A multi-agent based approach for a macroeconomic model: the simplified economy of a country and the adoption of a new currency

Maurizio Dipierro, Pietro Foini, Giuseppe Ortolano, Daniele Rama

Abstract

The report presents a multi agent approach for a macroeconomic model populated by households and firms, which interact through the market of one only consumer good. This work focuses on the observation of some market quantities, including prices, employment rate, total firms profit and many others, aiming for stability and robustness to be found. Simulations over a wide set of parameters shows price stability and a plausible employment rate oscillation.

The second part of this work involves the presence of the government through the agent Public Administration, which interacts with the system purchasing goods and hiring workers. As well as before, the integrated model economy stabilizes after a transient.

The observed economy is conceived as the economy of a political and monetary union. This project deals with the research of the outcomes which could eventually take places after the withdrawal of a certain group of agents from such a union.

1 Introduction

Agent Based Models are used as a tool for modelling and observing interactions between entities, namely the agents, which perform some kind of actions in a well defined enviroment. In their early developments, the main applications of their usage were attempts to run sociologic and game theory experiments from a computational point of view; starting from the modelling of simple agents interaction rules, researchers were able to perform experiments characterized by non-trivial, unexpected and emergent phenomena.

Consequently, the techninques took place and covered many scientific areas including biology, ecology and many other kinds of complex systems, as for economics.

Indeed, economic processes could be interpreted either as the result of the interaction of multiple entities or as the evolution of a dynamical system. However, the latter approach needs to be analytical, i.e. one should be able to conceive the behaviour of all the variables involved; unfortunately, most of the times this is not possible.

For these reasons multi-agent approaches are particularly appropriate for both economic and macroeconomic models, given that they do not require to impose assumptions about the aggregate dynamics of the system and, at the same time, emergent phenomena are clearly expected. Many economists and complex phenomena researchers have been contributed to the development of this study area such as K. Stieglitz[5], C. Bruun [2] while in recent times CK. Chan [3] and many others.

Modelling a macroeconomic system allows to observe with accuracy the economy of a closed system, such as a country or a city. A market of one or more types of goods could be implemented in order to describe the evolution of a set of quantities (prices, total offer, total demand) which are common to all the actors, for example including households, firms, finance institutions and the government.

This is the aim of the present project: firstly, to develop a simplified macroeconomic model able to reproduce a stablility scenario, using few kinds of agents and secondly, to observe the results of the detachment of a group of a small number of agents which give birth to a new, parallel economy.

For the authors, the observed whole economy represents the european one and, at the same time, the withdrawing group of agent is identified with the italian country. The evolution of some cumulative economic indicators, such as the GDP, the deficit and the ratio deficit over GDP is studied for both the groups.

Recently, studies in this direction reached academic and political interest: for instance, the Brexit's referendum results have raised questions about the consequences of this detachment, and it is possible to find some answers in the Samitas et al. paper[4], in which an agent based simulation framework is employed in order to test the effects of Brexit on financial stability on both sides of the Channel. Moreover, the effects of an eventual italian withdrawal from the European Union are discussed in the paper[1] written by Bagnai et al. This work aims to go further on the same path of the previous ones.

2 Model

In our model a formerly united group of agents is divided into two subgroups, with one bigger than the other, in an attempt to reproduce the macroeconomic dynamics of Europe and Italy respectively. Agent in our models are of three different classes:

- Firms
- Households
- Public Administration

Each of those agents will have an attribute indicating which group (Italy or Europe) it belongs to. These agents interact with each other following basic rules of economy. The households, if employed, receive a wage from the firms and consume a fraction of it every cycle. This, in conjunction with the investments from all the agents, the consumption from the unemployed households[1] and the Public administations (PAs) expenditure, is added to form the aggregate consumption that define the demands for both groups.

Firms and households will individually decide from which group to buy, while the PAs will automatically buy from their own group. So the two resulting demands, one for each group, with the respective productions, will be used to determine two different market prices. Those different prices will then evolve in parallel, according to the dynamics of the model. All workers are employed in their own group and the wage is different for Italian and Europeans as an effect of currency devaluation.

Only one consumer good is produced by all the firms, but since every agent is able to buy from the other group there will be a commercial balance nonetheless. The production is proportional to the workers of each firm and is tuned at each cycle with hirings or firings, eventually. A continued profit or loss of a firm determines respectively the expansion or bankruptcy of the firm.

The agent public administration controls the financial health of its group, i.e. it checks that the deficit is under a certain limit. For Europe no politics are implemented since its assumed that the trust from both its own citizen and foreign investors, will make up for an higher deficit. For Italy, especially in case of the adoption of a new currency this may not be true. So a policy that keeps the deficit under a certain limit, namely the Maasthrich limit of 3 % of the GDP (actually raised to 5 % in our model), needs to be implemented.

The only interaction between agents of the two groups is then the crossed demand, since neither a change of nationality nor to work for another group is allowed to the agents of each group.

[1]Unemployeds receive a benefit.

2.1 Agents

In this section the primary variables of each agent are explained, along with their main interaction procedures.

2.1.1 Households

Households attributes initialization are shown in the code below:

```
class Household:

    def __init__(self, is_employed):
        self.is_employed = is_employed
        self.my_group = my_group
        self.wage = 0
        self.consumption = 0
        self.hasInvested = False
        self.debt = 0
        self.addInvestmentToDemand = False
        self.nInstallmentsleft = 0
        self.savings = 0
```

Listing 1: Households initialization.

Attributes and initialization The leading attributes of the agent class Household are its employment status, namely ***is_employed*** and the group it belongs to, called ***my_group***.

The attribute ***is_employed*** is a boolean variable and, if true, the households perceive a wage, the same for all workers belonging to the same group. Otherwise the only income of the unemployed households is the unemployment benefit, ***unempBenefit***, which is equal to 30% of the normal wage. The attribute ***my_group*** is a binary variable: It is 0 for the Italian agents and 1 for the european ones.

The other households' attributes are:

- ***wage***. The perceived wage equals for all workers of each group and is reduced to 30 % in case of unemployment. The values given are 1 for European wages and 0.9 for Italian ones.
- ***consumption***. Total consumption for the cycle relative to the household.
- ***addInvestmentToDemand***. Each agent has the possibility to invest at a given cycle. This boolean variable records if an investment occurred this cycle and if true adds a constant quantity to the demand.
- ***hasInvested***. A boolean variable that records if the household has any previous debt left to repay.

- ***nInstallmentsleft***. A counter for how many cycles are left until the debt is repaid.
- ***debt***. A real parameter that determines the percentage of wage that is detracted each cycle from the wages of those who invested. Only the interest on the original loan is repaid.
- ***savings***. A value that keeps track of the amount each household did not consume each cycle.

Interactions and functions A household consumes a part of its wage unless it invests. It interacts with the market through its consumptions; the latter are computed using the function **calcConsumption** and its code is shown in appendix A. This is the most important function for the households.

The first attribute that is checked is ***is_employed***. If $False$ the household consumes its whole benefit. If $True$ the consumption is computed using the following formula:

$$consumption = a + b * (wage - debt) + \epsilon \tag{1}$$

where $a = 0.2$ represents sort of a minimum consumption, $b = 0.6$ is the fraction of the wage which the household chooses to consume and ϵ is a normal distributed variable $N(0, 0.1)$; the ***debt*** is set with the parameter ***pmt.repaymentRate*** which is equal to 0.2 if the household is still to repay a debt, otherwise vanishes.

At every cycle each household has a probability $p = 5\%$ to invest. If an investment occurs a quantity of 5 unities is added to their consumption. Only the interest on the investment is repaid for the following 10 cycles, during which the household is prevented from investing again.

Once set the quantity to consume, the agent has to decide in which group to consume. This is done using the **ProbConsOwn** function. This function returns a value for the probability of the agent to consume in its own group. The parameters involved in the determination of this value are:

- ***pmt.chauvinism***. This parameter accounts for the inclination of agents to buy in their own group, due to acessibility, import difficulties and even policies.
- ***pmt.Indipendence*** A number between zero and one that, even in a single good market such the one in the model, accounts for the fact that some goods must necessarily be brought outside the group [2]. The actual values chosen are discussed below.
- ***pmt.sensitivityToPrice*** A parameter that is inserted to determine the component of the probability concerning the difference of prices

[2]e.g. energy for Italy

ΔP. This probability is computed by taking the hyperbolic tangent of ΔP times the parameter, so that it controls how sensitive agents are to different prices.

The other functions are:

- **setEmploymentStatus** is the function which access the ***is_employed*** variable and sets its value to $True$ if the household is a worker and $False$ otherwise.
- **setInvestmentstatus** is the function that access and sets the value of the variable ***hasInvested*** to $True$ or $False$ depending on whether the household has an open investement or not.
- **calcSavings** this function computes the fraction of the wage a household chooses not to consume. For the unemployeds there are no savings since all of the benefit is consumed.
- **getMyGroup** this function access the variable ***my_group*** and returns its value.

2.1.2 Firms

Firms attributes initialization are shown in the code below:

```python
class Firm:

  def __init__(self, nWorkers):
    self.nWorkers = nWorkers
    self.my_group = my_group
    self.prodPerWorker = pmt.prodPerWorker
    self.wage = pmt.wage[self.my_group]
    self.firmHealth = 0
    self.hasInvested = False
    self.firmPrice = 0
    self.profit = 0
    self.production = 0
    self.debt = 0
    self.nInstallmentsleft = 0
    self.addInvestmentToDemand = False
    self.consumptionForInvestment = 0
```

Listing 2: Firms initialization.

Attributes and initialization The characterizing attributes of each firm are its number of workers and its profit, as well as the attribute ***my_group*** that has the same role it had for the households. The first one determines its productivity (***production***), which is proportional to ***nWorkers***, while the ***profit*** is computed as the difference between revenues and the firm expenditure.

$$production = prodPerWorker * nWorkers \tag{2}$$

As for the households firms can invest and the variables ***addInvestment-ToDemand***, ***hasInvested*** and ***nInstallmentsleft*** work in the same way discussed before but, in this case, the investment results in the creation of a new firm.

The other leading variables which set the firm's behaviour are:

- ***debt***. A real parameter that is detracted each cycle from the profit of the firm who invested. As before, only the interest on the original loan is repaid.
- ***firmHealth***. A counter that goes up by one every cycle there is a positive profit and down by the same amount every time it is negative.
- ***firmPrice***. A variable that states the price different for each firm.
- ***consumptionForInvestment***. Firms investments are not fixed but they depend on how many workers are hired for the new firm created by the investment.
- ***wage***. The value of the wage the firms pays to its employees.

Interactions and functions The productivity of each firm is computed using the function **calcProduction** which follows the equation (2).

As mentioned before, each firm applies a different price which is computed using **setFirmPrice**: given a market price determined as the aggregated demand over the total offer concerning its own group, this function returns a new price obtained adding a random quantity normally distributed around zero and with a variance of 20% of the market price.

Another important function is **checkHealth** that checks whether or not a firm is able to invest: if ***firmHealth***$= 5$ the firm invests. Analogously, if ***firmHealth***$= -2$ the firm goes bankrupt.

An investment from a firm results in the creation of a new firm in its own group. The amount of quantity for the investment is related to the number of workers of the incumbent firm; the investment's interests, i.e. the debt, are repaid by the firms at the same rate as it was for the households, reducing the firm profit. This implies that, given a variable investment the number of cycles needed to repay it will vary accordingly.

Those investments are the only source of consumption for the firms, so they fully determine the result of the function **calcConsumption**.

```
def calcConsumption(self):

    if self.addInvestmentToDemand == True:
        consumption = self.consumptionForInvestment
        self.addInvestmentToDemand = False
    else:
        consumption = 0
    return consumption
```

Listing 3: The function **calcConsumption** calculates the consumptions of a firm.

The profit is computed using the **calcProfit** function.

```
def calcProfit(self):

    self.profit = self.production*self.firmPrice - self.nWorkers
   *pmt.wage
    if self.hasInvested == True:
        self.profit -= self.debt
        self.nInstallmentsleft -= 1
        if self.nInstallmentsleft == 0:
            self.hasInvested = False
            self.debt = 0

    if self.profit < 0:
        self.firmHealth -= 1
    else:
        self.firmHealth += 1

    return self.profit
```

Listing 4: The function **calcProfit** computes the profit of a firm.

As said before, after this function is computed the ***firmHealth*** counter is updated.

Each cycle the firm plans its future production as follows:

$$P_{t+1} = P_t \left(\frac{D_t}{D_{t-1}} \right) \tag{3}$$

where P is the production and D the demand.

Since in our model the productivity factor $prodPerWorker = 1$, the offer equals the number of workers required and the rounded[3] difference $P_{t+1} - P_t = \Delta$ is exactly the number of workers a firm has to hire (or to fire) in order to meet the production plan.

The other functions are:

- **setInvestmentStatus** is the function that, if a firm's investment occurs, access and sets the value of ***hasInvested*** and ***addInvestmentToDemand*** to $True$. Moreover, it takes as argument the number of

[3]To the closest integer.

workers of the new firm to be created[4] in order to compute the number of installments that are left to be repaid.

- **getNumWorkers** returns the number of workers of the firm when is called.
- **updateNumWorkers** changes the number of workers of the in case of hiring or firing.
- **getMyGroup** A function that access the variable ***my_group*** and returns its value.
- **ProbConsOwn** The function that determines where the firm will invest. It works in the same way decribed before for the households.

2.1.3 Public Administration

Public Administration is the agent reflecting the action of the government in the system; this intervention results in purchasing goods from its own market and in hiring a given number of households, which is constant during the evolution of the model.

The introduction of this agent allows to observe a certain number of new quantities which are useful to evaluate the health of a nation like the deficit, the public debt and the GDP.

Households savings are fundamental to calculate both the deficit of a nation, which is the difference between the total savings of the households and the public expenditure, and the public debt, which results in the cumulative deficit over the whole simulation.

As stated before, after the withdrawal from the EU zone, i.e. the common market, the Italian public administration may find difficulties in implementing a monetary policy that arbitrarily exceed the deficit over GDP ratio. Indeed, even if Italy should not respect the limitations imposed by the Maastricht Treaty [5] anymore, on the other hand the withdrawal may undermine the confidence of the foreign investors in our country. As almost one third of Italian debt is owned by foreign investors, this could heavily limit the possibility to implement spending policies. Therefore, to account for this situation, we decided to introduce a function that reduce the purchasings of the PA in case deficit exceeds a given limit.

```
class PubAdm:

    def __init__(self, nWorkers):
        self.nWorkers = nWorkers
        self.wage = pmt.wage[self.my_group]
```

[4]Due to the investment.

[5]The Maastricht Treaty limits the deficit over GDP ratio of a member state to 3%.

```
self.my_group = my_group
self.purchasings = 0
self.PublicExpenditure = 0
self.deficit = 0
```

Listing 5: Public Administration initialization.

Attributes and initialization As mentioned before, the number of public employees is fixed in the model: ***nWorkers***= 27% of the initial number of italian workers[6] and ***nWorkers***= 25% of the initial number of European workers. The other main attributes are:

- ***purchasing***. A real variable that stands for the quantity of goods purchased by the PA; this value obviously determines the total demand of the country to raise.
- ***PublicExpenditure***. Every cycle the PA purchases goods and pays its workers. The sum of these two quantity results in the ***PublicExpenditure***
- ***deficit***. A real variable which represents the value of deficit updated at each cycle.
- ***my_group***. It is a binary variable: 0 for the Italian agents and 1 for the european ones.
- ***wage***. variable indicating the wages of the public workers for each group

Interactions and functions In order to compute the Public Administration consumptions, the function **calcConsumption** has been implemented: this function returns the value of ***purchasing*** evaluated as:

```
def calcConsumption(self):

    wages = self.nWorkers*pmt.wage
    self.purchasings = wages - random.randint(0, round(pmt.
   percPurchasingsPA*wages))
    return self.purchasings
```

Listing 6: The function **calcConsumption** computes the consumptions of the Public Administration.

where *percPurchasingsPA* is a parameter of the model which equals 0.1.

Another main function is **calcDeficit** which evaluates the deficit, as stated before, and a controls the index, namely ***ControlMaastr***, calculated as the ratio of the deficit to the GDP. For the european countries, this

[6]The italian public employment is 27% of the total italian workforce, `https://www.truenumbers.it/dipendenti-pubblici-2/`

index must not exceed the 3% as stated in the Maastricht Treaty. The other functions are:

- **getNumWorkers** returns the number of workers of the public employees when is called.
- **calcPublicExpenditure** is the function that computes the sum between purchasings of the Public Administration and the wages of the public employees.
- **getMyGroup** A function that access the variable *my_group* and returns its value.

2.2 Run

In this section the computational details of the simulation workflow are given in terms of system's initialization and functions involved.

2.2.1 Model initialization

The leading parameters of the model are the number of firms, the number of households and the number of cycles the simulation will perform. These three numbers are requested to the user who runs the file *start.py* and are used to generate the macroeconomic enviroment.

The enviroment generation happens in parallel for the two groups and follows a meticulous schedule: first, the public admistrations are created for the two groups and a certain fixed number of workers is hired becoming the public employment; then, firms are created and, immediately, a certain number of workers from of the same group are connected to each of them. As a matter of fact, every firm is identified through an index which allows the link between firm and its own workers. The total number of workers for each group is obtained deducting the corresponding unemployment rate percentage from the number of households. At the initialization stage, the paramater that control this rate is ***pmt.unemploymentRate*** which is fixed[7] ≈ 0.11 for Italy and ≈ 0.07 for Europe.

With the initial parameters mentioned so far the averge number of workers per firm is computed, namely ν. Thus, each firm is provided with a number of workers equals to ν plus a perturbating term randomically extracted from -10 to 10, in order to vary the distribution of employees per firm.

In this step, the variable ***is_employed*** of every worker is obviously set to $True$ and remains $False$ for the unemployeds.

[7]Sources: TheGlobalEconomy.com, 2018

During the initialization of the simulation, the variable ***my_group*** allows to differentiate each agent (both households and firms), which will belong to the Italian group (label 0) or the EU one (label 1).

The function performing these actions is **buildObjects**; a part of the funcion, showing the inizialization for italian agents is shown below.

```
def buildObjects(self):

  self.nu.append(round((nTotWorkersItaly/nTotFirmsItaly) * (1 -
    pmt.publicEmploymentRateItaly)))
    nTotWorkersItaly = 0

    for i in range(nTotFirmsItaly):
      aFirm = Firm(nWorkers = self.nu[0] + random.randint(-10,
   10), my_group = 0)
      self.firmsDict[self.idFirm] = aFirm
      nTotWorkersItaly += askAgent(aFirm, Firm.getNumWorkers)

      hList = []
      for j in range(askAgent(aFirm, Firm.getNumWorkers)):
        hList.append(Household(is_employed = True, my_group = 0)
   )
      self.householdsDict[self.idFirm] = hList
      self.idFirm += 1
    nTotWorkersItaly += workersPubAdmItaly
    self.nTotWorkers.append(nTotWorkersItaly)

    nUnemployedItaly = totHouseholdsItaly - nTotWorkersItaly
    self.nUnemployed.append(nUnemployedItaly)
    hList = []
    for i in range(nUnemployedItaly):
      hList.append(Household(is_employed = False, my_group = 0))
    self.householdsDict[0] = hList
```

Listing 7: Part of function **buildObjects** that generates agents for the group Italy.

Clearly the Public Administration agents, which are the first agent generated, decrease the number of workers avaliable for the firms; actually, a certain percentage of workers, controlled by the parameter ***pmt.publicEmploymentRate***, is assigned respectively to the italian PA, 27%, and to the european one, 25%.
.

2.3 Model actions

After the creation of the enviroment the model performs the following scheduled actions, in due order; all these actions represents the development of an unique cycle. Almost every action listed before is performed independently for each group, following their own dynamics. However certain actions require an interaction between the two groups.

1. At first, the total supply, P, of goods in each market is calculated summing the production of each firm of the group; in order to do this, the function **calcProduction** is called.
2. The total demand for a group, D, is computed summing the single consumptions of each household and of each firm (from both groups) that decided to buy in the group; as mentioned before, every cycle households automatically consume and invest eventually (with a given probability); rather, firms can consume only through investments (which occurs after a certain period of positive profit); these action are computed for both the agents Households and Firms through the corresponding two functions **calcConsumption**. The decision about from which group to buy is controlled by the function **ProbConsOwn**, that returns a value of probabilty of buying in their own group both for firms and households, while the final demands are returned by the function **calcDemand**, that counts the contribution of the PAs as well, that add to the demand of their respective group by a quantity given by the variable ***purchasings***. At each cycle the total demand of each group is saved in memory.
3. Using the quantities computed so far, the group market price is computed as $marketPrice = D/P$; the function involved is **setMarket-Price**.
4. Given the market price of the group, every firm applies its own price through the function **setFirmPrice**; as said before, this results in a certain distribution of prices in the market.
5. The trade happens and every firm calculates its profit with the function **calcProfit** and the total profit is evaluated summing each firm's profit.
6. At this point the health status of each firm is examinated through **checkFirmsHealth**; this function saves the indexes related to both the firms which can invest and the firms which are going bankrupt.
7. Firms which are going bankrupt are removed from the market through the function **removeFirms**: this removal happens passing the indexes mentioned before as argument of the function; consequently, the ***is_employed*** status of each former worker is set to $False$.
8. As a consequence of a firm's investment a new firm could be generated[8]

[8]New firms are created following exactly the same procedure of the model initialization. Therefore the incumbent firms will have about the same number of workers of the starting ones.

inside the same group. This procedure happens through the function **addFirms** which takes the indexes of the virtuous firms as argument; then, a new firm is created only if there are enough unemployeds in the group.

9. After the first cycle, every firm plans its production for the following time step. This is performed with the function **planProductionand-HireFire** which returns the integer Δ; referring to the section 2.1.2, if Δ is negative the firm needs to fire workers in order to match the planned production; instead, if Δ is positive, the firm needs an extra workforce and hires: anyway, the hiring happens only if there are enough unemployeds; as mentioned before all the hirings and firings happens within the group .
10. Finally, a certain number of quantities regarding the status of each group are computed: at first the public expenditure is evaluated and it is used to compute the GDP. As a matter of fact, the GDP is calculated for both the groups as $GDP = D_h+D_f+pubExp+tradeBalance$ where D_h is the households' demand, D_s the firms' demand (investments that are done in the given group) and both of them are computed with the function **calcDemand** while, the trade balance is the difference between the amount of exported and imported goods in an economic system.
    Another economic indicator of the system which is taken into account and computed is the deficit, calculated as the difference between the total households' savings and the public expenditure. Clearly it will be different for the two groups.
    Then, the ratio of the deficit to the GDP is observed in order to realize wheter the system is acceptable for the rules of the Maastricht Treaty or not (the actual limit percentage taken is 5%). Here the italian policy described before comes into play. If the percentage is higher than the limit the purchasing are reduced by 10% at each cycle, until it returns under the limit. If it is under the limit the purchasing are increased up until a maximum value that is the starting one. This adjustments are done by the global function **controlMaasForItalyPA**.
    The last relevant quantity which is computed is the public debt for each group and it is calculated adding the current deficit to the previous cumulative deficit.

The set of all of the previous steps forms an entire cycle; the whole simulation lasts the number of cycles which the user has requested.

Model parameters initialization		Europe	Italy
Global	**unemploymentRate**	0.07	0.11
	repaymentRate	0.20	0.20
	independence	1.00	1.00
	chauvinism	0.50	0.50
	sensitivityToPrice	1.50	1.50
Firms	**prodPerWorker**	1.00	1.00
	sigmaPrice	0.05	0.05
	goBankrupt	-2.00	-2.00
	doInvest	5.00	5.00
Households	**wage**	1.00	0.90
	unempBenefit	0.30	0.30
	pInvest	0.05	0.05
	nInstallments	10.00	10.00
	demandInvestment	5.00	5.00
Public Administration	**publicEmploymentRate**	0.25	0.27

Table 1: Initial model parameters for both european and italian markets.

3 Results

In this section the results of the model under different parameters, representing different scenarios are shown; for each scenario a certain number of average quantities have been observed and the results are summarized in the table 2 at page 28.

Some parameters, mostly the ones regarding the individual economies of the two groups, are left unchanged for the different scenarios. Those parameters are showed in table 1 for both Italy and Europe.

The following average results arise from simulations over a fixed number of cycle, ***nCycles*** $= 100$. This choice deals with the interpretation of a time step: given that firms behaviour involves hirings, firings and production plans, it seems valid to assume a cycle to be equivalent to a quarter. This interpretation does not actually affect the households, which in the model perceive a certain wage and consume a fraction of it: as a matter of fact, one could take this behaviour as an intensive property of a household.

The other parameters of the model are the ones used to determine the probability to consume in a given group and will be changed in the following subsections to simulate different scenarios. They are:

- Chauvinism
- Indipendence

- Sensitivity to price
- Weights for sensitivity to price and chauvinism (1,0)

3.1 Before the separation

The first scenario considered is the one where does not exist an actual monetary separation. In terms of the model parameters this can easily be achieved putting the wages for both groups at the same value (meaning that no separation with a following devalation has occourred). The chauvinism is then set to 0.5 for both groups.

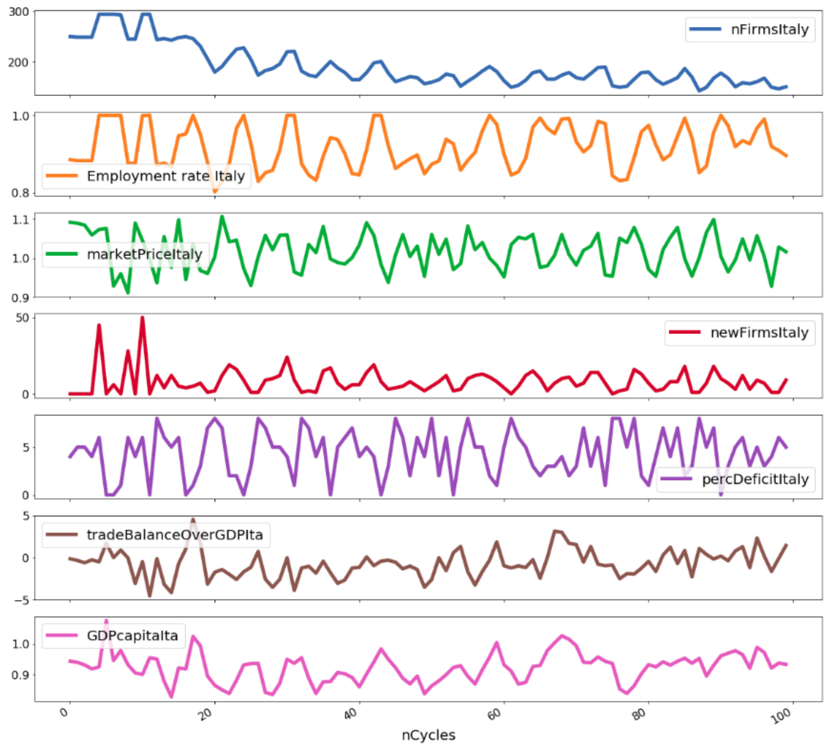


Figure 1: Italy without separation

This means that, without considering anything else, each agent has a 50% probability of buying in his own group, i.e. no preference toward any group is given. Then the weight for the sensitivity to prices is put to zero, meaning that no preferences are given to cheaper prices.

Thus, considering the normalization of the probability proportional to the number of workers in each market (and so to the productivity), this scenario is similar to considering a single group of agents where just some initial variables change by a neglectible amount (e.g. the initial unemployment rates). The outcomes for this scenario are shown in figure 1 for Italy and in figure 2 for Europe.

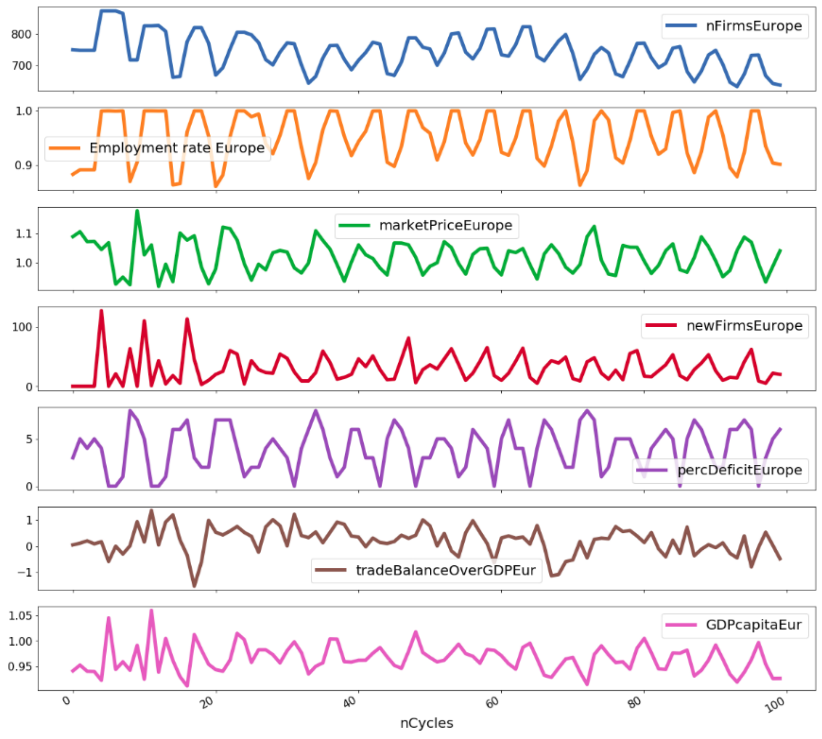


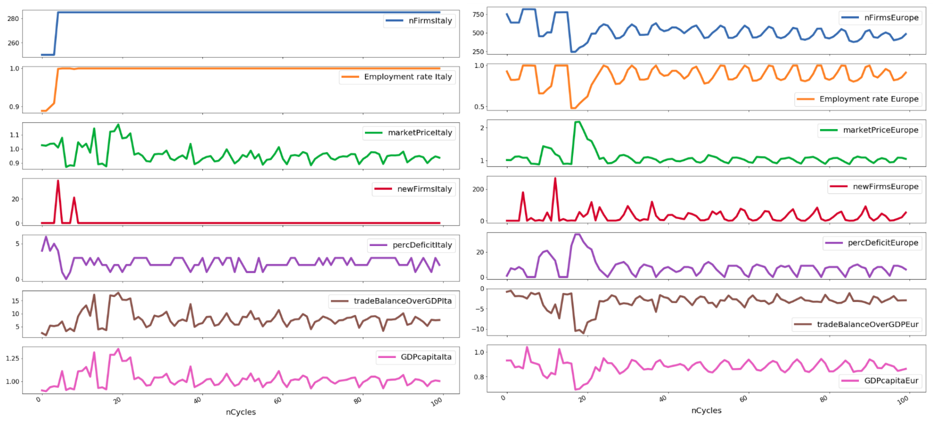
Figure 2: Europe without separation

As it can be seen, both markets are stable and their quantities are similar. The trade balances oscillate around zero because since there is no actual difference between there is no reason to have a net trade in either direction.

3.2 Devaluation

Following the precedent section a monetary devaluation is introduced, meaning that both italian wages and initial market price are decreased to 0.9, while all other parameters are left unchanged.

As we can see in fugure 3, devaluation leads to a growth of the Italian economy that quickly reaches its maximum, with a 100% occupation rate, and stays this way until the end of the simulation while Europe keeps its previous stability. This results can be easily explained if we analyze the fact that, altough the italian firms are selling at a lower price, they are paying their workers 10% less and, as a consequence, they are raising their profit. The decrease in consumption for the italian citizen impacts the european



(a) Italy **(b)** Europe

Figure 3: In case of italian devaluation policy, Italy experiences a wealthy period reaching the total occupation and an higher average GDP per capita.

firms in the same way as it impacts the italian one, since there is no preference for any group while evaluating where to consume.

This situation is however unreal since it is clear that citizens tend to buy from their own group for a variety of reasons, including more acessibility, income taxes, and policies promoting local good. For these reasons the parameter ***chauvinism*** is taken into account: it is set to 0.8 for each group, meaning that, neglecting everything else (like prices and quality), a person would buy 80% of the goods from its own group. This consumer choice turns out to affect expecially Italy in a dramatic way, as can be seen in figure 4 In this scenario all the italian firms go bankrupt after within thirty cycles. This happens because the hit in the italian consumption is taken for the larger part by italian firms which, paying less wages to the italian worker, make the demand to decrease, triggering a chain effect: indeed, cycle after cycle, more and more firms fail leading to a total impoverishment of the italian system.

At this point the parameter ***pmt.sensitivityToPrice*** is considered to account for the fact that people tend to buy where the price is cheaper. This variable is set to 1.5 and multiplies the difference between the prices of the two groups in the hyperbolic tangent[9] which gives the probability to buy in the respective group given a price difference, again neglecting other factors[10]. This new probability is then averaged, using some weights, to the chauvinism to give an aggregate probability. The weights are taken $[0.5, 0.5]$ meaning that the same importance is given to the willingness to buy in the respective group

[9]In order to control its smoothness.

[10]$(1 - tanh(x))/2$ is taken to have a probability between 0 and 1

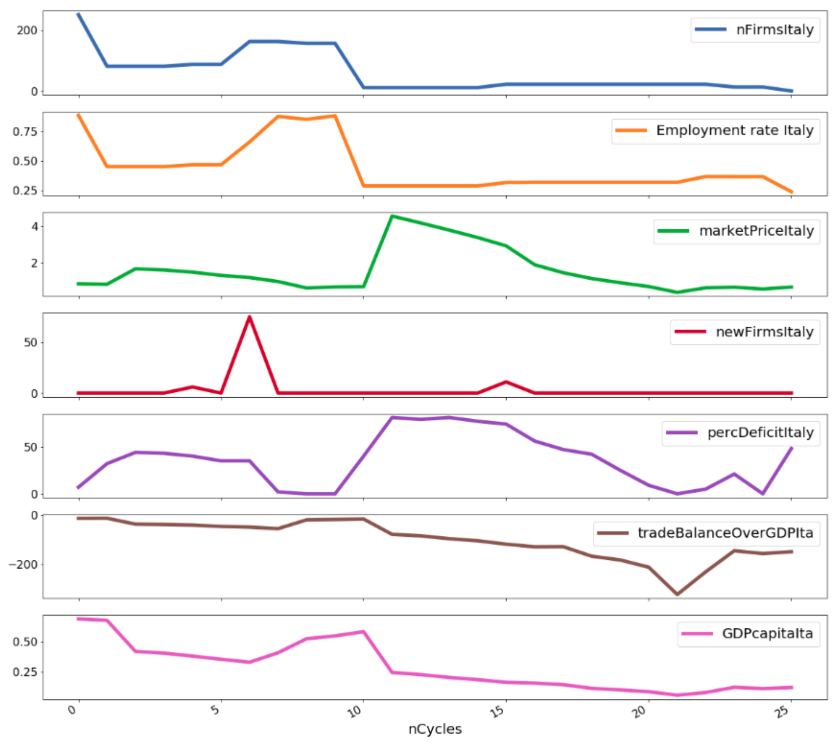
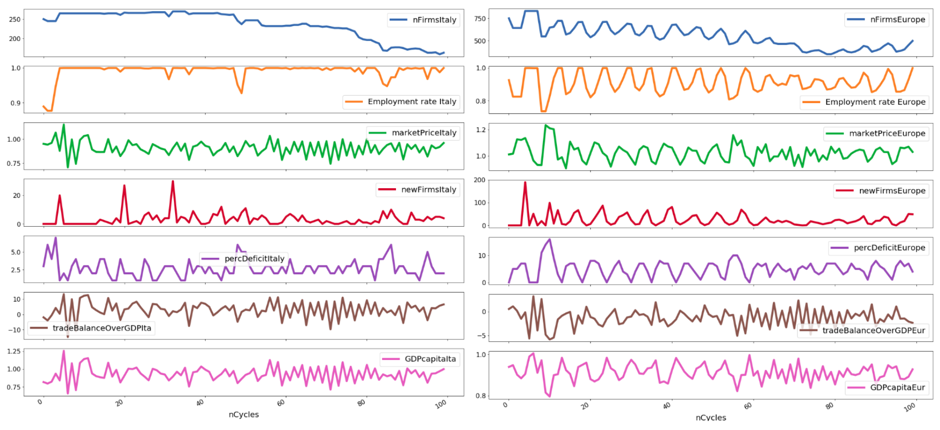


Figure 4: Setting the chauvinism to 0.8 makes the italian economy to collapse.

and to buy at a cheaper price. The simulation leads to the following: As it is

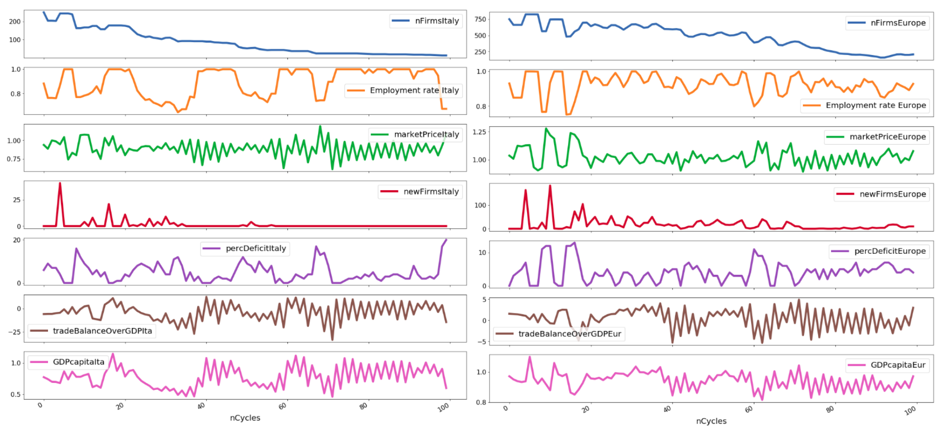


(a) Italy **(b)** Europe

Figure 5: Accounting for the buyer sensitivity to price makes the italian economy to last.

shown in figure 5 the italian market recovers when the sensitivity to price is accounted. The employement rate is higher than in europe and close to total occupation. The market price stays lower for Italy than for Europe trought the simulation but the greater volume of consumption given by the lower price balance out the lower price and makes Italian firms to prosper. In fact the trade balance stays positive for Italy (meaning that the export is greater than the import) almost through all the simulation by a significant amount, and that's what is needed for the italian economy to avoid the collapse.

Finally the parameter ***independence*** is varied. It was, in all the previous scenarios, set to 1 both for Italy and Europe. In an attempt to emulate, even in a one mrket good, the fact that Italy will need to buy outside a certain part of the total consume, e.g. the energy (that last year in Italy was imported for the 92%). The parameter was set to 0.85 meaning that no matter what's the inclination, in every case, on average, at least the 15% of the total consuption must be done outside for italian agents.



(a) Italy **(b)** Europe

Figure 6: In these runs the ***independence*** parameter is varied in order to account for the imports a country as Italy is forced to face.

In this case, as it can be seen in figure6, Italy starts to struggle, the employement rate drops and is less than the european one. Moreover, the firms are less stable than before. Since some import is now forced, the trade balance is close to zero so, the cheaper prices are not balanced anymore by the grownth in volume.

3.3 Italian policies after the separation

To enhance its status Italy could implement some policies to increase the percentage of consume within the group. In the model this traslates in a

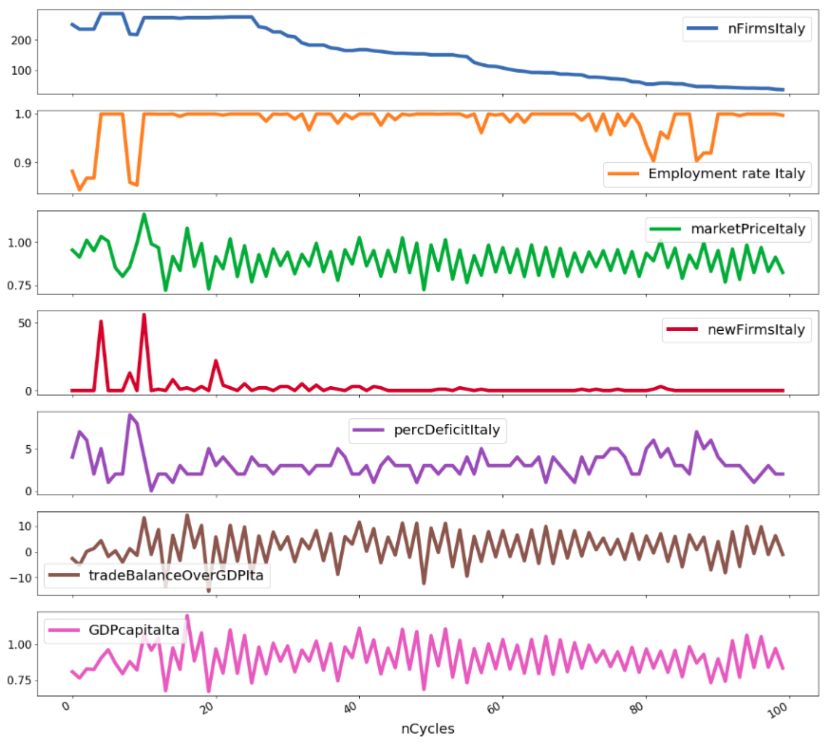


Figure 7: Increasing italian chauvinism results in a general improvement of the italia economy.

raise of the italian variable **chauvinism** to 0.9 while the european one is left unchanged.

In this final case Italy come back to having a positive trade balance and so a higher volume leading to a good situation, while Europe is stable as well as in the other cases. The italian unemployment rate oscillates around 3% and is less than europe one that oscillates around 8%.

4 Conclusions

The main goal of this project is to reproduce the basic properties underlying a macroeconomic system, such as the European Single market, and to evaluate the impact that the withdrawal of a consistent portion of this market (Italy has been considered in this case) could have with respect to the whole system. These have been achieved using a Multi Agent System, whose agents are households, firms and public administrations, and whose interactions are governed by some basic rules of a macroeconomic system.

Even though very essential rules were consideres, results show the emergence of a collective dynamics in terms of some macroeconomic indicators, such as GDP, market price, unemployment rate and more, which seems very plausible from an economic point of view.

The simulation shows stability under a wide range of parameters, representing different scenarios.

At first, the whole system before the separation is taken into account and the general stability of the two entangled markets is shown. Then, after the detachment of one of the two market from the other, a monetary devaluation is implemented. In this case, two main factors related to the consumer's behaviour are considered: the tendency to buy in his own country and his sensitivity to price. Both this factors have evident impact on the economy. While considering only the former may cause Italian economy to collapse, taking into account also the latter, allows for a stable equilibrium between the to markets. Finally, a parameter related to the need of buying a certain good (i.e. energy) outside Italy is considered. In this case, this characteriscic seems to afflict Italian economy and clearly shows how its wealth strongly depends on the trade balance with external markets.

As already mentioned, the model implements some basic features of a much more complex economic scenario. However our work demonstrates that using a Multi Agent Systems approach could extend the investigation upon unexpected outcomes, conferring researchers a wider point of view: is this the case of the present work in which, almost for every scenario, emerges that the success of one side results in damage for the other one.

A calcConsumption

```
 def calcConsumption(self):

      if self.is_employed == True:
         if self.hasInvested == False:
            p = random.uniform(0, 1)
            if p < pmt.pInvestHousehold:
               self.hasInvested = True
               self.debt = pmt.repaymentRate
               self.nInstallmentsleft = pmt.
nInstallmentsHouseholds
               self.addInvestmentToDemand = True
      if self.is_employed == True:
         self.wage = pmt.wage[self.my_group]
         if self.hasInvested == True:
            self.wage -= self.debt
            self.nInstallmentsleft -= 1
            if self.nInstallmentsleft == 0:
               self.hasInvested = False
               self.debt = 0
         self.consumption = pmt.a + pmt.b*self.wage + random.
normalvariate(0, 0.1)
         if self.consumption > self.wage:
            self.consumption = self.wage
            if self.addInvestmentToDemand == True:
               self.consumption += pmt.
demandForInvestmentHouseholds
               self.addInvestmentToDemand = False
      else:
         self.consumption = pmt.unempBenefit[self.my_group]
         self.debt = 0
         self.wage = 0
      return self.consumption
```

Listing 8: Function that computes the consumptions of a household.

B ProbConsOwn

```
def ProbConsOwn(self, priceItAndEur, nTotWorkers):
   totalWorkers = nTotWorkers[0] + nTotWorkers[1]
   not_my_group = 1 - 1 * self.my_group
   prob1Part1 = pmt.chauvinismWeight * pmt.chauvinism[self.
my_group]
   prob1Part2 = pmt.sensitivityToPriceWeight * (1 - np.tanh(pmt
.sensitivityToPrice*(priceItAndEur[self.my_group] -
priceItAndEur[not_my_group])))/2
```

```
    prob1 = (prob1Part1 + prob1Part2)/(pmt.chauvinismWeight +
   pmt.sensitivityToPriceWeight)
    prob2 = prob1 * (nTotWorkers[self.my_group]/totalWorkers)*(
   totalWorkers/(2*prob1*nTotWorkers[self.my_group] - prob1*
   totalWorkers + totalWorkers - nTotWorkers[self.my_group]))
    pOwnGr = prob2 * pmt.Indipendence[self.my_group]
    return pOwnGr
```

Listing 9: Function that return the probability to consume within own group.

C planProduction

```
def planProduction(self, previousDemand, actualDemand):
    if self.production == 0:
        self.deltaWorkforce = 0
    else:
        plannedProduction = self.production * (float(actualDemand)
   / previousDemand)
        plannedWorkforce = round(plannedProduction / self.
   prodPerWorker)
        self.deltaWorkforce = plannedWorkforce - self.nWorkers
    return self.deltaWorkforce
```

Listing 10: Function that plane the production of a firm.

D calcDemand

```
def calcDemand(self):
    demandItaly = 0
    demandEurope = 0
    self.importExportIta[0] = 0
    self.importExportIta[1] = 0
    for key in self.householdsDict:
        for worker in self.householdsDict[key]:
            c = askAgent(worker, Household.calcConsumption)
            p = askAgent(worker, Household.ProbConsOwn, self.
   marketPrice, self.nTotWorkers)
            group = askAgent(worker, Household.getMyGroup)
            if group == 0:
                if random.uniform(0,1) < p:
                    demandItaly += c
                else:
                    demandEurope += c
                    self.importExportIta[0] += c
            if group == 1:
                if random.uniform(0,1) < p:
                    demandEurope += c
                else:
```

```
            demandItaly += c
            self.importExportIta[1] += c

    for firm in self.firmsDict.values():
      c = askAgent(firm, Firm.calcConsumption)
      p = askAgent(firm, Firm.ProbConsOwn, self.marketPrice,
    self.nTotWorkers)
      group = askAgent(firm, Firm.getMyGroup)
      if group == 0:
        if random.uniform(0,1) < p:
          demandItaly += c
        else:
          demandEurope += c
          self.importExportIta[0] += c
      if group == 1:
        if random.uniform(0,1) < p:
          demandEurope += c
        else:
          demandItaly += c
          self.importExportIta[1] += c

    demand = [demandItaly, demandEurope]
    return demand
```

Listing 11: Function that computes the demands and the trade balance.

E checkFirmsHealth

```
  def checkFirmsHealth(self, Idx_group):
    idxToDel = []
    newFirms = []
    for key, aFirm in {key:self.firmsDict[key] for key in list(
    self.firmsDict.keys()) if self.firmsDict[key].my_group is
    Idx_group}.items():
      if askAgent(aFirm, Firm.checkHealth) == -1:
        idxToDel.append(key)
      if askAgent(aFirm, Firm.checkHealth) == 1:
        newFirms.append(key)
    return (idxToDel, newFirms)
```

Listing 12: Function that checks the firm health returning the bankrupting condition or the investment condition.

F removeFirms

```
  def removeFirms(self, idxToDel, Idx_group):
    self.firmsDict = {key:self.firmsDict[key] for key in list(
    self.firmsDict.keys()) if key not in idxToDel}
```

```
 self.nTotFirms[Idx_group] = len(list({key:self.firmsDict[key
] for key in list(self.firmsDict.keys()) if self.firmsDict[
key].my_group is Idx_group}))
 print()

 for key in idxToDel:
   for worker in self.householdsDict[key]:
     askAgent(worker, Household.setEmploymentStatus, False)
     askAgent(worker, Household.setInvestmentStatus, False)
     self.householdsDict[Idx_group].append(worker)
     self.nUnemployed[Idx_group] += 1
     self.nTotWorkers[Idx_group] -= 1
   del[self.householdsDict[key]]
```

Listing 13: Function responsible for the removal of a bankrupting firm.

G addFirms

```
def addFirms(self, newFirms, Idx_group):
  self.nNewFirms[Idx_group] = 0
  for key in newFirms:
    nNewWorkers = self.nu[Idx_group] + random.randint(-10, 10)

    if nNewWorkers <= self.nUnemployed[Idx_group]:
      newFirm = Firm(nWorkers = nNewWorkers, my_group =
Idx_group)
      self.nNewFirms[Idx_group] += 1
      self.idFirm += 1
      self.firmsDict[self.idFirm] = newFirm
      self.nTotFirms[Idx_group] += 1

      self.nTotWorkers[Idx_group] += nNewWorkers
      self.householdsDict[self.idFirm] = self.householdsDict[
Idx_group][:nNewWorkers]
      for newWorker in self.householdsDict[self.idFirm]:
        askAgent(newWorker, Household.setEmploymentStatus,
True)
        self.nUnemployed[Idx_group] -= 1
      self.householdsDict[Idx_group] = self.householdsDict[
Idx_group][nNewWorkers:]
      askAgent(self.firmsDict[key], Firm.setInvestmentStatus,
nNewWorkers)
```

Listing 14: Function responsible of the integration of a new firm in the market.

H planProductionAndHireFire

```
def planProductionAndHireFire(self, Idx_group):
 for key, aFirm in {key:self.firmsDict[key] for key in list(
self.firmsDict.keys()) if self.firmsDict[key].my_group is
Idx_group}.items():
   extraWorkforce = aFirm.planProduction(previousDemand=self.
demand[Idx_group][self.t - 1], actualDemand=self.demand[
Idx_group][self.t])

   if extraWorkforce > 0:
     for i in range(extraWorkforce):
       if self.nUnemployed[Idx_group] > 0:
         newWorker = self.householdsDict[Idx_group][0]
         askAgent(newWorker, Household.setEmploymentStatus,
True)
         self.householdsDict[key].append(newWorker)
         del(self.householdsDict[Idx_group][0])
         askAgent(aFirm, Firm.updateNumWorkers, 1)
         self.nTotWorkers[Idx_group] += 1
         self.nUnemployed[Idx_group] -= 1

   if extraWorkforce < 0:
     for i in range(extraWorkforce):
       newUnemployed = self.householdsDict[key][0]
       askAgent(newUnemployed, Household.setEmploymentStatus,
 False)
       askAgent(newUnemployed, Household.setInvestmentStatus,
 False)
       self.householdsDict[Idx_group].append(newUnemployed)
       del(self.householdsDict[key][0])
       askAgent(aFirm, Firm.updateNumWorkers, -1)
       self.nTotWorkers[Idx_group] -= 1
       self.nUnemployed[Idx_group] += 1
```

Listing 15: planProductionAndHireFire takes into account the current production and the current profit in order to establish the following time step production through hirings/firings.

I controlMaasForItalyPA

```
 def controlMaasForItalyPA(self):
 if self.t > 0 and self.percDeficit[0] > pmt.controlPercMaas:
   self.counterIta += 1
   classicPurchasings = askAgent(self.PADict[0], PubAdm.
calcConsumption)
   self.PADict[0].purchasings = classicPurchasings - pmt.
percPurchasingsPA*self.counterIta*classicPurchasings
   demand = self.PADict[0].purchasings
 else:
   if self.counterIta > 0:
        self.counterIta -= 1
      classicPurchasings = askAgent(self.PADict[0], PubAdm.
calcConsumption)
      self.PADict[0].purchasings = classicPurchasings - pmt.
percPurchasingsPA*self.counterIta*classicPurchasings
      demand = self.PADict[0].purchasings
  return demand
```

Listing 16: Function that adjust the purchasings for Italy in order to respect the limitations.

J Average observed quantities

Average observed quantities per scenario

		nFirms	rateEmp	GDPcapita	deficit/GDP	tradeBalance/GDP
BS	IT	220.02±17.21	0.95±0.04	0.96±0.03	4.42±2.41	-0.02±1.49
	EU	659.59±41.24	0.95±0.04	0.96±0.02	4.38±2.44	-0.01±0.50
DEV	IT	285.00±0.00	0.99±0.02	1.03±0.08	2.25±0.67	8.12±2.91
	EU	491.48±86.19	0.88±0.11	0.88±0.05	7.80±6.61	-3.33±1.87
PS	IT	154.39±61.96	0.99±1.06	0.92±0.10	2.99±1.06	2.50±5.50
	EU	377.06±53.96	5.18±1.98	0.91±0.04	5.17±1.98	-1.12±1.98
IN	IT	42.18±46.60	0.79±0.28	0.81±0.16	4.52±4.61	-3.06±12.54
	EU	520.87±83.15	0.94±0.05	0.94±0.05	4.34±2.81	0.10±2.41
IT-CH	IT	198.34±44.95	0.99±0.02	0.93±0.19	3.01±1.28	2.36±8.74
	EU	290.24±115.63	0.89±0.08	0.90±0.07	6.42±5.28	-1.68±4.22

Table 2: Average observed quantities per scenario: BS-before separation, DEV-devaluation, PS-sensitivity to price, IN-indipendence, IT-CH- italian chauvinism.

References

[1] Alberto Bagnai, Brigitte Granville, and Christian A. Mongeau Ospina. Withdrawal of Italy from the euro area: Stochastic simulations of a structural macroeconometric model. *Economic Modelling*, 64(April):524–538, 2017.

[2] Charlotte Bruun. Agent-Based Keynesian Economics: Simulating a Monetary Production System Bottom-Up. Technical report, 1999.

[3] CK Chan and Ken Steiglitz. An agent-based model of a minimal economy. *Princeton University*, (1994):1–33, 2008.

[4] Aristeidis Samitas, Stathis Polyzos, and Costas Siriopoulos. Brexit and financial stability: An agent-based simulation. *Economic Modelling*, (September):1–12, 2017.

[5] Ken Steiglitz, Michael L Honig, and Leonard M Cohen. A computational market model based on individual action. Technical report, 1993.