICS

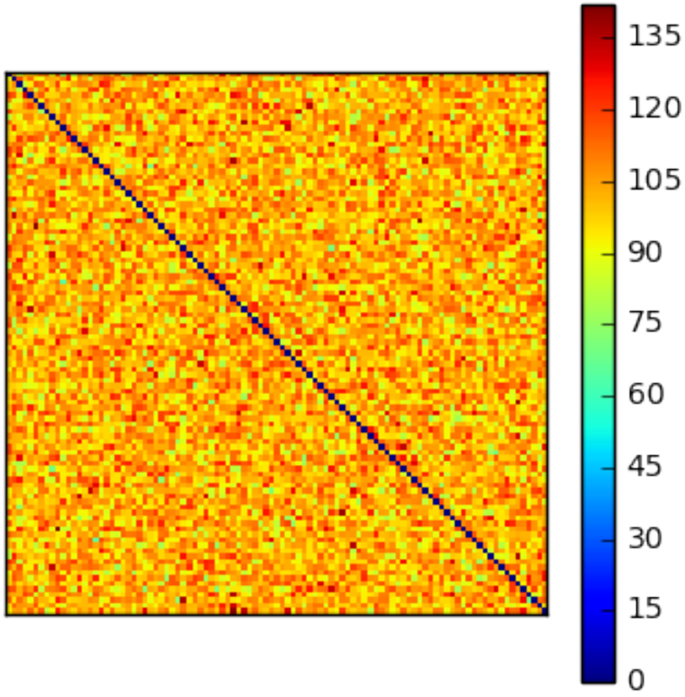
$$p_{ij} \propto |m_i - m_j|^{-\alpha}(c_{ij} + 1)^{\gamma}$$

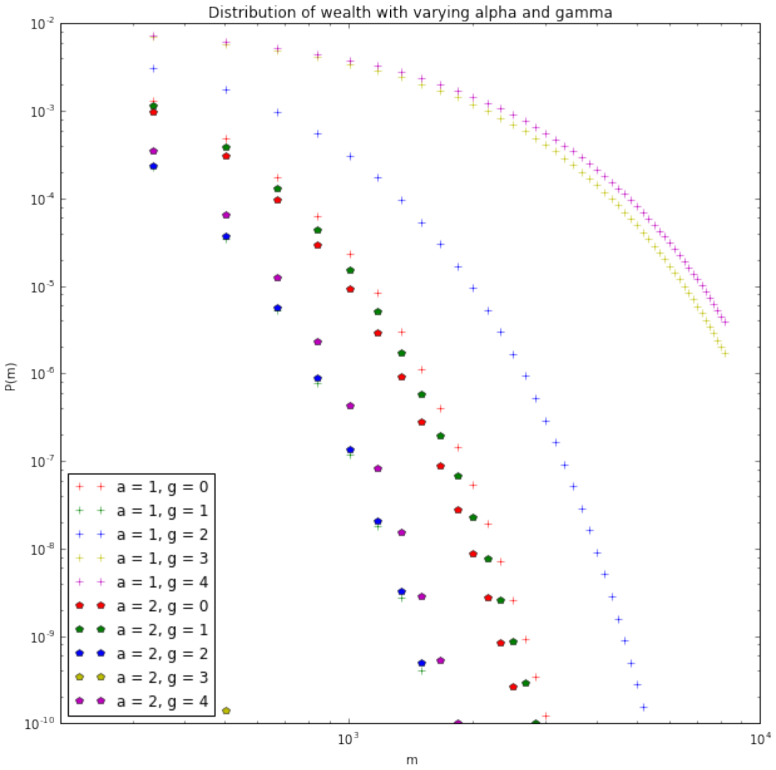
THE CODE



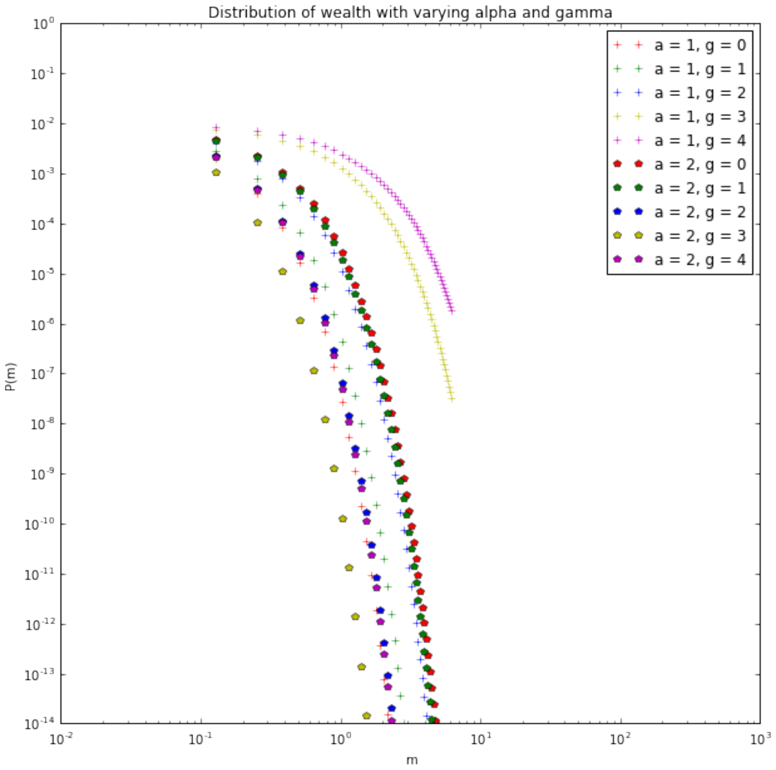
```python
def tr_nearest_prev(agent_array, tr_num,a,gamma):
    c_i_j = np.zeros((len(agent_array)-1, len(agent_array)-1))
    #print(c_i_j)
    people = copy.copy(agent_array)
    current = 1
    while current <= tr_num:
        i,j = random.sample(range(0, len(agent_array)-1),2)
        #print(i,j)
        m_i = people[i]
        m_j = people[j]
        #print(m_i,m_j)
        total_m = m_i + m_j
        diff = m_i-m_j
        if diff==0:
            diff = 1
        abs_diff = abs(diff)
        prob = 1/(abs_diff**(a))*(c_i_j[i][j]+1)**gamma
        user_set = random.uniform(0,1)
        current += 1
        c_i_j[i][j]+=1
        c_i_j[j][i]+=1
        #print(prob)
        if prob>=user_set:
            epsilon = random.uniform(0,1)
            if people[i]>0 and people[j]>0:
                people[i] = epsilon*total_m
                people[j] = (1-epsilon)*total_m
            else:
                continue
        else:
            continue

    #print(total_money_begin, total_money_end)
    return people
```

COOL COLORMAP



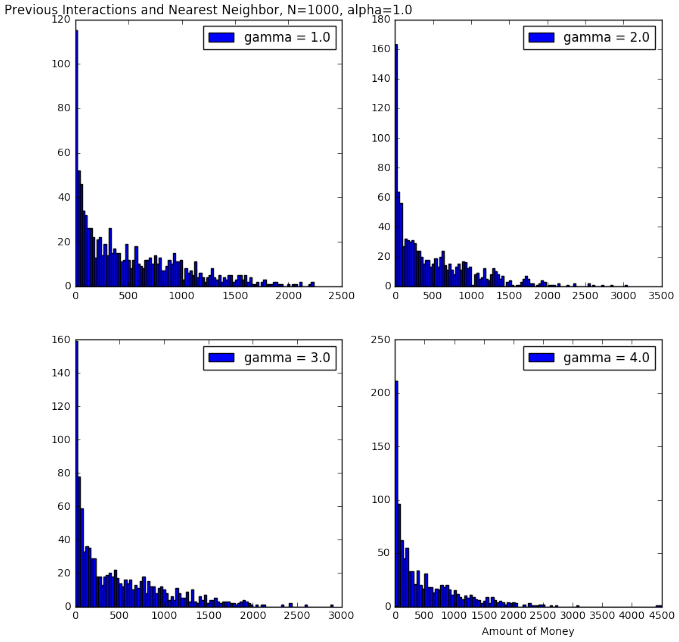


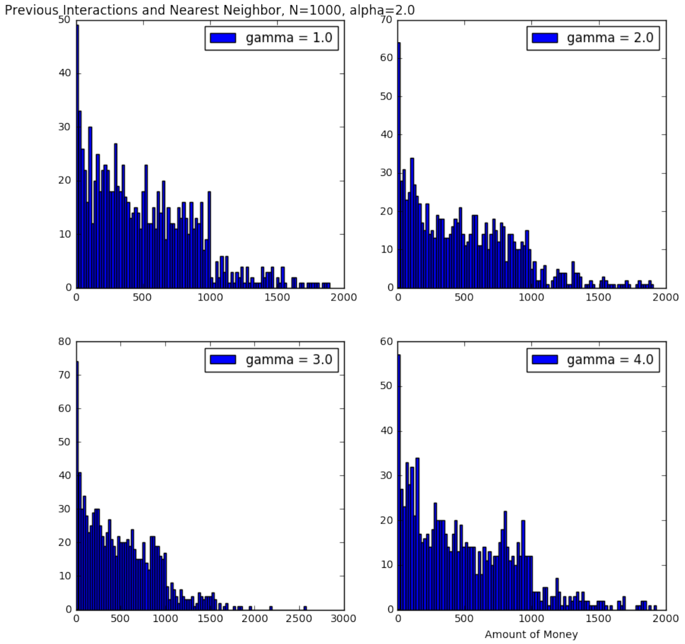
THE RESULTS: GOSWAMI AND SEN

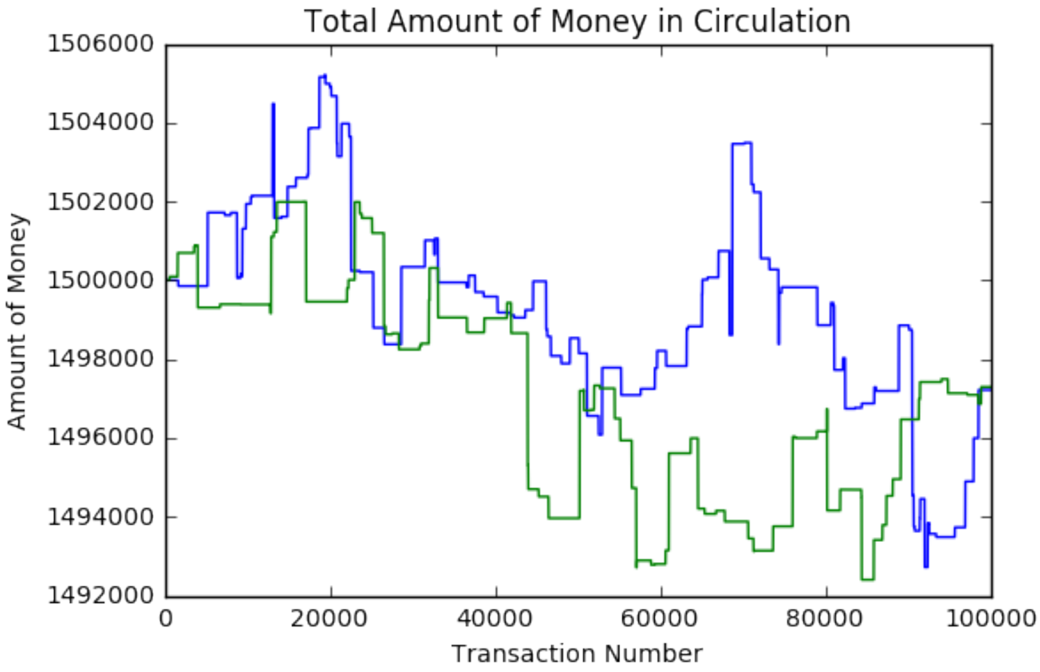


ANOTHER LOOK

LESS INTERESTING BUT COOL: DISTRIBUTIONS







NOW, THERE'S A PROBLEM

WHAT MORE COULD HAVE BEEN DONE?

- Interests on Savings
- Proximity Included in Nearest Neighbor
- One is the Business and Can Charge and Extra "Tax" and the Other the Consumer, Who Pays the Tax
- Good and Bad Experiences with Previous Interactions
- So Much More

SOME GENERAL REFLECTIONS

PROJECT 4 WAS COOL, BUT I LEARNED MORE THAN THIS

TOP 10 PHY480 LESSONS/TAKEAWAYS

1. Don't Try and Be More Clever Than the Code
2. Hardcoding is Only the Answer in Desperation (Absolute Desperation)
3. Beware Numerical Precision
4. CPU Time is a Thing, Be Nice to the Computer
5. Computation is About Expanding on Knowledge and Should be Fun
6. Language Can Have a Big Impact
7. Root Things in Algorithms, Let Math Be Your Friend
8. Get Git
9. I'm Capable of A Lot More Than I Gave Myself Credit For Originally
10. A Really Awesome Joke

THANK YOU FOR A GREAT SEMESTER, MORTEN