Università di Torino

Scuola di Scienza della natura
Dipartimento di Fisica
Corso di laurea magistrale in fisica dei sistemi complessi

Brownian and more complex agents to explain markets behavior

Marcello Nieddu

Supervisor: Prof. Pietro Terna
Examiner: Prof. Marco Maggiora

Academic year: 2016/2017

Contents

Chapter 1

Introduction

Our work is inspired by a real world problem: how can local farmers act or what kind of instruments they can use to survive in a global economy? To answer this question, one has to understand market mechanisms that regulate the economic activity. Orthodox approach to economics makes use of aggregated equation based models to explain those mechanisms. However, as explained subsequently, to write down solvable model, this approach hides the complexity inherent to markets.

To study market mechanism, we use instead agent based modeling (ABM), a new kind of modeling approach, that is not tied to analytical tractability. We use a physical theory assisted method (Brownian agent based modeling) to build more market models with increasing complexity.

In chapter 2 we describe the real world problem that have inspired the thesis. In chapter 3 we describe complex systems with particular attention to those aspects that compromise analytical tractability. In the first section we give a general description, while in the second one we examine the consequences of applying complex system approach to economics.

In chapter 4 we present ABM. The first section is dedicated to ABM description. In the second one we talk about some consequences of adopting ABM as our modeling paradigm. Finally, in the third section we talk about the drawbacks in using ABM.

In chapter 5 we describe Brownian agents. In the first section we explain why we have chosen BA in our work. In the second one we give the formal definition and in the third we describe a powerful tool of BA: the adaptive landscape, a spatiotemporal field of communication used to model indirect interaction between agents.

In chapter 6 we describe the software that we have used to build our models: Swarm Like Agent Protocol in Python (SLAPP).

Chapter 7 describes the market vision of F. Hayek, which fit well with both complex system and ABM approach.

In the three subsequent chapters we describe the three models we have built. All the three chapters are divided in three sections: in the first we give the model description; in the second we explain and report the code we have written; in last section we show and comment the results.

Finally, in the appendix, we describe the basic features of Brownian motion.

Chapter 2

Starting with an asparagus

The thesis is inspired by a business idea, called "Appiu" developed by Federicco Puddu and Sara Defraia of Dipartimento di Agraria dell' Università di Sassari.

Appiu ("asparagus" in Sardinian) is an on-line marketplace for agricultural products. Its scope is to increase the volume and to improve the reliability of direct selling between local farmers and consumers.

From the production place, agricultural goods have to flow through chains of exchanges before reaching the final consumer. In a simplified description, food supply chains are composed by three types of actors:

- farmers: they produce agricultural good.
- distribution actors (wholesalers, retailers): they buy from farmers and sell to consumers.
- consumers: they buy the final product

The distribution element has a crucial role in the process: it collects and elaborates production and consumption information, making possible the flow of food. But it is not efficient and often chains are too long, products can travel several kilometers to end near the starting point. It is composed by many individuals each acting according to its own interests and beliefs and under its own constrains. Viewed as a whole, it is not worried about environmental and social impact of trading choices.

The authors of Appiu claim that this situation disadvantages local farmer sardinian firms. Small production implies higher unitary costs. Small firms are also compelled to sell to distributors at prices given by international concurrence. This undermines the survival of local production and biodiversity.

Direct selling is a possible and practiced alternative for small farmers. But it has serious problems: it is time and space consuming. Farmers are constrained to spend time for selling.

The authors' idea is to create a virtual space where producers can establish their offer and propose them to consumers through an on-line platform, bypassing the big distribution.

They think that it can simplify sale for producers. Consumers are becoming more sensible to environmental problems, they express the desire to buy local products. Recent studies sustain that short food chain are more efficient and can grow thanks to trust development.

Trust development and self organization are the two ideas on which APPIU is based. It is possible to build a model (or more models) that can tell something about the impact of this idea?

Chapter 3

Complex systems

3.1 Complex systems

Complex systems are everywhere in our world, from our societies to the atmosphere of the earth, from the Internet to the cells that constitute our bodies. The food supply chains system discussed in the previous chapter, is another example of complex system. Due to their transverse nature, the study of complex systems involves many disciplines, each contributing with its own scope, theatrical definitions and methodology. Then, there is not a precise and commonly accepted definition of complex system. Nevertheless, there are common features in almost all definitions that allow to understand the term.

We want to start with a nice definition, given in [14], to list these common features:

> By complex system, it is meant a system comprised of a (usually large) number of (usually strongly) interacting entities, processes, or agents, the understanding of which requires the development, or the use of, new scientific tools, nonlinear models, out of equilibrium descriptions and computer simulations.

The strong interaction between elements is one of the main characteristic of complex systems. From statistical physics we know that in presence of strong interaction we can not decouple the descriptions of the single parts of a system to look at the mean individual behavior. Further, we know from dynamical systems theory that strong interactions, often translated as non-linear coupling terms between the equations that constitute the system, may lead to chaotic behavior. Chaotic systems are extremely sensible to initial conditions, and their motion is often non stationary and aperiodic. Then, the dynamic behavior of the elements of a complex system is strongly path dependent. This is why authors, in the above definition, speak about the need of non-linear models and out of equilibrium descriptions.

We note also that in the definition given, there are not restrictions in the definition of the entities: they may be simple, or they may be composed by others entities. From this reasoning we can speak of other two important features of complex systems. First, interactions between simple entities are sufficient to give rise to complex behaviors. Second, since entities of a complex system

may be complex systems in their turn, there are more level of structures, and interactions may occur among same level entities or between different level ones.

Another important aspect of complex systems that does not appear in the above definition is the interaction between entities and the environment where they live or operates. In classical theoretical approaches, often the environment is conceived as a passive or quasi-static element. In complex systems, the elements are able to change, although locally, their environment which in turn influences their future actions. A clear example is the interaction between plants and the earth atmosphere: agents and environment co-evolve.

The features exposed suggest that it is difficult to treat complex systems with analytical models. This is why in the definition given at the start of the chapter, Richards et al. speak about the need of computer simulations. As we will explain in chapter 4, computer is not used as an approximation tool; complex systems compel us to rethink our modeling approaches, and their study, together with computer use and diffusion have given the birth to Agent Based Modeling approach.

Since the thesis deals with socioeconomic systems, we want to summarize this section quoting Borril and Tesfatsion [2]:

> Nevertheless, it is extremely difficult to capture physical, institutional, and behavioral aspects of social systems with empirical fidelity and still retain analytical tractability. Entities in social systems are neither infinitesimally small nor infinitely many, nor are their identities or behaviors necessarily indistinguishable from one another. Common simplifications, such as assumed homogeneous behaviors or the existence of single representative agents, are thus problematic. Moreover, the social sciences cannot separate observers from "the real world out there." Rather, social scientists must consider multiple observers in a continual co-evolving interaction with each other and with their environment.

3.2 Complexity in economics

From early 1990s, economists are trying to use complex systems approach to economy. However, there is not agreement on how this approach can be used and what are the consequences to view economy as a complex system ([8]). For our purposes we can identify two different conceptions. On one side it is believed that complex systems approach can fit in the standard accepted theory, and one has to use it only in those cases where standard analysis fails. The supporter of the other side claims that the two approaches to economics are so different that is it not possible to merge them.

Here we follow the line of thought proposed by A. Kirman ([11]), an unorthodox economist. To understand Kirman's reasoning we have to talk a little about economic theory.

Economic theory has developed many market models (abstractions) to understand economic activities of human societies. These models are grounded on some assumptions about human behavior and markets. An economic system is thought as a set of people that buy or sell something from other people. Each agent is characterized by a function that measures utility and one that

measures costs associated to the purchase (or sale) of a given quantity of good, both commonly expressed in money. Everyone tries to maximize the net utility compatibly with a some kind of constraint (for example a budget constraint). It is believed that when none has more market power than others and there are not institutions (perfect concurrence), this system reaches an efficient equilibrium state. This means that exchanges go on until all the exchanges that rise up the sum of net benefits are performed and doing another exchange implies a reduction of this sum.

To better understand how market equilibrium works we consider a system composed only by buyers and sellers where only one type of good is exchanged. We assume that goods sold by different agents are similar in quality and differ only by their price. We assume also that every agent can potentially interact with, and is perfectly informed about all the others. In this situation none is able to change the price proposed or accepted by the others. To make clear this point, suppose that one of the sellers decide to rise up the price with respect that proposed by the others. She will not sell because consumers, being perfectly informed about all agents, will buy elsewhere for a lower price; then she is compelled to decrease the price if she wants sell her goods. In a similar way, if a consumer pretends to buy at a price lower than that accepted by the others she will remain unsatisfied; none seller will change his price to satisfy only one consumer, while all the others accept the proposed one. As said above, this situation is called by economist perfect concurrence. Agents in perfect concurrence are considered 'price takers', because they can not change the market price, and they decide and act considering its fixed and given from the outside.

We denote with q, $U(q)$ and $C(q)$ respectively the quantity of good considered, the utility function and the cost function. Deriving the utility function $U(q)$ with respect to the quantity q we obtain the marginal utility function, which measures in unit of utility per unit of quantity how much one benefits from buying another one unit of product. It is assumed a non increasing function of its arguments, to capture the intuitive fact that more one has, less she wants. Marginal cost function is defined in a similar way, and it is assumed a non-decreasing function of the quantity. For each agent maximum net utility $U(q) - C(q)$ occurs when marginal utility and marginal cost are equal, i.e. when the derivative of the net utility is zero.

In figure 3.1 the red line represent the marginal benefit of consumers, it is called demand curve; the blue one stands for the marginal costs of sellers, and the horizontal dashed one represent the market price. The market price line is the marginal costs function of consumers and at the same time is the marginal benefit function of sellers.

The equilibrium corresponds to the intersection point between demand and supply curve, which is also the intersection point between marginal benefits and marginal costs curves for each category. The equilibrium condition allows one to determine the quantity exchanged given the demand and supply curve. This equilibrium quantity q_e is optimal in the sense that if in a given time a smaller quantity $q < q_e$ has been exchanged, both consumers and sellers can raise up their net benefit. But once the quantity q_e has been exchanged no more trade is favorable.

This however is a static situation and it is valid only for a given period; demand and supply can vary in time. Consider for example a demand increment, as in figure 3.2. It is believed that the market adapts very rapidly to the new

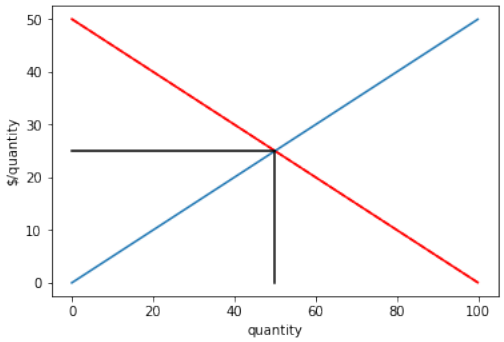


Figure 3.1: Demand-Supply graph

conditions and the system goes to the new equilibrium point, which is again the intersection point between demand and supply curve. If we consider successive increments, the system moves on a set of equilibrium points.

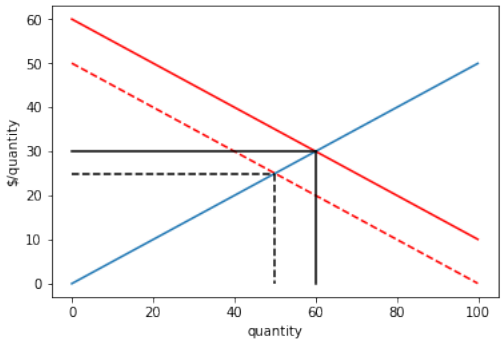


Figure 3.2: Demand variation

This is very similar to what happens in a thermodynamic reversible transformation: the system under consideration is in thermodynamic equilibrium in every instant of the transformation. If we change the temperature of a perfect gas, in every instant the atoms will be distributed according to the Maxwell-Boltzman distribution calculated for the actual value of temperature and volume. This is because deviation from the Maxwellian distribution are suppressed in a much smaller times than those in which temperature changes significantly.

Kirman claims that in economics a similar explanation to justify the equilibrium assumption does not exist:

> [It has been] argued that 'spontaneous order' would arise from the behavior of rational self-interested individuals. The conviction that this will happen has persisted, but the Achilles' heel of modern economic theory is that we have never been able to specify the mechanism that would engender such order.

He said that there have been several attempts to find a mechanism that drives an economy to its equilibrium. An important example of driving mechanism is

the Walrasian tattonnement process (from L. Walras), which we use as a specific example to talk about the problems of the others mechanisms proposed. Markets evolve as if there were an auctioneer that find the equilibrium price (those that maximize the overall utility) through successive try and errors. She chooses a price for a given product and asks agents the quantities that they want to buy or to sell to that price. If the demanded quantity and the supplied one are different, the auctioneer change the price to reduce this difference. She repeats the procedure until the demanded and supplied quantities are equal.

But Walras was not able to show that equilibrium is always reached in this way. An important problem of the tatonnement process is that it could need a prohibitive amount of information to find the equilibrium price vector starting from anyone out of equilibrium. The same problem is shared with others price adjustment processes. Another critical aspect of the tatonnement process is that he tries to explain a decentralized self-organization process but it is a centralized one, due to the presence of the auctioneer.

To what extent agents are informed? How they use the information they have and how this produces the outcomes observed? This is another point where the two approaches to economics, that of general equilibrium theory and that of complex systems, are incompatible. In self-organization processes agents use only local information, without reference to the global pattern that emerges. The relation between the global and the local level is not clear, and it is difficult to draw conclusions about individuals observing solely their aggregated actions. Instead, in the general equilibrium theory agents do not make errors, they have access to uncorrupted information about the whole system, they act each independently of the others and the global behavior reflects that of the mean individual.

The conditions of perfect competition, that of perfectly informed agents all exactly optimizing their net utility, do not match perfectly real situations and it is not believed they are really true. But allowing agents to do errors and to know only a part of the whole system does not guarantee that the final state will be the efficient one. Furthermore, it is more difficult to analyze the global behavior. Agent's actions depend on how she expects the others will behave.

To tackle this problem, economists have introduced the so called "rational expectations' hypothesis". It states that the expectations of the agents do not differ systematically from the predictions of the equilibrium model. Considering again the above example of a one product market, the rational expectations' hypothesis means that each agent take a price equal to the market price plus a noise term with zero mean, $P = P_{eq} + \hat{\epsilon}$. So global result is not so different by that of perfect competition.

But is unrealistic that people know a model of their behavior, believe in it, and behave as this predicts. People exchange beliefs not true information. Furthermore, sometimes people lie to earn from consequent mistakes of others. So the assumption of unbiased prediction too is very unrealistic. Rational expectations' hypothesis is justified only by the solvability of the model.

Environments is composed by other agents and this implies, following again Kirman [11]

> there are phenomena whose existence and verity is independent of those who contemplate them, but this is not true of economic phenomena. In an economy, self-realizing hypotheses are perfectly pos-

sible. We know, from theory, that if enough people come to believe that there is a causal relation between some phenomenon that is initially totally unrelated to the state of the economy and the economy itself, then such a relation can develop.

Chapter 4

Agent based model

4.1 A new kind of modeling approach

In the traditional top-down modeling approach the model that represents the system under study has to be analytically tractable; this request comes from the need to have numerical results from the model. To guarantee analytical tractability one (or more) of the following assumptions are often made: it is assumed that components are homogeneous (indistinguishable), that the interactions are negligible or weak and linear, that the system tends to an equilibrium or to a steady state. But, as said in [2]:

> [...] the classical idea that we can deduce solutions (or "future states") for systems a priori, purely from a study of their structural characteristics, is beginning to be overshadowed by the realization that many systems are computationally irreducible.

We have seen that complex systems have characteristics that undermine traditional theories and experiments. There is not a clear relationship between the components and the whole; elements are heterogeneous in characteristics and behavior; there is non-linear feedback (positive and negative) among the elements and between elements and their environment; there is not tendency to equilibrium, instead elements co-evolve with their environment and the final state is strongly path-dependent.

The diffusion of computers have relaxed the analytical tractability request. But its use to study complex systems is something more than solving a complicated equation or integration. It has contributed to the birth of a new practical and theoretical modeling paradigm, known as Agent Based Modeling.

Following Galan et al. [9]:

> Modelling is the process of building an abstraction of a system for a specific purpose. Thus, in essence, what distinguishes one modelling paradigm from another is precisely the way we construct that abstraction from the observed system. [...] agent-based modelling is a modelling paradigm with the defining characteristic that entities within the target system to be modelled and the interactions between them are explicitly and individually represented in the model (see

> Figure 1). This is in contrast to other models where some entities are represented via average properties or via single representative agents.

So, while traditional approaches focus on variables and aim to write down a system of equations of motion for those, with ABM the focus of abstraction activity has shifted on the entities that comprise the systems, and on how their characteristics, the interactions among them and with the environment give rise to the global behavior.

Agents are characterized by data structures and methods to change these data; an agent can interact with others agents, for example changing one of her variables in dependence of the result of an information exchange with another agent.

However, the notion of agent depends in some degree on the field of study considered. In informatics agents are intended in a very wide sense: any entity, object or even process could be a valid agent. Systems of this kind of agents are more frequently termed Multi Agent System (MAS). In the context of social sciences agents are human like entities, not processes. From now on we adopt this latter notion of agency.

The data associated with each agent may have different level of accessibility. An agent may have internal variables that other agents can not see or can not change directly. In a socioeconomic context, an example of this type of variables is the reservation price of an agent, which is the higher (lower) price accepted to buy (to sell) a unit of product. In an interaction between two agent the reservation prices remain hidden variables that influence the interaction course, but the interaction itself involves only declared prices. Only the owner of a given reservation price variable can change it, although the entity of the change could depend on the interaction outcomes.

Depending on the focus of the analysis of a system that one want carry on, agents may have a very complex internal structure. For example there are studies on the relation between the learning capability of economic agents and the outcome of strategic games ([5]). In this context agent are equipped with neural networks and the learning capability is studied in terms of the dynamics of the weights of a reinforcement learning process.

Interactions too are of quite different type. Interactions can be distinguished by the level of the entities involved; indeed, agents may interact with another agent or with a group of agents, that can be viewed as an agent of an upper level with respect to a person.

In many practical situation it is very difficult to develop an ABM model that has predictive power. So what are the aims of an ABM model?

Following Conte et al. [6]:

> Models can also be used to explain the behaviour or properties of target systems. Explanation is a highly controversial notion [93]. Consistent with the scientific literature on modelling, we hold a view of explaining as showing how things work [94]. When we build models for explanatory purposes, we try to make adequate representations of target systems, similar to the predictive case. However, because there is a difference in what we use the model for, different properties enable models to be adequate to their intended use. Unlike

> predictive uses, which primarily involve optimizing the models to make their output as accurate and precise as we need it to be, the explanatory use requires us to learn how the component parts of the target system give rise to the behaviour of the whole.

Instead of predicting future states, one tries to build an explanation of a global behavior of a system. This is done by looking at the outputs of simulations of an ABM and checking whether the searched emergent behavior has arisen, suggesting that some futures of agents or of interactions present in the model are partially responsible for the global behavior observed. As formulated by Epstein ([7]):

> situate an initial population of autonomous heterogeneous agents in a relevant special environment; allow them to interact according to simple local rules, and thereby generate or grow the macroscopic regularity from the bottom up.

But ABM reveals useful even when the model is not tight to a specific real system. We read from the "Manifesto of computational social science" [6]:

> There is an additional explanatory use of models, one that is even more remote from the predictive case. Sometimes we want to understand how hypothetical systems work. There is really no analogue to this in the prediction case because in that context we are interested in predicting the behaviour of actual target systems. Sometimes in the course of our trying to explain actual target systems, we make comparisons to hypothetical systems. Fisher famously said that if we want to explain the prevalence of two sex organisms, we should start by explaining what the biological world would be like if there were three sexes.

Our work deal with this latter case: we will analyze a hypothetical agricultural market. This choice is due to the difficulty to collect data about the real systems and, when present, to analyze them since they are almost always aggregated data.

4.2 A subtler view on ABM

We have seen that in ABM, entities of the target system are represented directly. We have also seen that to explain a phenomenon involving the target system one has to build it in a computer simulation.

But this is not an escamotage to evade the difficulty of write down an equation. The differences between the traditional equations based modeling approach and ABM can reveal some subtle aspect of modeling activity and models usage.

Borril and Tesfatsion ([2]) argue that there are (at least) two ways of conceiving modeling activity:

> In analytical modeling, as well as in computer modeling used as an approximation tool, systems are typically represented from a God's-Eye-View (GEV). The mathematician or programmer presides over

> the modeled world like some form of Laplace's demon, able in principle to discern the entire course of world events based on a complete understanding of the initial state of the world as well as its laws of motion. [...] ABM supports a LOV [Local Observer View] modeling approach in the sense that the "reality" of each ABM agent is confined to the network of agents within which it interacts.

Authors argue that ABM are a mix of constructive mathematics and classical modeling approach. It is out of the scope of the thesis to present constructive mathematics, here we describe only its relevant feature; to a more complete review we remand to [4].

Constructive mathematics differs from classical one mainly on what is intended to be a proof of a statement. Constructive mathematicians consider proved a statement if one can give a procedure to obtain that statement. Differently from classical math they do not consider proved a statement if its contrary is false, i.e. they refuse the law of excluded middle.

Constructive math uses intuitionistic logic which is based on a different interpretation of logical connectors and quantifiers (see figure 4.1), known as BHK interpretation (from Brower, Heyting and Kolmogorov).

∨ (or):	to prove $P \vee Q$ we must either have a proof of P or have a proof of Q.
∧ (and):	to prove $P \wedge Q$ we must have both a proof of P and a proof of Q.
⇒ (implies):	a proof of $P \rightarrow Q$ is an algorithm that converts any proof of P into a proof of Q.
¬ (not):	to prove $\neg P$ we must show that P implies $0 = 1$.
∃ (there exists):	to prove $\exists x P(x)$ we must construct an object x and prove that $P(x)$ holds.
∀ (for each/all):	a proof of $\forall x \in S\ P(x)$ is an algorithm that, applied to any object x and to the data proving that $x \in S$, proves that $P(x)$ holds.

Figure 4.1: BHK interpretation. From [4]

An interesting example, taken from the same reference [4], is that of the proposition

$$\forall x \in \mathbb{R}, (x = 0 \vee x \neq 0) \tag{4.1}$$

From a classical point of view this statement is true because of the law of the excluded middle, but is undecidable for constructive mathematician. For a real number suspected to be non-zero, she has to prove that $x \neq 0$. She has to give an algorithm that for every real number x gives a rational number r such that $0 < r < x$ (a rational number is needed because a computational procedure, being implemented on a computer can deal only with rational numbers). Since a computer deals only with discrete approximations every sufficient small real numbers will be evaluated as zero. Since we are not able to give this algorithm, the proposition (4.1) is undecidable.

In ABM, an agent can acquire information only constructively by changing her internal structure by means of her methods in response to information received by environment or by other agents. But an agent can represent a classical mathematician that uses GEV procedures to build her world representation. In this sense, following again Borril and Tesfatsion ([2]), we can state that:

> [...] ABM is an alternative and more appropriate form of mathematics for the social science. [...] ABM combines constructive and classical modeling approaches. As is true for real people, agents can only acquire new data about their world constructively, through interactions. Nevertheless, again like real people, ABM agents can have uncomputable beliefs about their world that influence their interactions.

So ABM well adapts to social science since it represents a social system as composed by agents, each with her own representation and theories about the world and the other agents. Theories are partially shared, giving raise to "objective" (in the sense that it is shared among individuals) knowledge or to standard accepted theories. Common knowledge and single agent's theory (both concerning agents' behavior) interact continually: an accepted theory on human social behavior may influence the actions and beliefs of each agent, that in turn, may lead to a new theory.

Borril and Tesfatsion go further and ask "Is ABM the right mathematics for physical sciences?" One can answer no, because physics studies that properties of matters and universe that do not depend on the historical path followed, nor can physical phenomena depend on physical theory. But to a deeper analysis the answer may be not so obvious. To this end we want to talk about Landauer's thought about physical laws and their relation with computation:

> Science, and physics most particularly, is expressed in terms of mathematics, i.e. in terms of rules for handling numbers. Information, numerical or otherwise, is not an abstraction, but is inevitably tied to a physical representation. The representation can be a hole in a punched card, an electron spin up or down, the state of a neuron, or a DNA configuration. There really is no software, in the strict sense of disembodied information, but only inactive and relatively static hardware. Thus, the handling of information is inevitably tied to the physical universe, its content and its laws. This is illustrated by the right-hand arrow of Fig. 2 . We have all been indoctrinated by the mathematicians and their sense of values. Given ϵ , $\exists N$, such that - - - - -. Now we come back and ask, "Does this arbitrarily large N , this implied unlimited sequence of infallible operations, really exist?"

What Landauer claim is that since executable mathematical algorithms are information handling processes they have to respect physical constraints of reality. In turn, physical laws have to be expressed in terms of executable mathematical algorithms. Thus, "we need a self-consistent formulation of the laws of physics" as shown in figure 4.2.

This vision suggest also another aspect of GEV modeling. It seems that classical modeling approach believe in a sort of platonic plane of reality from where pure laws, expressed in mathematical language dictate matter how to behave. But where are physical law physically situated?

> Mathematicians declared their independence of the real physical universe, about a century ago, and explained that they were really describing abstract objects and spaces. If some of these modelled

> real events, all right, but . . . . They ignored the question which we have emphasized. Mathematicians may be describing abstract spaces, but the actual information manipulation is still in the real physical universe.

Also, the acquisition of information, and then the measurement process, has to respect physical constraints. The belief that the value of a given physical observable is independent on the measurement procedure is in part justified if there exists a measurements process that does not change the observable value, avoiding dissipation. Landauer believed that since noise (or fluctuations) is always present, a measurement, and also a computation, can not avoid irreversible procedures, needed to store and preserve the information from degradation (due to diffusion processes which Landauer identifies with noise). Despite many noise-free reversible scheme of computation and measurement have been proposed, all of these are based on questionable (GEV) assumptions: typically to avoid fluctuations (noise) in the computational device it is proposed to increase its size or to slow down the process of computation. But, following again Landauer ([12])

> How many degrees of freedom, for example, can be kept together effectively, to constitute a computer? [...] It is important to understand that the [noise] immunity is bought at the expense of the size of the potentials involved, or - more or less equivalently - the size of the parts, and not at the expense of energy dissipation. In the real universe, with a limited selection of parts, arbitrary immunity may not be available.

If there is an energetic cost of acquiring information, we can not think anymore measurement as an operation that reveal the internal state of a system without influence. Borril and Tesfatsion quote the recent development of Relational Quantum Mechanics to say that "there are no observer-independent states". However, as we suggest, Landauer's argument, combined with complex system approach is sufficient to make the same conclusion.

4.3 ABM drawbacks

ABM are not tied to analytical tractability and assumption of homogeneity or of equilibrium state are not necessary. This makes models nearer reality. But it has also some drawbacks.

We have said that the purpose of an ABM is to explain a global behavior by building it in a bottom-up fashion. We can distinguish three phases in this process: in the first phase it is built an executable model, then it is run in a computer and finally the results are analyzed and interpreted. But it is possible that one makes some mistake in writing the program, and what she believes the computer is doing is different from what the computer is actually doing. We say that an error has occurred.

Errors are not only possible, according to Gilbert ([10]):

> You should assume that, no matter how carefully you have designed and built your simulation, it will contain bugs (code that does something different to what you wanted and expected).

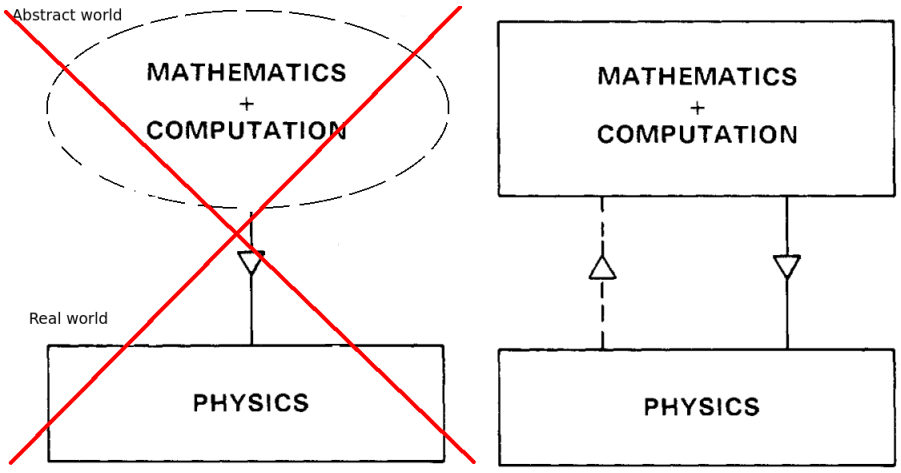


Figure 4.2: Landauer circuit. Taken from [12]

Since the model is often unsolvable, one has no clear idea of the output that a simulation will produce before running it. This makes more difficult to find errors that are influencing the global pattern. Further, errors depend explicitly on the modeler's intentions, so it is necessary to know its to verify the presence of errors.

Even if there are not errors, it is difficult to understand which of the futures of the model is actually producing the observed behavior. This is due to two different reasons. First, since ABMs represent more realistically complex systems, agents can have a lot of interrelated internal variables and a lot of possible actions. So determining what are the relevant model's features in the emergence of a global pattern is a hard task since there are a lot of possible choices.

Second, it is possible that a part of the model believed irrelevant is actually influencing the global pattern. Following Galan et al. ([9]) we say that an artifact has occurred. An example: space grid shape.

In order to deal with errors and artifacts it is common practice to verify and validate a model. Verifying means assuring the absence of errors. So a model is considered correct if it does not contain errors. Instead, a model is considered valid if its results are coherent with the purposes of the model.

Understanding when and how errors and artifacts may occur requires a more subtle comprehension of the ABM use. There is not a common accepted way to represent the study of a system through an ABM. Here we follow the representation proposed in [9], whose intent is not to represent faithfully the actual process followed by modelers, it is to give instruments to more easily detect errors and artifacts / to facilitate verification and validation.

They first divide the study of a system through ABM in three parts: the construction of the model, its execution on a computer, the analysis of the output produced. They further subdivide the construction part in different step, each associated with a particular role or person involved in the process. They identify four roles: the thematician, the modeler, the computer scientist and the programmer.

The thematician carry out the abstraction step. She has to identify the target

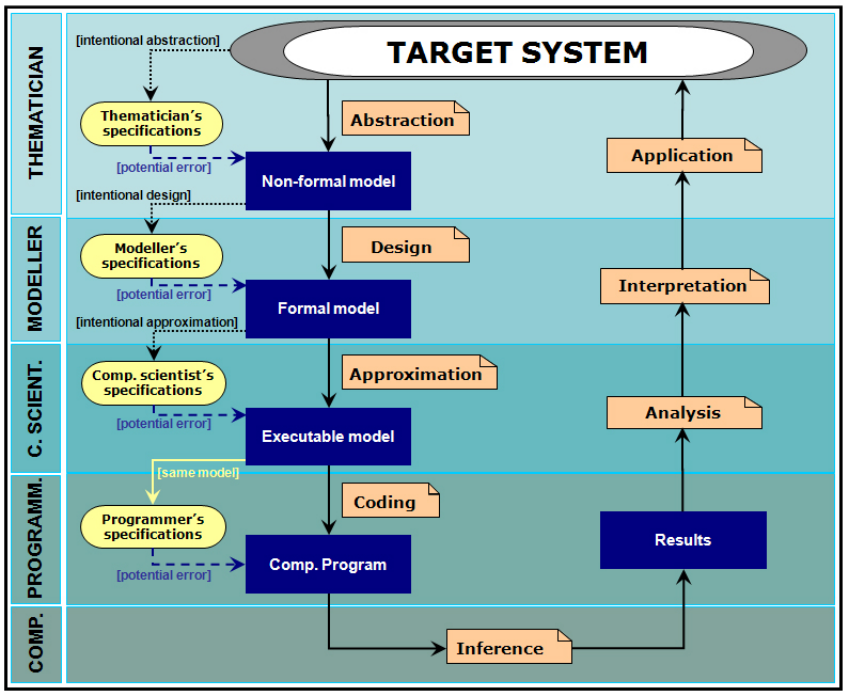


Figure 4.3: Modeling scheme, taken from [9]

system and to define the objectives of the modeling activity. She extracts the characteristics from the real system under study that she judges relevant for her purposes. Finally, she builds a non-formal model where the entities, the world where they live in and the causal relationships are defined. The expression 'non-formal model' means that it is expressed in a non-formal language.

The modeler translates the thematician model into a formal-model, expressed in mathematical language. The computer scientist converts the modeler's model into an executable one. Finally, the programmer translates the model of the previous step in a given programming language.

According to [9]

> One of the main motivations to distinguish between the modeller's role and the computer scientist's role is that, in the domain of agent-based social simulation, it is the description of the modeller's formal model what is usually found in academic papers, but the computer scientist's model what was used by the authors to produce the results in the paper. Most often the modeller's model (i.e. the one described in the paper) simply cannot be run in a computer; it is the (potentially faulty) implementation of the computer scientist's approximation to such a model which is really run by the computer. As an example, note that computer models described in scientific papers are most often expressed using equations in real arithmetic, whereas the models that actually run in computers almost invariably use floating-point arithmetic. Note also that we consider that the defining feature of a model is the particular input-output relation-

> ship it implies. Consequently, two different programs that provide the same input-output relationship would actually be two different representations of the same (executable) model, even though they may be written in different languages and for different operating systems.

In the abstraction step artifacts can not appear, but there may be errors.

Modeler introduce artifacts, since she has to make some accessory assumptions or implementation choices to write down a formal model (such as a particular shape of the lattice used to discretize agent's world). These assumptions may be relevant to the global behavior although more frequently are retained neutral. Errors are less probable.

Computer scientist too may introduce artifacts, because she has to build a formal executable model. She too has to make accessory assumptions.

Finally, the programmer can not introduce artifacts, only errors.

Authors suggest to modeler and computer's scientist role to:

- implement different models retaining the core assumptions of the previous step but varying accessory ones. This prevent artifacts.
- explore parameter space to be sure that results do not depend on the particular region chosen. This prevent artifacts.
- develop abstraction mathematically tractable to verify potential errors.

Chapter 5

Brownian Agents

5.1 A minimalistic approach to ABM: Brownian Agents

ABM allows to directly modeling the microscopic detail of a system without imposing homogeneity or equilibrium assumptions to facilitate or make possible formal analyses. But ABM are not fully independent on analytical tractability requirements.

Errors are very likely to occur in the model construction; if one has no idea of the possible outcomes, it is very unlikely she could find errors only reading again the code. One of the advises to deal with errors is to build additional abstractions of the abstract model that are analytically tractable.

Further, it is difficult to interpret simulations output without a theory that at least gives an intuitive picture of the possibles results. This is due in part to the intrinsic complexity of the target system. But the modeling activity too is responsible for these difficulties. Referring to the roles depicted in the previous section, we have seen that both the modeler and the computer scientist have to make some accessory assumptions to specify the model according to their tasks. They can introduce artifacts that make more difficult to understand which part of the model is actually influencing the global behavior.

So the opportunity to include in the model as much detail as possible is opposed by the need to have an intuitive picture of the behavior of the model. In Existing ABMs, agents and interaction rules are strongly dependent on the context of application. There is not a general rule that tells how much detail one has to retain in her model; all we can say is that it depends on the target system and on the purposes of the modeling exercise.

At the lowest level of complexity we find passive particle agents. They passively react to external stimuli, i.e. their response to a given stimulus is always the same. On the other extreme there are complex agents. They have learning capabilities, they can develop a representation of their world and the actions of the others agents, and they can use this representation to make decisions.

The right agents' complexity level to chose depends on the objectives and on the application context of the research. For example, if the context of interest is one where people use their rational capabilities, or in general where intelligent actions are performed, and if the purpose is to explore the influence of agents'

learning capabilities on the global level it is better to use complex agents.

We have seen that the components of a system do not need to be complex to display complex global behavior. Even when system components and interactions are complex, not all their futures are involved in the observed global pattern. This depends on the objectives of research. If, for example, we want to study the prices at which economic exchanges between people are performed we can model agents without considering their whole history of life. However, in the social context, the problem of identify what are the relevant features for a specific purpose remains open.

Our focus is not on agents' internal structure, but is on the self-organization properties of the system and on emergent behaviors. As thematicians, we identify the target system with a hypothetical agricultural market. We want answer to questions such as "Does the system converge to a state where the prices of the exchanges occurred are similar?", or "What are the conditions to observe a global shift of preference toward short food supply chain ?". We want to study how information is exchanged among agents, and how the flux of information influences exchanges dynamics.

So, instead of determining what are the relevant features, we assume that agents have simple internal structure. It may seem that this choice is a too rough approximation. We note however that although equipping agents with artificial neural network (ANN) to account learning capabilities seems a better approximation it remains to check how accurate is to assume that human brain can be modeled by an ANN.

Following Schweitzer [16], we adopt a minimalistic approach to ABM design. As modelers, we use Brownian agents (BA). As explained by the same author, to adopt a minimalistic approach means that:

- Instead of incorporating as much detail as possible, we want to consider only as much detail as is necessary to produce a certain emergent behavior. This rather "minimalistic" or reductionistic approach will, on the other hand, allow us in many cases to derive some analytical models for further theoretical investigation, and thus gives the chance to understand how emergent behavior occurs and what it depends on.
- To find the simplest "set of rules" for interaction dynamics, we will start from very simple models that purposefully stretch the analogies to physical systems. Similar to a construction set, we may then add, step-by-step, more complexity to the elements on the microlevel, and in this way gradually explore the extended dynamic possibilities of the whole model, while still having a certain theoretical background.

Brownian agents have a level of complexity which lies between those of reactive and complex agents. As ABM agents, they have internal degree of freedom and methods to change them after an interaction. Differently from particle agents, they can react to a given stimulus in more ways, and they can decide to start an interaction with other agents, or they can change their environment. But they have not a representation of the world, they do not act to maximize some utility function, and they have not explicitly learning capabilities.

The rules governing BA behavior are Langevin type equation of motion for their variables. According to these equations, the change in time of a given

degree of freedom is expressed as the sum of deterministic and stochastic influences. The deterministic part collect all the causal influences that affect the considered variable, for example the effect of interactions with other agents, or an eigendynamic effect such as a decay term. The stochastic term represents the sum of all the influences that operate in smaller time and length scale than those of the agent.

In this sense we are neglecting internal structure problem: we do not chose a complex structure to understand which are the relevant features. So we are not compelled to justify the use of that structure to represent human agents. We give agents a simple structure, and we allow them to act only through simple rules because we are not interested with neural activity nor with how people give structures to their knowledge. We are not interested with the single agent characteristics or actions. "Instead of specialized agents, the Brownian agent approach is based on a large number of 'identical' agents, and the focus is mainly on cooperative interaction instead of autonomous action."([16]).

Again from [16] we read:

> Observing self-organization depend on the specif level of description or on the focus of the observer. [...] self-organization theory is an aithetical theory. The particular level of perception for self-organization has been denoted mesoscopy[446]. It is different from a real microscopic perception level that focuses primarily on the smallest entities or elements, as well as from the macroscopic perception level that rather focuses on the system as a whole. Instead, mesoscopy focuses on elements complex enough to allow those interactions that eventually result in emergent properties or complexity on the macroscopic scale.

5.2 Definition

Brownian agents, as Brownian particle, move on a space, which we assume bidimensional from now on. Each one is characterized by a set of state variables u^k (where the index k runs over all state variables) that may be discrete or continuous, external or internal.

External variables are observable from the outside. The position and the velocity of the agent (assumed continuous) are examples of this variable type.

Internal variables are not directly observable (by others agents) and may be inferred only indirectly from the actions of the agent. An example used later is the reservation price of a consumer: she buys a product if the price is smaller (or is not too bigger) than her reservation price without communicating it. An internal discrete variable may be used also to model state dependent activity: the possible actions that an agent may pursue depend on the value assumed by the internal variable.

The state variables evolve according to a Langevin type equation:

$$\frac{du_i^k}{dt} = f_i^k + F_i^k \tag{5.1}$$

where the index i is referred to agents, f_i^k represents all the deterministic contributions to the variation of the state variable u_i^k and F_i^k represents the stochastic ones.

The deterministic term may results from the interaction with the others agents or from an external field or influence. So it may depend upon the state variables of all the other agents and from the parameters associated to the environment. The stochastic term represents the sum of all the influences that operate in smaller time and length scale than those of the agent.

To fix ideas we consider Brownian particles in a viscous fluid subjected to an external potential $U(\overrightarrow{r})$, the relevant state variables are the position vector $u(1)_i = \overrightarrow{r}_i$ and the velocity $u(2)_i = \overrightarrow{v}_i$. The evolution equations for the variables are:

$$\frac{d\overrightarrow{r_i}}{dt} = \overrightarrow{v_i} \tag{5.2}$$

$$\frac{d\overrightarrow{v_i}}{dt} = -\frac{\gamma}{m}\overrightarrow{v_i} - \frac{1}{m}\nabla U(\overrightarrow{r}_i) + \frac{\sqrt{2S}}{m}\hat{\xi}_i \tag{5.3}$$

where γ is the friction constant that depends on the viscous fluid, $\hat{\xi}$ is a Gaussian white noise process with zero mean and unit variance and the constant S represent its strength (see Appendix A for a more detailed exposition).

For the first equation we have only a deterministic term $f_i^1 = \overrightarrow{v}_i$ while $F_i^1 = 0$. In the second equation we have both the stochastic term $F_i^2 = \frac{\sqrt{2S}}{m}\hat{\xi}_i$ and the deterministic one $f_i^1 = -\frac{\gamma}{m}\overrightarrow{v_i} - \frac{1}{m}\nabla U(\overrightarrow{r}_i)$. Varying the strength of the noise S one observes behaviors of agents ranging from stochastic to deterministic, depending on the relative magnitude of stochastic and deterministic terms.

If we consider a harmonic potential with elastic constant k_{el} we can express the deterministic term as

$$f_i^1 = -\frac{\gamma}{m}\overrightarrow{v_i} - \frac{k_{el}}{m}\overrightarrow{r}_i \tag{5.4}$$

In this example the parameters associated with the environment that affect the deterministic term are the friction constant γ and the elastic constant k_{el}. More generally we collect all the environment parameters in a vector $\overrightarrow{\sigma}$ and we state that the deterministic term of each variable equation is a function of this vector $f_i^{(k)} = f_i^{(k)}(\overrightarrow{\sigma})$.

The deterministic term captures also interactions between agents. If particles in the above example are electrically charged, we have to consider an additional term in eq.5.4

$$F_i^{int} = \frac{1}{4\pi\epsilon}\sum_j \frac{q_i q_j}{|\overrightarrow{r_i} - \overrightarrow{r_j}|^3}(\overrightarrow{r_i} - \overrightarrow{r_j}) \tag{5.5}$$

where ϵ is the dielectric constant of the material (supposed homogeneous for simplicity) where the particles are immersed.

From this expression we can see that the deterministic contribution to the variation of each particle variable depends on all the position vectors of the other particles and on their charges. More generally for Brownian agents that have additional degrees of freedom, indicating all the variables vectors $\overrightarrow{u_i}$ of all the agents with $\{\overrightarrow{u}\}$, we write $f(k)_i = f(k)_i(\{\overrightarrow{u}\}, \overrightarrow{\sigma}, t)$ to account for the fact that the evolution of a given variable of each agent may be influenced by the other variables of the same agent, by all the variables of all the other agents, and by the environment parameters.

We have also inserted an explicit time dependence to account for environmental changes, not captured by the parameters vector σ, that affect agents' actions but are independent of them. An example is the interchange between day and night.

To build a Brownian ABM, first we have to declare all the variables $\overrightarrow{u_i}$ for each agent; then we have to specify the equations regulating variables evolution, and finally we have to specify the external parameters $\overrightarrow{\sigma}$ and other possible dynamical changes in the environment.

5.3 Interaction via an Adaptive Landscape

We have seen that the deterministic term in eq.(5.1) can be used to model interactions between agents, and we have shown how to write down the term in the case of interactions through an interaction potential. However, BABM includes also interaction based on 'if..then' scheme or other type of interactions; some examples are shown in [16]. In the same reference it is proposed a generalization of the interaction potential idea that constitutes the real power of BA approach.

Let us consider again the example of electrically charged Brownian particles. The force in eq.(5.5) experienced by the i-th particle can be expressed as the gradient of the total interaction potential as

$$\overrightarrow{F_i}^{int} = -\nabla_i U_{tot}^{int} \tag{5.6}$$

where $\nabla_i = (\frac{\partial}{\partial x_i}, \frac{\partial}{\partial y_i})$ and the total interaction potential is given by

$$U_{tot}^{int} = \frac{1}{2}\sum_{i,j}\frac{1}{4\pi\epsilon}\frac{q_i q_j}{|\overrightarrow{r_i} - \overrightarrow{r_j}|} \tag{5.7}$$

The interaction potential depend on the positions and charges of all the particles. In this system of electric charges, every charge contribute to build the potential that in turn influence every particle.

This scheme can be generalized to agents' variables different from the electric charge. One can assume that agents variables give rise to a generalized interaction potential V, which depends on the variables of all the agents. Given a variable $u_i^{(k)}$, the deterministic term may be expressed as the gradient of the interaction potential with respect to that variable

$$f_i^{(k)} = -\frac{\partial V(\{\overrightarrow{u}\}, \overrightarrow{\sigma}, t)}{\partial u_i^{(k)}} \tag{5.8}$$

However, as pointed out by Schweitzer ([16]), this approach is valid only for system where $\frac{\partial f_i^{(k)}}{\partial u_j^{(k)}} = \frac{\partial f_j^{(k)}}{\partial u_i^{(k)}}$. He introduces a generalization of this approach: agents interact through an *adaptive landscape*. To explain this concept we assume that agents are characterized by only one discrete variable α, with n possible values, in addition to position and velocity; we assume that this variable represents different states of activity of agents.

We associate a number for each value of α to each point in the space. An agent at a particular position (x, y) contribute or can modify only the number that corresponds to the value of its variable α_i. In this way, all the agents give

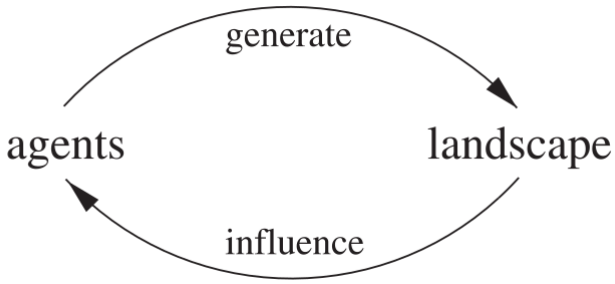


Figure 5.1: Feedback between agents and adaptive landscape. Image taken from [16].

rise to a spatial structure of numbers $h_\alpha(\overrightarrow{r}, t)$ that changes in time; Schweitzer call it a "multicomponent spatio-temporal scalar field" or *adaptive landscape*.

The equations of motion in this example are

$$\frac{d\overrightarrow{r_i}}{dt} = \overrightarrow{v_i} \tag{5.9}$$

$$\frac{d\overrightarrow{v_i}}{dt} = -\frac{\gamma}{m}\overrightarrow{v_i} - \frac{1}{m}\nabla h_{\alpha_i}(\overrightarrow{r}_i, t) + \frac{\sqrt{2S}}{m}\hat{\xi}_i \tag{5.10}$$

Every agent feel a force proportional to the spatial gradient of the field. The field is build up by agents but it influences the agents, so there is a feedback between agents behavior and field dynamic, as shown in figure 5.1. This justify the use of the adjective "adaptive".

> [...] one can imagine the adaptive landscape as a medium for indirect communication among the agents. This means that each action of the agents may generate certain information (in terms of changes of the adaptive landscape) that may propagate through the system via the medium, influencing the actions of other agents.

Interpreting the adaptive landscape as a spatiotemporal information field allow us to state a dynamical rule for its evolution. In addition to the actions of agents, we want to consider processes that account for a finite lifetime of the information stored at each position and its diffusion in space. So we can write in general

$$\frac{\partial h_\alpha(\overrightarrow{r}, t)}{\partial t} = \sum_i s_i(\alpha, t)\delta_{\alpha_i,\alpha}\delta(\overrightarrow{r} - \overrightarrow{r_i}(t)) - k_\alpha h_\alpha(\overrightarrow{r}, t) + D_\alpha \nabla^2 h_\alpha(\overrightarrow{r}, t) \tag{5.11}$$

The first term account for agents' actions: only agents that are in (x, y) and have $\alpha_i = \alpha$ contribute to the field with the term s_i, which may depend on time and on the value of α_i. The second term represent a decay process that account for the finite lifetime. Finally, the last term represent a diffusion process with diffusion constant D_α, different for each field component.

Chapter 6

Software used: SLAPP

To build simulations of our models we use an agent based simulation shell, namely, SLAPP which stands for Swarm-Like Agent Protocol in Python. The software package, tutorials and examples files, and a reference handbook of SLAPP can be found at https://github.com/terna/SLAPP/. We describe only the basic features of SLAPP and the reader is referred to the above mentioned website to a more detailed description.

As the name suggest, SLAPP is an implementation of Swarm in python. Swarm was a project of the Santa Fe Institute, which was started with the aim to give AB modelers standardized software to build AB simulations and to make easier the sharing of simulations software and results among scientists ([13]):

> Most scientists are not trained as software engineers. As a consequence, many home-grown computational experimental tools are (from a software engineering perspective) poorly designed. The results gained from the use of such tools can be difficult to compare with other research data and difficult for others to reproduce because of the quirks and unknown design decisions in the specific software apparatus. [...] A subtler problem with custom-built computer models is that the final software tends to be very specific, a dense tangle of code that is understandable only to the people who wrote it. [...]
>
> In order for computer modeling to mature there is a need for a standardized set of well engineered software tools usable on a wide variety of systems. The Swarm project aims to produce such tools through a collaboration between scientists and software engineers. Swarm is an efficient, reliable, reusable software apparatus for experimentation. If successful, Swarm will help scientists focus on research rather than on tool building by giving them a standardized suite of software tools that provide a well-equipped software laboratory.

In Swarm formalism, and then in SLAPP, the basic element of a computer experiment is the agent, an actor able to generate events or to act. Processes that involve agents are described by means of a schedule of events: a list of ordered events or actions followed by agents. The time in the simulation of a process goes on together with the execution of the scheduled activities. One of the main idea of Swarm is to collect a list of agents, the environment where

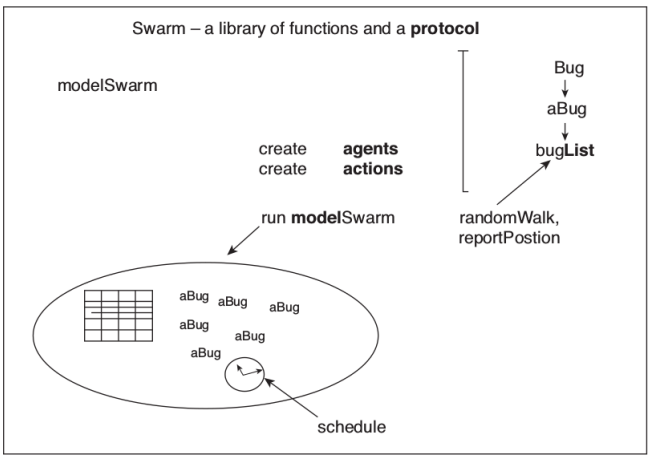


Figure 6.1: Representation of modelSwarm class. "Bug" is the agents class, "aBug" is an instance of the "Bug class", i.e. the agents, ans "bugList" is the list containing all the agents. "RandomWalk" and "ReportPosition" are the actions to be performed, that constitutes the events schedule. Image taken from [1]

they act together with an events schedule in a unique object, the "swarm" (see figure 6.1).

We have seen in the chapter "Agent based model" that to build an ABM, one has to specify the agents the environment and the interaction rules. So a swarm may represent an ABM, may be that we want to simulate. For this reason it is called "modelSwarm" in SLAPP. Figure 6.1 shows explicitly the protocol to build a "swarm", suggested in its definition: first create the agents, then create the actions schedule and finally run the model, allowing agents to act according to the schedule.

A swarm can also contain other swarms, each with its own agents and events schedule, to represent different level of interactions. Nested swarms may represent higher level entities composed by many individuals that act as a single, such as companies or governments.

This formalism allows to associate to each agent a swarm that she will use as a model of her world. To this end Swarm includes also an "observer" class ("observerSwarm" in SLAPP), that owns one or more models ("swarm" object) that she can run, together with instruments to analyze the results (see figure 6.2). We note also that the observer class may represent also the person which is doing the computer experiment, included our work.

Swarm was based on a set of libraries which have become less useful with the increasing diffusion and development of OOP (Object Oriented Programming) languages. SLAPP has maintained the protocol of Swarm and implement the libraries exploiting python. The choice of the language is due both to its simplicity and to its capability of communicating with other languages and software.

SLAPP package is organized in different folders, many of them contains tutorials. The actual code of SLAPP is contained in the folder "6 objectSwarmObserverAgents_AESOP_turtleLib_NetworkX", which from now on we call the

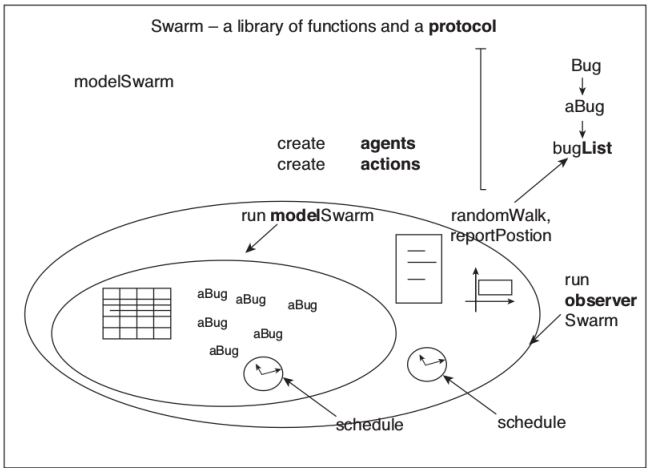


Figure 6.2: Representation of observerSwarm class. Image from [1]

main folder. Here we find the folder "$$slapp$$", that contains the model-independent part of SLAPP, and the user is recommended to not modify this part. In particular, it contains the files

- modelSwarm.py
- observerSwarm.py

where the classes "modelSwarm" and "observerSwarm" are defined. Note that not all functionalities of the classes are implemented here and, in general, the user is allowed to define or re-implement methods of the classes in other files. It contains also other files that together with the last cited classes implement the scheduling mechanism. The other folders contain examples of simulations, whose aim is to show SLAPP capabilities and to present different possible ways to set-up an ABS.

To build a new model one has to create a new folder inside the main folder. The name of the folder is used to start the project when SLAPP is run.

In the folder, we have to write agents classes through files with the ".py" extension. As said above we have also to define some functionalities of "modelSwarm" and "observerSwarm" classes. Note that this is not a drawback, it is the SLAPP feature that allow to build new models inside it, without modifying the main Swarm protocol. Looking at the examples contained in the main folder, we can define methods of the first class through the file "mActions.py" and methods of the latter in the file "oActions.py". In particular, we have to define the function "CreateAgentClass" in "mActions.py".

Once we have defined agents' classes and the agents' creation step, we have to define the events schedule. SLAPP has three levels or layers of scheduling activities: the observer layer, the model layer and the "AESOP" layer, as it is called in SLAPP descriptions (see the above mentioned website or see [1]), that of the agents. The last layer is a feature of SLAPP not shared with Swarm. Its introduction serve to distinguish the activity of the "modelSwarm" from the events schedule that regulate agents behavior.

	A	B	C	D	E	F	G	H	I	J
1					comments here or in successive columns					
2	#	1			standard (background) actions, like move, are applied to “all”					
3	bland	eat			bland agents are those not specified in dedicated .txt files,					
4	bland	dance			with the related names reported in the agTypeFile.txt file					
5	#	2								
6	#	4								
7	all	0,5	dance		all agents acting					
8	tasteC	eat			tasteC agents acting					
9	#	5								
10	all	eat								
11	all	dance								
12	#	7								
13	tasteA	0,5	dance		tasteA agents acting					
14	#	8								
15	tasteB	dance			tasteB agents acting (no agents of this type exist here)					
16										

Figure 6.3: A events schedule. Image taken from an example of the main folder of SLAPP

The actions defined in the schedule of each layer are done cyclically. A cycle begin with observer actions. Typically, the observer update her clock state and ask the model to do a cycle of its activity; but she can also change model parameters, or she can extract information from the model asking agents or a subset of them to send messages. The model too has a clock that is updated; in general model clock can be different from observer clock. During one of its cycles, the model activate the events schedule of the agents, but it can also perform other actions.

SLAPP user can define observer actions in the “oActions.py” file. Then she has to write in the file “observerActions.txt” those actions that she want the observer performs, having care of the order in which actions are written, being the same in which those are executed. The same reasoning applies to model actions; in this case the relevant files are “mActions.py” for actions definitions and “modelActions.txt” for the model schedule.

Events schedule of AESOP layer is defined via spreadsheet tables; to this end it can be used the files “schedule.xls”. To explain how to write the events schedule we refer to the example shown in figure 6.3.

Rows starting with the “#” character serve to define the time at which actions have to be executed; the value of time is written in the second column and it is intended that all actions written below, until another “#” sign, will be executed during that time. Writing another time value in the third column tell the program that actions below have to be executed for every time included between the two values given.

The others non-empty rows contain in the first column a group name, which is the name of the set of agents questioned; in the second column there is an action name. These rows tell the program that the subset of agents indicated by the name in the first column has to make the action written in the second one. Group names have to been declared in the file “agType.txt” and they have to match the values of the agents’ variable “agType”. Action names are the same of agents’ methods.

We can also implement environment characteristic different from default modifying the class in the file “WorldState.py”. Finally, we can set up initial procedures, such as that of asking the number of cycles to be done, through the file “parameters.py”.

Chapter 7

Hayekian market

In the second chapter we have seen that the standard accepted theory, GET, tries to describe the evolution of markets in terms of a series of equilibrium states. To accomplish this task, GET relies on a set of assumptions that has revealed unrealistic, particularly on the light of complex systems reasoning.

Roughly a market is conceived as a set of people acting independently of one another, each trying to maximize her net utility; actions are based on the knowledge of the external events that can influence the markets and on the knowledge of other agents' features (tastes, reservation prices, etc.). Equilibrium is defined as the situation where the actions performed by all the agents maximize the total net utility. According to perfect competition hypothesis, every agent has perfect and complete knowledge. So every one takes its decisions on the basis of the same set of information of all the others. This allow to reduce the study of the whole system to the study of the single agent, and to deduce the final outcome from initial conditions only, without examining the detailed path of actions.

Since complete and error free information seem to be unrealistic, rational expectations hypothesis relaxes these requirements; agents can take decisions which are not optimal. But it is assumed that errors due to incomplete information are random and unbiased and so cancels out when considering the actions of many agents. Then, it is still possible to focus only on the single agent, since the complete common set of error free information is replaced with a set that contains only random errors and lacks of information about randomly selected part of the system or about randomly selected external events.

We have seen that complex systems approach undermines this picture, mainly because it does not assume equilibrium and because it does not try to eliminate interactions. ABM usage plays an important role in refusing these assumptions since their main utility resides on the need of analytical tractability to have answers from models.

But critiques to GET comes also from economics itself. Particularly useful to our purposes is the interpretation of markets behavior given by F. Hayek[1], a twentieth century economists, whose thought has recently attracted new attention. Following Bowles et al.[3]:

[1]Friederick Hayek (1899-1992) was an Austrian-British economist. He won the Nobel Memorial Prize in Economic Sciences (in pair with Grunnar Mydral)in 1974.

> Friedrich A. Hayek is known for his vision of the **market economy as an information processing system characterized by spontaneous order**: the emergence of coherence through the independent actions of large numbers of individuals, **each with limited and local knowledge**, coordinated by prices that arise from decentralized processes of competition.

Hayek had a different notion of equilibrium with respect that exposed above. He thought that every person has subjective and incomplete information on the world and on the other people, and consequently she has certain foresight about futures events and about other people actions. Based on this information, everyone makes an actions plan to be carried out. Equilibrium occurs when all the plans can be realized without disturbing each other. This means that in equilibrium the foresights of every one are not contradicted; the actions of an agent and the order in which these are performed do not cause changes in other agents' plans.

However, this condition is unlikely to be fulfilled in our world and Hayek himself was not convinced that market economy is constantly in equilibrium. Following again Bowles et al. [3]

> But Hayek was not particularly interested in the properties of equilibrium itself, and saw the strength of the market economy as arising from the learning and diffusion of new information that it accomplishes in disequilibrium. Unforeseen (and often unforeseeable) changes in economic fundamentals that are initially recognized by only a small number of individuals would lead, through the messages conveyed by changes in prices, to adjustments across the entire economy.

Both in equilibrium or in disequilibrium situations, information plays an important role in Hayek's thought. To make clear this point it is useful to look at Hayek's critique of Walrasian equilibrium mechanism and of perfect competition hypothesis.

He argues that the walrasian auctioneer, or a central planner more generally, can not have the necessary information needed to drive market at equilibrium. Indeed, to accomplish her task, the auctioneer has to know taste and future intentions of all the agents involved. This is impossible not only because of the time needed to acquire and handle such an amount of information, but also because this information does not exist before interactions have taken place. According to Hayek it arises from interactions.

This view of information is incompatible also with assumptions of perfect competitions hypothesis. Hayek believed that giving agents objective, correct and complete information in an economic model rules out the actual mechanism that can drive the system towards equilibrium: the generation of information, its diffusion and its use. What is interesting for Hayek is how the different pieces of information distributed over people are communicated and used to give rise to market global behavior, especially in presence of unforeseen changes. Following again Bowles et al. ([3]):

> Hayek's belief was that this process would lead to a diffusion of individually acquired knowledge across the economy and result in a

> more effective utilization of knowledge than would be possible under a centralized mechanism. In Hayek's view, the data that individuals have at their disposal consists of "abstract signals" including prices proposed, actions taken by others, and if bargaining actually takes place, information gained in the bargaining process even when no transaction was agreed upon (Kirman, Schulz, Härdle, and Werwatz 2005).

In our models we adopt Hayek's perspective, but we do not assume that the equilibrium is always reached. In particular, we retain that changes in prices are to be interpreted as information that flows through the systems, and that influences agents' behavior. But we do not assume that it diffuses to the whole system, nor that the system will reach a "desirable" state if left to itself.

Chapter 8

The first model

8.1 Model description

In our work we study a market of agricultural goods through agent-based models. As said in the chapter on ABM (at the end of the first section), when there are real data on this market, they are aggregated data, unusefull to our purposes. Then, we study an hypothetical simplified market.

We refer to a Hayekian market model, where economic agents have limited and local information about the environment and about the others agents. In this market, agents are not passive price takers. Instead everyone has a reservation price which she does not communicate to other, and by which she evaluates prices proposed by other agents: she will buy (sell) if the proposed price is not to bigger (smaller) than her reservation price. So prices of goods are not given but they emerge from a self organization process.

To exploit Brownian Agent approach, we develop more models starting from a simple one and then adding up some degree of complexity. In all the models we have built, we consider only two types of agent, consumers and producers, neglecting intermediaries. We further assume that producers have always enough products to satisfy all consumers' requests, and that consumers do not have budget constraint.

In the first model agents live in a square of side L which constitutes the environment. We have considered both reflecting walls and periodic boundary conditions. Consumers move in the square as Brownian particles; their state variables are the position vector $\overrightarrow{r_i}$, the velocity vector $\overrightarrow{v_i}$ and the reservation price θ_i (the index i run over all consumers).

Producers are in fixed position and can't move. Every producer occupy a finite area of the square, and everyone can sell only if a consumers has entered her region. Producer's state variable are then her vector position $\overrightarrow{r_j}$ (which remains fixed) and her reservation price θ_j (the index j runs over all the producers).

When a consumer is inside the region of a producer, she looks at the proposed price, and she buys if this is less than her reservation price. When she buys she decreases her reservation price by a fixed quantity λ^c, whereas the producer who has sold increases her reservation price by the quantity λ^p. They do the inverse if the consumer refuses the producer's offer. We take $\lambda^c \gg \lambda^p$ because, in real situations, producers are slower than consumers in changing their reserve

prices.

The evolution of the state variables of each consumer is determined by the set of equations:

$$\frac{d\overrightarrow{r_i}}{dt} = \overrightarrow{v_i} \tag{8.1}$$

$$\frac{d\overrightarrow{v_i}}{dt} = -\gamma\overrightarrow{v_i} + \sqrt{2S}\xi_i \tag{8.2}$$

$$\frac{d\theta_i}{dt} = \lambda_i\hat{\mu_i}(1-2\hat{s_i}) \tag{8.3}$$

The first two equations are those of a Brownian particle that diffuse in a viscous media, hence $\gamma = \frac{\gamma_0}{m}$ represents the friction divided by the mass of the particle and the term $S = \gamma\frac{k_BT}{m}$ is related to the diffusion coefficient $D = \frac{k_BT}{\gamma_0}$.

We consider a further simplification dealing with the over-damped limit of the equations of motion, i.e. when $\gamma_0 \gg 1$. In this limit, accelerations are suppressed in a much smaller time than those required to appreciate sensible variation of the position. Then, we can set $\frac{d\overrightarrow{v_i}}{dt} \approx 0$ (adiabatic approximation) to find

$$\overrightarrow{v_i} = \sqrt{2D}\xi_i \tag{8.4}$$

Substituting in the position equation we find

$$\frac{d\overrightarrow{r_i}}{dt} = \sqrt{2D}\xi_i \tag{8.5}$$

Then, consumers' state variable are reduced to positions and reservation prices.

The third equation represents the change in the reserve price; $\hat{\mu_i}(t)dt$ is the number of meeting between the ith consumer and the producers in the time interval $[t, t+dt]$, and $\hat{s_i}(t)dt$ is the number of purchases done in the same time interval divided by the number of encounters. The first quantity depends only upon the relative position between the ith consumer and all the producers, the latter depends only on the differences between the reservation price of the consumer and that of the producers.

For the producer we have

$$\frac{d\theta_j}{dt} = \lambda_j\hat{\mu_j}(2\hat{s_j}-1) \tag{8.6}$$

As for consumers, μ_j represents the times that the j-th producer meets the consumers and $\hat{s_j}$ the fraction of encounters in which the producer have sold.

To obtain a rough result, we assume that every producer always have enough products to satisfy all consumers' requests. In this way we can calculate the number of times the ith consumer meet the jth producer in the time interval $[t, t+\Delta t]$ through the integral

$$\int_0^{\Delta t}\int_{x_j-l/2}^{x_j+l/2}\int_{y_j-l/2}^{y_j+l/2} p_i(x,y,t+\tau)dxdyd\tau \tag{8.7}$$

Where $p_i(x,y,t)$ is the time-dependent probability density of the ith consumer's position.

Consumers move as Brownian particles so after a sufficiently large time they reach a stationary state, i.e. the probability density of finding a particle in a given position is time independent and it reads (see appendix A):

$$p(\overrightarrow{r}) = \frac{1}{L^2} \tag{8.8}$$

which is uniform. Inserting it in the integral (8.7) we obtain

$$\nu_{i,j} = \Delta t \frac{l^2}{L^2} \tag{8.9}$$

The number of producer that a consumer meets in the time interval $[t, t + \Delta t]$ is then

$$\mu_i \Delta t = N_p \Delta t \frac{l^2}{L^2} \tag{8.10}$$

and we obtain for the quantity μ_i of equation (8.3)

$$\mu_i(t) = N_p \frac{l^2}{L^2} \tag{8.11}$$

This coefficient does not depend on time and on the position of the consumer, nor on the i index.

In a similar way we obtain for the producer

$$\mu_j(t) = N_c \frac{l^2}{L^2} \tag{8.12}$$

So we can see that, after the diffusion process has reached the equilibrium, the evolution of the reserve prices does not depend on consumers' positions. It depends only on the distribution of the reserve prices of the agents, on their numbers and on the velocity at which consumers and producers change their price after a meeting.

Rewriting only the reserve prices equations with the results (8.11) - (8.12)

$$\frac{d\theta_i}{dt} = \lambda_i N_p \frac{l^2}{L^2}(1 - 2\hat{s_i}) \tag{8.13}$$

$$\frac{d\theta_j}{dt} = \lambda N_c \frac{l^2}{L^2}(2\hat{s_j} - 1) \tag{8.14}$$

Since every encounter between a consumer i and a producer j implies a change of $\Delta\theta_i = \pm\lambda_i$ in the consumer's price and a correspondent change of $\Delta\theta_j = \mp\lambda_j$ in producer's price we have

$$\frac{\Delta\theta_i}{\lambda_i} + \frac{\Delta\theta_j}{\lambda_j} = 0 \tag{8.15}$$

and we conclude that the quantity $\sum_i \frac{\theta_i}{\lambda_i} + \sum_j \frac{\theta_j}{\lambda}$ remains constant during the evolution of the system.

If the interactions are uniform we expect that in the long time limit all the prices are similar. Denoting with the subscript I the initial value of a quantity and with the subscripts F its value at a successive time we can write

$$\sum_i \frac{\theta_i^I}{\lambda_i} + \sum_j \frac{\theta_j^I}{\lambda_j} = \sum_i \frac{\theta_i^F}{\lambda_i} + \sum_j \frac{\theta_j^F}{\lambda_j} \tag{8.16}$$

Setting $\theta_i^F = \theta_j^F = \theta^H$, where θ_H is the Hayekian equilibrium value, we obtain

$$\theta_H = \frac{\sum_i \frac{\theta_i^I}{\lambda_i} + \sum_j \frac{\theta_j^I}{\lambda_j}}{\sum_i \frac{1}{\lambda_i} + \sum_j \frac{1}{\lambda_j}} \tag{8.17}$$

In the case $\lambda_i = \lambda_j = \lambda \forall i, j$ we see that the sum of prices is conserved and that the final price is simply the initial mean price. So to study the evolution of the system towards the final price we can look at the variance. Denoting with $d_i = \theta_i - \theta^H$ the difference between the price of the i-th consumer and the final price, and with $d_j = \theta_j - \theta^H$ that relative to the j-th producers, after an interactions between the two agents we have

$$d_i' = d_i \pm \lambda \tag{8.18}$$
$$d_j' = d_j \mp \lambda \tag{8.19}$$

From these expressions we can find the variance variation caused by the interaction

$$\Delta\sigma^2 = -\frac{2\lambda}{N-1}(|\theta_i - \theta_j| - \lambda) \tag{8.20}$$

So if the difference of prices before the interaction is greater than the price step then this difference will be reduced after the interaction and also the variance will decrease since $\Delta\sigma^2 < 0$. On the other hand if the difference is smaller than λ we have $\Delta\sigma^2 > 0$.

When $\lambda_i \neq \lambda_j'$ the mean price is not conserved and we can not use the variance to looking for equilibrium convergence. However we can use in a similar way the function

$$f^{eq} = \sqrt{\frac{1}{N-1}\sum_k d_k^2} \tag{8.21}$$

where the index k runs over all the agent and $d_k = \theta_k - \theta^H$. We have considered the square root for future convenience.

8.2 Code

As said in chapter 7, we use SLAPP shell to implement our simulation. We describe in the following: the classes we have written, the function used by "modelSwarm" to create the agents, the event schedule relative to the three levels available on SLAPP.

Since consumers and producers have similar characteristics we have defined a parent class, the class "Agent", from which the classes "Consumer" and "Producer" inherits (fig. 8.1). We have also used the SLAPP class "WorldState",

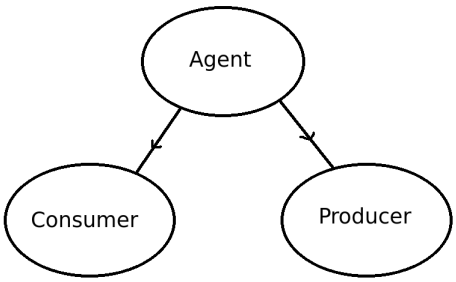


Figure 8.1: Classes diagram

adding some variables and methods. This class is used to represent environment features and to collect data from agents.

Agent class data members are:

- number: an integer representing agent's identity.
- lX,rX,bY,tY: four float representing respectively the left, rigth, lower and upper limit of agents' environment.
- myWolrdState: a variable containing the address of a "WorldState" object.
- xPos,yPos: 2 float numbers representing agent's position coordinates.
- agtype: a string variable, empty by default, to be used by son classes.
- price: the reservation price of the agent.
- pricelist: a list to record prices.
- p_step: a float number representing the variation of the reserve price after an interaction.
- noise: a float used by the method "noisy_change" described below.

Its methods are:

- change: takes a variable $m_{out}(\pm 1$, rap presenting the outcome of a meeting between another agent) and changes the price variable through the formula

$$p_{t+1} = p_t + p\_step * m_{out} \tag{8.22}$$

- noisy_change: takes a variable $m_{out}(\pm 1$, rap presenting the outcome of a meeting between another agent) and changes the price variable through the formula

$$p_{t+1} = p_t + p\_step * m_{out} + noise * m_{out} * \hat{u} \tag{8.23}$$

where $\hat{u}$ is a random number taken from a uniform distribution over $[-1, 1]$

- record: appends the actual value of the price to the price list

We report below the code of the class.

```
class Agent(SuperAgent):
    def __init__(self, number, myWorldState,
                 xPos, yPos, price, p_step, noise, lX=-20, rX=19, bY=-20, tY=19, agTy
        # the environment
        self.agOperatingSets = []
        self.number = number
        self.lX = lX
        self.rX = rX
        self.bY = bY
        self.tY = tY
        if myWorldState != 0:
            self.myWorldState = myWorldState
        self.agType = agType

        # the agent
        self.xPos = xPos
        self.yPos = yPos
        print("agent", self.agType, "#", self.number,
              "has been created at", self.xPos, ",", self.yPos)
        self.price = price
        self.priceList = []
        self.priceList.append(self.price)
        self.p_step = p_step
        self.noise = noise

    def change(self, response):
        self.price = self.price + self.p_step*response

    def noisy_change(self, response):
        self.price = self.price + self.p_step*response
                           + self.noise*random.uniform(-1,1)*response

    def record(self):
        self.priceList.append(self.price)
```

Producer class data members are:

- radius: a float number representing the radius of the producer selling area.
- mycList: a list used to record the the identity of consumers that have interacted with the producer.

Its methods are:

- update_cList: takes as argument an object "Consumer" and, if not even present, appends consumer's identity number to the list mycList.

```
class Producer(Agent):

    def __init__(self, number, myWorldState,
                 xPos, yPos, radius, price, p_step, noise, chain_length,
                        lX=-1, rX=1, bY=-1, tY=1, agType=""):

        Agent.__init__(self, number, myWorldState,
                 xPos, yPos, price, p_step, noise, lX, rX, bY, tY, agType)

        self.radius = radius

        self.mycList = []

    def update_cList(self, consumer):

        if(not consumer.number in self.mycList):
          self.mycList.append(consumer.number)
```

For consumers' motion we consider the over-damped limit (see Appendix A), then we do not use equations (5.2) but instead (8.24).

Consumer class data member are:

- step: a float representing the time step used by functions "evolve" and "noisy_evolve".
- diff: a float that stand for the diffusion coefficient.
- x,y: two lists used to record position.
- myseller: a variable used to record the last seller from which the consumer have bought (used in the second model).
- bought: a variable used in the second model and third model.
- sradius: the radius of consumer's "field of vision" (used in the second model).
- strength: a variable used in the third model.

Its methods are

- evaluate: returns 1 if consumer's price is not smaller than the price of the Producer object passed as argument, else it returns -1.
- evolve: updates position according to the rules (Euler algorithm for SDE)

$$\overrightarrow{r}_{n+1} = \overrightarrow{r}_n - \nabla h(\overrightarrow{r}_n) * step + \sqrt{2 * diff * step} * \overrightarrow{\xi}_n \qquad (8.24)$$

  where ∇h is the gradient of the adaptive landscape used in the third model. In the first and in the second model we have $h(\overrightarrow{r}) = 0 \forall \overrightarrow{r}$.
- maintain_reflecting_walls: assure that consumer's position respect reflecting walls boundary condition.

- maintain_pbc: assure that consumer's position respect periodic boundary condition.
- record_position: save the current position in the lists x and y
- search_producer: check if the consumer is in the area of any producer. If she is, this methods call the method "evaluate" to evaluate producer's offer. After the "evaluate", both producer and consumer update their prices through the method "change" of class "Agent". The method calls "Worldstate" methods to keep track of meetings and purchases.
- search_producerV2: the only change with respect to the previous method is the way in which consumer search producers' area. To be used when producers are at the nodes of a square lattice.

The code:

```
# Consumer.py
from Tools import *
from agTools import *
from Agent import *
from Producer import *
import commonVar as common
import numpy as np

class Consumer(Agent):

    def __init__(self, number, myWorldState,
                 xPos, yPos, step, diff, price, p_step, noise,
                  lX=-1, rX=1, bY=-1, tY=1,  agType=""):

        Agent.__init__(self, number, myWorldState,
                 xPos, yPos, price, p_step, noise,lX, rX, bY, tY, agType)


        self.step = step
        self.diff = diff
        self.x = []
        self.y = []
        self.myseller = 0
        self.bought = 0

    def evolve(self):
        grad = np.zeros(2)

        grad = self.myWorldState.AL.get_gradient(self.xPos,self.yPos)

        self.xPos += -grad[0]*self.step +
                np.sqrt(self.diff*2*self.step)*random.gauss(0,1
        self.yPos += -grad[1]*self.step +
                np.sqrt(self.diff*2*self.step)*random.gauss(0,1)
```

```
    def mantain_reflecting_walls(self):           #ATTENTION: displacement from the boarders ha
                                                  #to be smaller than the size of the world

        if(self.xPos>self.rX):
            self.xPos = self.rX-(self.xPos- self.rX)
        if(self.xPos<self.lX):
            self.xPos = self.lX+(self.lX- self.xPos)
        if(self.yPos>self.tY):
            self.yPos = self.tY-(self.yPos- self.tY)
        if(self.yPos<self.bY):
            self.yPos = self.bY+(self.bY- self.yPos)


    def mantain_pbc(self):            #NOTE: the world has to be centered on (0,0)

        if(self.xPos>self.rX or self.xPos<self.lX):
            LX = self.rX - self.lX
            xprov = self.xPos - self.lX
            self.xPos = xprov%LX + self.lX

        if(self.yPos>self.tY or self.yPos<self.bY):
            LY = self.tY - self.bY
            yprov = self.yPos - self.bY
            self.yPos = yprov%LY + self.bY


     def record_position(self):

        self.x.append(self.xPos)
        self.y.append(self.yPos)


    def search_producer(self):
        distance = 0
        for agent in self.agentList:
           if(type(agent) is Producer):

              distance = pow((self.xPos-agent.xPos),2) + pow((self.yPos-agent.yPos),2)
              distance = np.sqrt(distance)


              if(distance < agent.radius):
                 agent.update_cList(self)
                 self.myWorldState.meetingList.append(common.cycle)
                 response = self.evaluate(agent.price)
                 if(response == 1):
                    self.myWorldState.purchaseList.append((common.cycle, agent.price))
```

```
                self.myseller = agent
                self.bought = response
                self.last_purchase = agent.price

                self.change(-response)
                agent.change(response)
                break


    def search_producerV2(self):
        L = self.rX -self.lX

        l = L/common.sqNp

        r = np.array([(-L+l)/2, (-L+l)/2])

        n1 = int((self.xPos+L/2)/l - 1/2)
        m1 = int((self.yPos+L/2)/l - 1/2)
        rs = r+np.array([n1*l,m1*l])
        if(self.xPos> rs[0]+l/2):
          n1 = n1+1
        if(self.yPos > rs[1]+l/2):
          m1 = m1 +1
        j = common.sqNp*m1+n1

        distance = 0

        for agent in self.agentList:
           if(type(agent) is Producer and agent.number == j):

             distance = pow((self.xPos-agent.xPos),2) + pow((self.yPos-agent.yPos),2)
             distance = np.sqrt(distance)


             if(distance < agent.radius):
                agent.update_cList(self)
                self.myWorldState.meetingList.append(common.cycle)
                response = self.evaluate(agent.price)
                if(response == 1):
                   self.myWorldState.purchaseList.append((common.cycle, agent.price))

                self.myseller = agent
                self.bought = response
                self.last_purchase = agent.price

                self.change(-response)
                agent.change(response)
             break
```

```
    def evaluate(self, price):
        if(self.price >= price):
            return 1
        else:
            return -1
```

As mentioned above, "Worldstate" class is defined and used in other SLAPP project. Here we describe only the data members and methods used in our simulation.

The data members are:

- purchaseList: a list used to record purchases prices.
- meetingList: a list used to record the encounters between producers and consumers.
- AL: a "AdaptiveLandscape" object, described in the third model.

Its methods are:

- buildLandscape: create a "AdaptiveLandscape" object (described in section10.2).
- updateBoundaries: described in section10.2
- updateLandscape: described in section10.2
- showLandscape: described in section10.2

```
# WorldState.py
from Tools import *
from AdaptiveLandscape import *


class WorldState(object):



    def __init__(self):
        # the environment
        self.generalMovingProb = 1
        print("World state has been created.")
        self.purchaseList=[]
        self.meetingList=[]
        self.AL = 0

    def buildLandscape(self):

        self.AL = AdaptiveLandscape(common.size, common.Npoints, common.decay,
                      common.field_diffusion, common.time_step)
```

```
        print(self.AL.h)

    def updateBoundaries(self):
        self.AL.mantain_pbc()

    def updateLandscape(self):
        self.AL.evolve()

    def showLandscape(self):
        self.AL.show2d()


    # ",**d" in the parameter lists of the methods is a place holder
    # in case we use, calling the method, a dictionary as last parameter

    # set generalMovingProb
    def setGeneralMovingProb(self, **d):
        if "generalMovingProb" in d:
            self.generalMovingProb = d["generalMovingProb"]
            print("general moving probability now set to",
                  self.generalMovingProb, "in world state")
        else:
            print("*********** key 'generalMovingProb' is not defined")
            self.generalMovingProb = 1

    # get generalMovingProb
    def getGeneralMovingProb(self):
        return self.generalMovingProb
```

To create the agents modelSwarm uses the function "createTheAgentClass". The function create consumers taking the diffusion coefficient, the price step, the time step and noise values from the file "commonVar.py" (see chapter 6), and so for producer variables price step, radius and noise. Consumers are created at random position taken from a uniform distribution over the square of side 2 centered on the origin. Producers instead are created at the nodes of a regular square lattice. This function sets also the initial prices configuration that we have considered (see the next section).

```
def createTheAgent_Class(self, line, num, agType, agClass):
    # explictly pass self, here we use a function

    # check if the file having the content of agClass and extension
    # .py exists

    common.agClassVerified = False
    if not common.agClassVerified:
        try:
            exec("import " + agClass)
```

```
            common.agClassVerified = True
        except BaseException:
            print("Missing file " + agClass + ".py")
            os.sys.exit(1)

    print(agClass)

    L=self.worldXSize
    N = common.sqNp
    l = L/N
    r = np.array([(-L+l)/2, (-L+l)/2])

    m = int((num)/N)
    n = num-N*m
    r += np.array([n*l,m*l])

    x = random.uniform(-1,1)
    y = random.uniform(-1,1)


    p_price = random.gauss(15,1)
    c_price = random.gauss(15,1)


    # first step in exec:
    # access the files of the classes to create the instances
    # N.B. to simplify the structure of SLAPP, the name of the
    # class and the name of the file containing it, have to be the same.
    if len(line.split()) >= 1:  # weak control, can be improved
        try:
            space = {
                'num': num,
                'sW': self.worldState,
                'random': random,
                'leftX': self.leftX,
                'rightX': self.rightX,
                'bottomY': self.bottomY,
                'topY': self.topY,
                'x': x,
                'y': y,
                'xprod': r[0],
                'yprod': r[1],
                'diff': common.diffusion,
                'c_price_step': common.price_step,
                'p_price_step': common.relative_inertia*common.price_step,
                'p_price': p_price,
                'c_price': c_price,
                'time_step': common.time_step,
                'prrad': common.producers_radius,
                'noise': common.noise,
```

```
                'group' : group,
                'chain_length': chain_length,
                'agType': agType}

            if(agClass == 'Consumer'):
                exec("from " + agClass + " import *;" +
                 "anAgent = " + agClass + "(num, sW," +
                 "x," +
                 "y," +
                 "time_step," +   #step
                 "diff," +   #diffusion coefficient
                 "c_price," +
                 "c_price_step," +       #price step
                 "noise," +
                 "agType=agType)", space)


            elif(agClass == 'Producer'):
                exec("from " + agClass + " import *;" +
                 "anAgent = " + agClass + "(num, sW," +
                 "xprod," +
                 "yprod," +
                 "prrad,"+     #radius
                 "p_price," +
                 "p_price_step," +       #price step
                 "noise," +
                 "agType=agType)", space)
            anAgent = space['anAgent']
            self.agentList.append(anAgent)

        except BaseException:
            print("Argument error creating an instance of class", agClass)
            os.sys.exit(1)
```

The scheduling at the level of the observerSwarm and of the modelSwarm are simple: for the observer we have the instructions (contained in the file oActions.txt)

```
modelStep clock
```

The first command tells the observer to evolve its model, the second to update her clock. For the model we have (from the file mActions.txt)

```
read_script
```

This instructions represents the step that the observer asks its model to make. The instructions tells the model to read and execute the events schedule of agents.

Figure 8.2 shows agents' event schedule

	A	B	C	D
1	#	1		
2	WorldState	computationalUse	buildLandscape	
3				
4				
5	#	1	100000	
6	tasteA	evolve		
7	tasteA	manatain_pbc		
8	tasteA	record_position		
9				
10	tasteA	search_producerV2		
11				
12	all	record		

Figure 8.2: First model event schedule

With the first instruction, the "WolrdState" creates a plain landscape (zero in every point), used by consumers' method evolve. In the first and second model the field remains zero.

The next three instructions refer to consumers, whose "agType" variable is equal to "TasteA", the string value used in the scheduling mechanism to call and quest consumers. Note that to change boundary conditions we have only to substitute "mantain_pbc" with "mantain_reflecting_walls".

The next instructions call the methods search_producerV2 (which can be substituted with search_producer when considering producer at random position).

The last row, ask all the agent to record their prices.

Finally, when a simulation terminates we analyze and record the data through the functions that we have wrote in the file "oActions.py" (see chapter 6). The functions are:

- d01b, do2a and do2b: functions necessary to SLAPP functioning.
- show_results: calls all the methods listed below.
- predict: calculate the predicted finale price θ^H.
- show_purchases: plot the price of purchases done against time.
- plot_link_number: plot the number of different consumers that have interacted with a given producer against producers' identity number.
- prod_position: plot producers' position.
- cons_position: plot consumers' position.
- equilibrium_function: calculate the function f^{eq} (see eq.8.21).
- mean_p_price: calculate producers mean price at time t, passed as argument.
- mean_c_price: calculate consumers mean price at time t, passed as argument.
- show_mean_producer: plot mean and standard deviation of producers prices against time.

- show_mean_consumer: plot mean and standard deviation of consumers prices against time.
- show_Chistogram: plot the distribution of consumers' prices
- show_Phistogram: plot the distribution of producers' prices
- save: save the list of agents in a binary file and it asks the user to insert the file name. It uses pickle python module to save user defined objects. It also save parameters values in a ".txt" file.

```
from Tools import *
from Agent import *
from Consumer import *
from Producer import *
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import networkx as nx
import pickle as pk

def do1b(address):
    pass


def do2a(address, cycle):
    self = address  # if necessary

    # ask each agent, without parameters

    print("Time = ", cycle, "ask all agents to report position")
    askEachAgentInCollection(
        address.modelSwarm.getAgentList(),
        Agent.reportPosition)


def do2b(address, cycle):
    self = address  # if necessary

    # ask a single agent, without parameters
    print("Time = ", cycle, "ask first agent to report position")
    if address.modelSwarm.getAgentList() != []:
        askAgent(address.modelSwarm.getAgentList()[0],
                 Agent.reportPosition)



def show_results(address):
    self = address
    print(type(address))
    print('Total number of encounters = ',
```

```
            len(address.modelSwarm.worldState.meetingList))
    print('Total number of purchase = ',
            len(address.modelSwarm.worldState.purchaseList))
    plot_link_number(address)

    show_purchases(address)

    predicted_price(address)
    print('Predicted final price', predicted)
    prod_position(address)
    cons_position0(address)
    print(address.nCycles)
    for i in range(6):
        t = int(address.nCycles/5)*i
        if(t > address.nCycles-1):
           t = -1
        cons_position(address, t)

    equilibrium_function(address)
    show_mean_producer(address)
    show_mean_consumer(address)

    for i in range(6):
        t = int(address.nCycles/5)*i
        if(t>address.nCycles-1):
           t = -1
        show_Chistogram(address, t)
    for i in range(6):
        t = int(address.nCycles/5)*i
        if(t>address.nCycles-1):
           t = -1
        show_Phistogram(address, t)

    #price_distance(address, 0)
    #price_distance(address)
    selfsave(address)

predicted = 0

def predict(address,t):
    self = address
    predict=0
    weight_sum = 0
    if(t<address.nCycles):
     for ag in address.modelSwarm.getAgentList():
        predict += ag.priceList[t]/ag.p_step
        weight_sum += 1./ag.p_step
     predict = predict/weight_sum
     return predict
    else:
```

```
     print('error: t is greater than total simulation time ')


def predicted_price(address):
    self = address
    global predicted
    weight_sum = 0

    for ag in address.modelSwarm.getAgentList():
        predicted += ag.priceList[0]/ag.p_step
        weight_sum += 1./ag.p_step
    predicted = predicted/weight_sum
    return predicted


def show_purchases(address):
    self = address

    asc = []
    ordi = []

    for prc in address.modelSwarm.worldState.purchaseList:
       asc.append(prc[0])
       ordi.append(prc[1])
    asc= np.array(asc)
    ordi = np.array(ordi)
    plt.title('price of exchange')
    plt.ylabel('price')
    plt.xlabel('time')
    plt.plot(asc,ordi,'*')
    plt.show()


    num_epoc = int(address.nCycles)
    t = np.arange(num_epoc)
    n_enc = np.zeros(num_epoc)
    time = 0
    while (time < len(t)):
       for enc in address.modelSwarm.worldState.meetingList:
           if(time == enc):
              n_enc[time] += 1
           if(time<enc):
              break
       time +=1

    plt.plot(t,n_enc)
    plt.title('number of encounters')
    plt.ylabel('encounters')
    plt.xlabel('time')
    plt.show()
```

```
    num_camp = int(num_epoc/100)
    t_camp = np.arange(num_camp)
    n_encmed = np.zeros(num_camp)
    pmed = np.zeros(num_camp)
    q =np.zeros(num_camp)
    j=0
    for i in range(num_camp):
        n=0

        while(j<100*(i+1)):
                n_encmed[i] += n_enc[j]
                j += 1

    j=0
    n_purchase=len(asc)
    for i in range(num_camp):
        n=0

        while(j< n_purchase and 100*i<= asc[j] and  asc[j]<100*(i+1)):
                pmed[i] += ordi[j]
                n +=1
                j +=1
        q[i]=n
        if(n>0):
            pmed[i]=pmed[i]/n
    success= np.zeros(num_camp)
    for i in range(num_camp):
       success[i] = q[i]/n_encmed[i]

    plt.plot(t_camp,success)
    plt.title('percentage of purchase over encounters ')
    plt.ylabel('percentage')
    plt.xlabel('time')
    plt.show()

    #plt.plot(t,pmed)
    fig = plt.figure()
    bx = fig.add_subplot(2,1,1)
    bx2= fig.add_subplot(2,1,2)
    bx.plot(t_camp,pmed,'*')
    bx2.plot(t_camp,q)
    #plt.plot(t,q)
    plt.show()


def plot_link_number(address):

    self = address
```

```
    for ag in address.modelSwarm.getAgentList():
        if(type(ag) is Producer):
            #i+=1
            plt.bar(ag.number,len(ag.mycList), color = 'b')

    plt.title('number of distinct consumers')
    plt.ylabel('encounters')
    plt.xlabel('producer id')
    plt.show()


def prod_position(address):

    self = address

    fig = plt.figure()
    ax = fig.add_subplot(1,1,1,aspect ='equal')
    for ag in address.modelSwarm.getAgentList():
        if(type(ag) is Producer):
            ax.add_patch(patches.Circle((ag.xPos,ag.yPos),
                      ag.radius, facecolor = 'w', edgecolor = 'Brown'))

    plt.title('Producers position')
    plt.xlim(xmin=-1.2,xmax=1.2)
    plt.ylim(ymin=-1.2,ymax=1.2)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.draw()
    #plt.show()



def cons_position(address,t):
    self = address

    asc = []
    ordi = []

    for ag in address.modelSwarm.getAgentList():
        if(type(ag) is Consumer):
            asc.append(ag.x[t])
            ordi.append(ag.y[t])

    asc = np.array(asc)
    ordi = np.array(ordi)
    plt.plot(asc,ordi,'*')
    title = 'Consumers position at time =' + str(t)
    plt.title(title)
    plt.xlim(xmin=-1.2,xmax=1.2)
```

```
    plt.ylim(ymin=-1.2,ymax=1.2)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()


def equilibrium_function(address):

    time = np.arange(address.nCycles-1)
    eqdist = np.zeros(address.nCycles-1)
    Nag = len(address.modelSwarm.getAgentList())
    for t in range(address.nCycles-1):
        for ag in address.modelSwarm.getAgentList():
            eqdist[t] += pow(ag.priceList[t]- predicted ,2)
        eqdist[t] = eqdist[t]/Nag
        eqdist[t] = np.sqrt(eqdist[t])


    plt.plot(time,eqdist)
    plt.title('dispersion around the equilibrium value')
    plt.xlabel('time')
    plt.ylabel('d^2')
    plt.show()


def mean_pprice(address,t=-1):
    cprice =[]
    for cs in address.modelSwarm.getAgentList():
        if(type(cs) is Producer):
            cprice.append(cs.priceList[t])
    cprice = np.array(cprice)
    return (cprice.mean(),cprice.std())


def mean_cprice(address,t=-1):
    cprice =[]
    for cs in address.modelSwarm.getAgentList():
        if(type(cs) is Consumer):
            cprice.append(cs.priceList[t])
    cprice = np.array(cprice)
    return (cprice.mean(),cprice.std())


def show_mean_producer(address):
    self = address
    asc = np.arange(len(address.modelSwarm.getAgentList()[0].priceList))
    ordi = np.zeros(len(asc))
    ordi2 = np.zeros(len(asc))
    pr = np.zeros(len(asc))
    for i in range(len(asc)):
```

```
        (ordi[i],ordi2[i]) = mean_pprice(address,i)
        pr[i] = predict(address,i)
        #pr[i] -= predicted

    fig1 = plt.figure()
    ax1 = fig1.add_subplot(1,2,1)
    ax1.set_title('Mean producers prices')
    ax2 = fig1.add_subplot(1,2,2)
    ax2.set_title('Standard deviation')
    ax1.plot(asc,ordi)
    ax1.plot(asc, pr)
    ax2.plot(asc,ordi2)
    plt.show()


def show_mean_consumer(address):
    self = address
    asc = np.arange(len(address.modelSwarm.getAgentList()[0].priceList))
    ordi = np.zeros(len(asc))
    ordi2 = np.zeros(len(asc))
    pr = np.zeros(len(asc))
    for i in range(len(asc)):
        (ordi[i],ordi2[i]) = mean_cprice(address,i)
        pr[i] = predict(address,i)

    fig1 = plt.figure()
    ax1 = fig1.add_subplot(1,2,1)
    ax1.set_title('Mean consumers prices')
    ax2 = fig1.add_subplot(1,2,2)
    ax1.plot(asc,ordi)
    ax2.set_title('Standard deviation')
    ax1.plot(asc, pr)
    ax2.plot(asc,ordi2)
    plt.show()


def show_Chistogram(address,t = -1):

    cprice =[]
    for cs in address.modelSwarm.getAgentList():
        if(type(cs) is Consumer):
            cprice.append(cs.priceList[t])
    cprice = np.array(cprice)

    plt.hist(cprice)
    title = 'Consumers reservation price distribution' + str(t)
    plt.title(title)
    plt.show()
```

```
def show_Phistogram(address,t = -1):

    pprice =[]
    for pr in address.modelSwarm.getAgentList():
        if(type(pr) is Producer):
            pprice.append(pr.priceList[t])
    pprice = np.array(pprice)

    plt.hist(pprice)
    title = 'Producers reservation price distribution' + str(t)
    plt.title(title)
    plt.show()


def selfsave(address):

    self = address
    tFile = "BA_(Landscape)/SimulationData/FirstModel/UniqueDistribution/EqualInertia/" +
                input("Insert the file name where you want to save results: ")
    myFile = open(tFile, "wb")
    pk.dump(self.modelSwarm.getAgentList(),myFile)
    myFile.close()

    lc = 0
    lp = 0
    for ag in address.modelSwarm.getAgentList():
        if(type(ag)is Consumer):
           lc += 1
        else:
           lp += 1

    myFile2 = open(tFile+'.txt',"w")
    myFile2.write('seed = ' + str(common.mySeed)+ "\n")
    myFile2.write('Number of iterations = ' + str(address.nCycles) + "\n")
    myFile2.write('time step = ' + str(common.time_step)+ "\n")
    myFile2.write('Number of consumers = ' + str(lc)+ "\n")
    myFile2.write('Consumers diffusion = ' + str(common.diffusion) + "\n")
    myFile2.write('price step = ' + str(common.price_step)+ "\n")
    myFile2.write('Number of producers = ' + str(lp)+ "\n")
    myFile2.write('relative inertia = ' + str(common.relative_inertia)+ "\n")
    myFile2.write('producers radius = '+ str(common.producers_radius)+ "\n")
    myFile2.write('noise = '+ str(common.noise)+ "\n")
    myFile2.write('field decay = '+str(common.decay)+ "\n")
    myFile2.write('field diffusion = ' + str(common.field_diffusion)+ "\n")
    myFile2.write('field box side = '+ "\n")
    myFile2.close()

def otherSubSteps(subStep, address):
    return False
```

8.3 Results

In our simulations we have mainly dealt with periodic boundary conditions and with producer occupying the nodes of a regular square lattice. However, we have also considered reflecting walls as boundary conditions and random producers initial position (taken from a uniform distribution over the square) to assure we are not introducing artifacts (see section "ABM drawbacks").

Consumer position are taken randomly from a uniform distribution over the square. We have chose two different type of price initial condition. In the first, all the agents' prices are picked from a Gaussian distribution. In the second, agents' prices are picked from two Gaussian distribution with different mean, one for producers and the other for consumers. All consumers update their prices by the same amount $\lambda_i = \lambda_c \forall i$; producers' price step too are all equal, $\lambda_j = \lambda_p \forall j$, but the case $\lambda_c = \lambda_p$ is only one of those we have treated.

We have performed simulations varying the numbers of consumers ($N_c = 20, 50$), the number of producers ($N_p = 10, 16, 25$), consumers' price step ($\lambda_c = 0.3, 0.15, 0.03$), the ratio between the price step of consumers and producers ($\frac{\lambda_c}{\lambda_p} = 1, 10^{-1}, 10^{-2}, 10^{-3}$), producers' radius ($r_p = 10^{-1}, 5 * 10^{-2}$) and diffusion coefficient ($D = 1, 10^{-1}, 10^{-2}, 5 * 10^{-2}, 10^{-3}$). We have maintain fixed the side of the world $L = 2$ and the time step $\Delta t = 10^{-1}$.

Results:

- Simulations confirm the existence of equilibrium: prices of agents evolve towards a common value, given by equation (8.17). Figure 8.3 shows graph of purchases price for various simulations; the time, measured in iteration step, is reported on the x-axis, prices at which exchanges are done are depicted on y-axis (Note that prices of interaction where the consumer does not buy are not reported in these graphs). The graphs shown are relative only to simulations where consumers' and producers' price step are the same, i.e. $\lambda_c = \lambda_p$. All the graphs but the last one are relative to simulation with the first type of initial condition for prices, i.e. agents' price come from a unique Gaussian distribution for both producers and consumers; the last one is referred to the second type of initial condition.

  After a initial period, exchanges are all done at similar prices, confirming Hayek hypothesis.

- Equilibrium occurs when the mean square displacement from the final price (the function (8.17)) is of the same order of the biggest price step. Figure 8.4 shows standard deviation of the prices of all agents (which for $\lambda_c = \lambda_p$ is equal to the function (8.21)) for the same simulation of figure 8.3.

- We find that the mean number of encounters is independent from the diffusion coefficent and it is roughly given by the equation

$$\mu = N_c N_p \frac{\pi r_p^2}{L^2} \tag{8.25}$$

  as shown by the following table, whose values are relative again to the same simulations of figure 8.3.

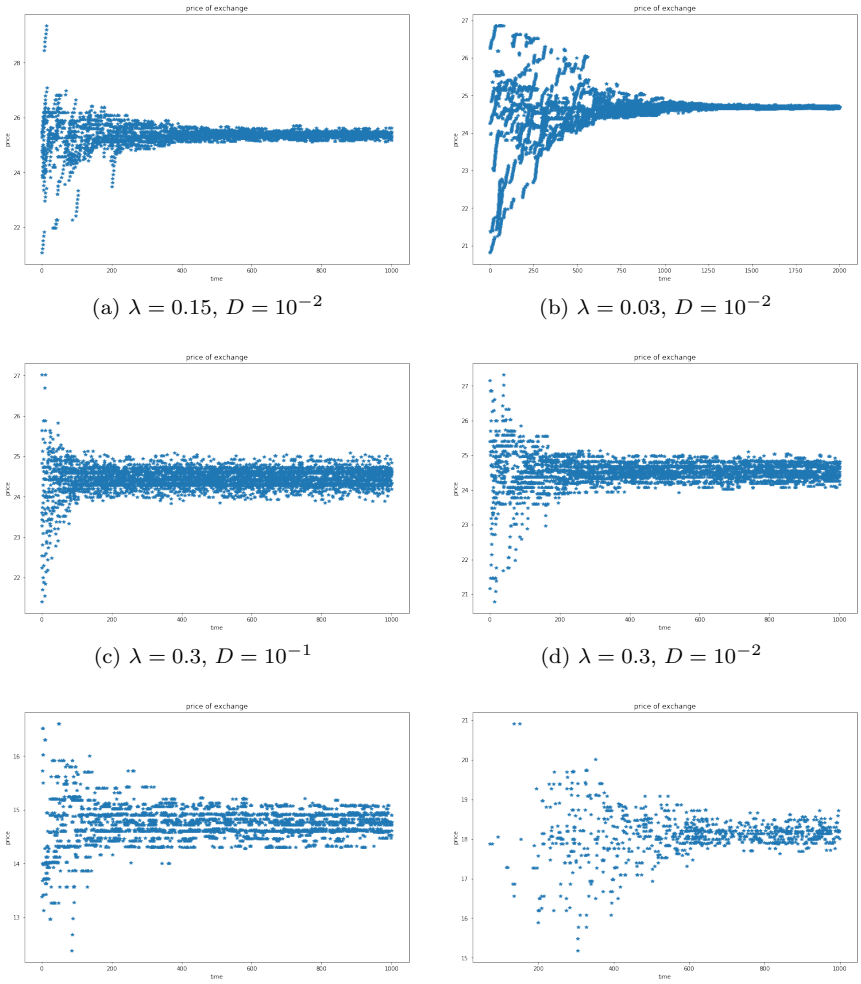
Encounters	Predicted	D
9.53	9.82	0.01
9.43	9.82	0.01
9.96	9.82	0.1
9.26	9.82	0.01
6.32	6.28	0.01
9.43	9.82	0.001

Figure 8.5 shows the number of encounters in each time step against iteration time. We can see that changing D does not change the mean value, it influences the variance only.

- The time of convergence is a decreasing function of λ and r_p. See figure 8.6
- The time of convergence is a decreasing function of D. See figure 8.7
- Decreasing the ratio between the price step of producers and consumers increase the time of convergence, see figure 8.8.

  When the ratio $\frac{\lambda_p}{\lambda_c} \ll 1$, consumers prices are fast variables and adapt to producers prices. In figure 8.9 are shown the histograms of consumer prices relative to a simulation where we use an initial condition of the second type describe at the beginning of this section, i.e. producers prices are distributed according to a Gaussian with a mean greater than that of consumers' price distribution. This latter price distribution is concentrated around a mean which is more similar to producers' mean more we decrease the ratio $\frac{\lambda_p}{\lambda_c}$.

  Then, many exchanges are done at the initial price of producers. Although this system will converge to a unique price in the long run, exchanges are performed out of equilibrium. Since we are modeling an agricultural market, and since we are not considering birth or death processes we may consider constant the quantity bought by consumers in a given time interval. So our simulation time has to be related to the number of purchases. So we can not say that market has reached equilibrium if more than 1000 purchases are done at different prices.



(a) $\lambda = 0.15$, $D = 10^{-2}$

(b) $\lambda = 0.03$, $D = 10^{-2}$

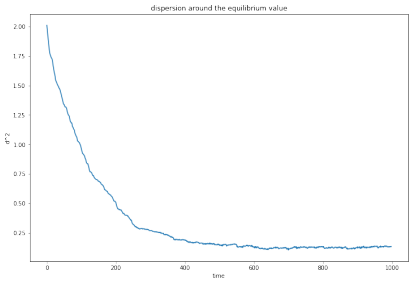
(c) $\lambda = 0.3$, $D = 10^{-1}$

(d) $\lambda = 0.3$, $D = 10^{-2}$

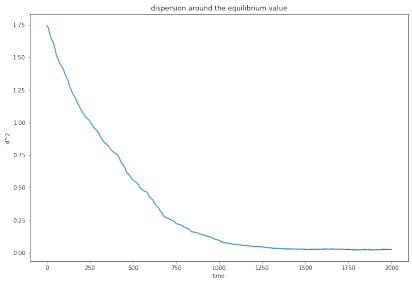
(e) $\lambda = 0.3$, $D = 10^{-2}$, $N_p = 16$

(f) $\lambda = 0.3$, $D = 10^{-2}$

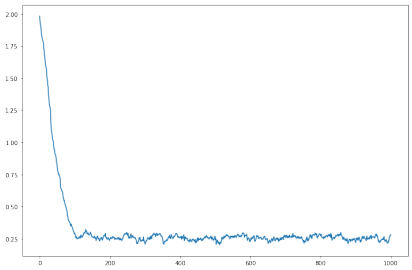
Figure 8.3: Exchange prices. $N_c = 50$ and $N_p = 25$ for all the simulations but the fifth, for which $N_p = 16$.



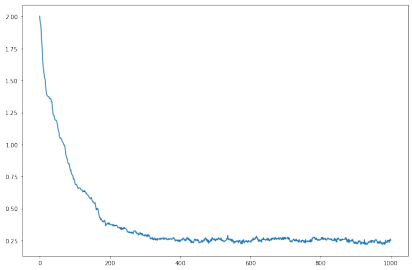
(a) $\lambda = 0.15$, $D = 10^{-2}$



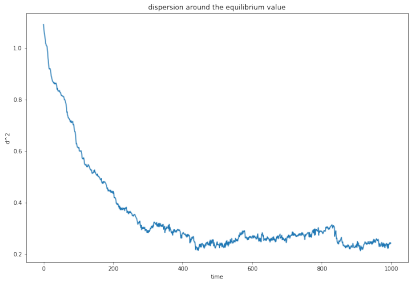
(b) $\lambda = 0.03$, $D = 10^{-2}$



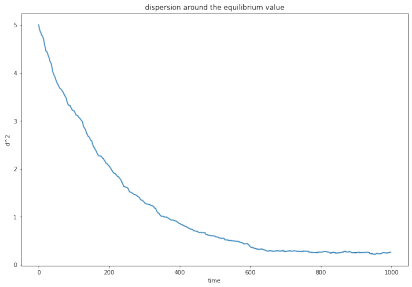
(c) $\lambda = 0.3$, $D = 10^{-1}$



(d) $\lambda = 0.3$, $D = 10^{-2}$

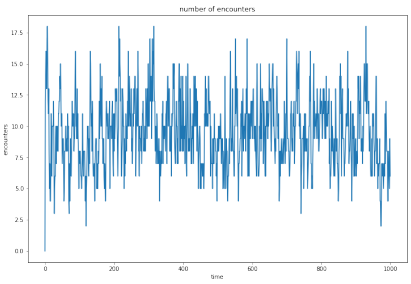


(e) $\lambda = 0.3$, $D = 10^{-2}$, $N_p = 16$

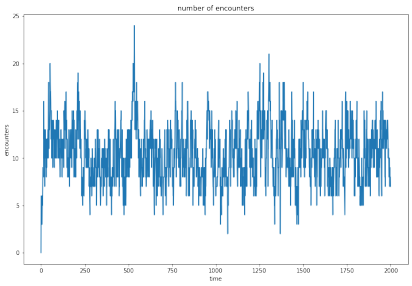


(f) $\lambda = 0.3$, $D = 10^{-2}$

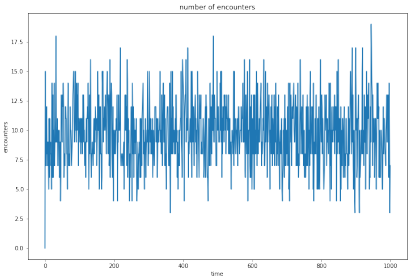
Figure 8.4: Standard deviation.



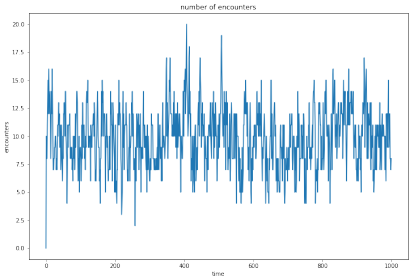
(a) $\lambda = 0.15$, $D = 10^{-2}$



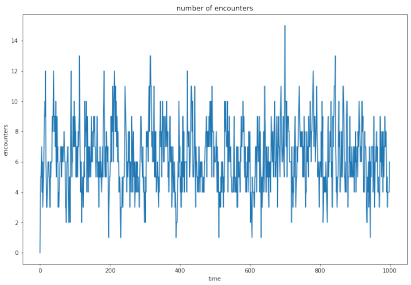
(b) $\lambda = 0.03$, $D = 10^{-2}$



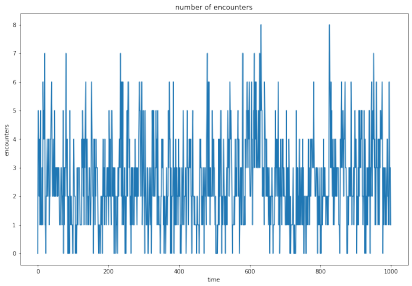
(c) $\lambda = 0.3$, $D = 10^{-1}$



(d) $\lambda = 0.3$, $D = 10^{-2}$

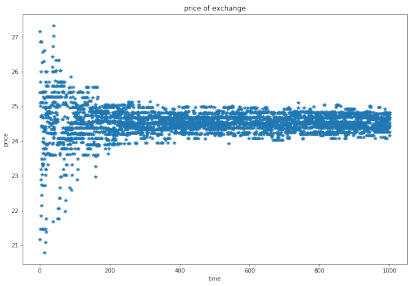


(e) $\lambda = 0.3$, $D = 10^{-2}$, $N_p = 16$

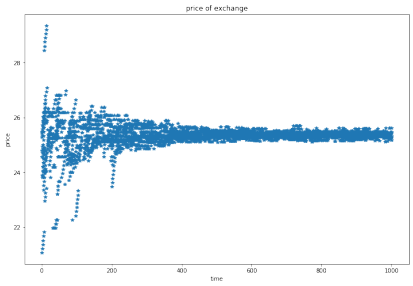


(f) $\lambda = 0.3$, $D = 10^{-2}$

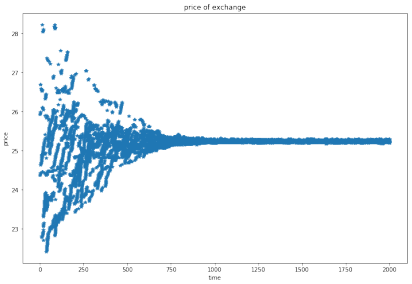
Figure 8.5: Number of encounters per iteration.



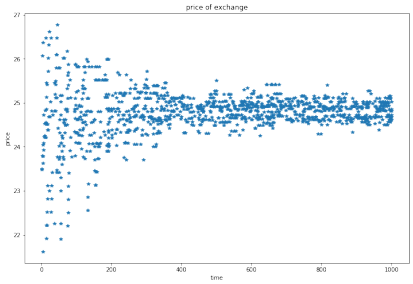
(a) $\lambda = 0.3$, $r_p = 0.1$



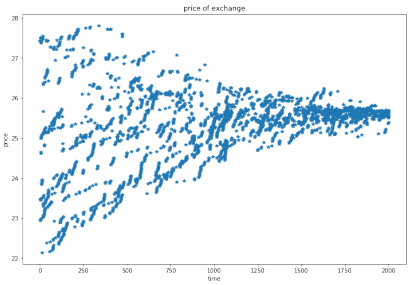
(b) $\lambda = 0.15$, $r_p = 0.1$



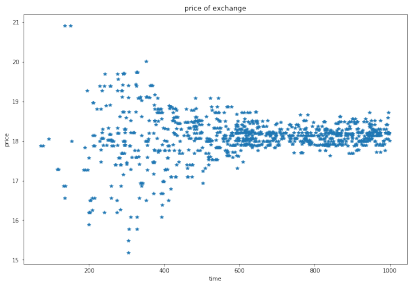
(c) $\lambda = 0.03$, $r_p = 0.1$



(d) $\lambda = 0.3$, $r_p = 0.05$

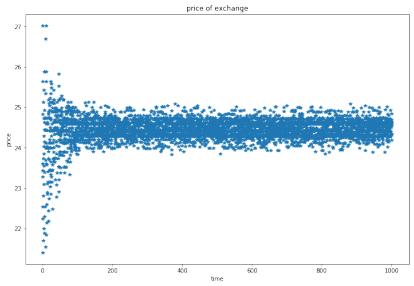


(e) $\lambda = 0.3$, $r_p = 0.05$

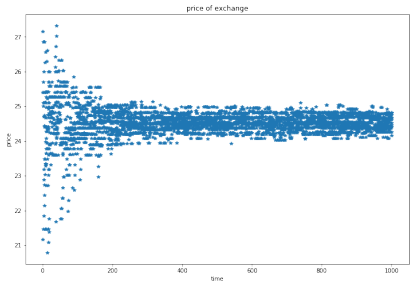


(f) $\lambda = 0.3$, $r_p = 0.1$

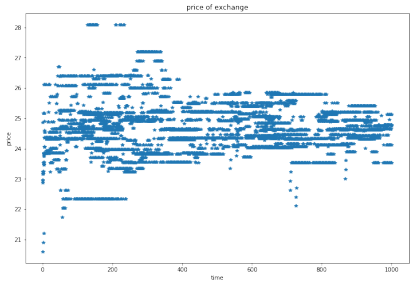
Figure 8.6: Exchange price. For all $D = 10^{-2}$



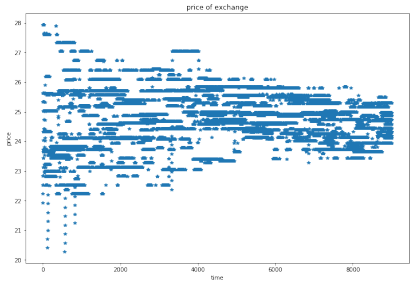
(a) $\lambda = 0.3$, $D = 10^{-1}$



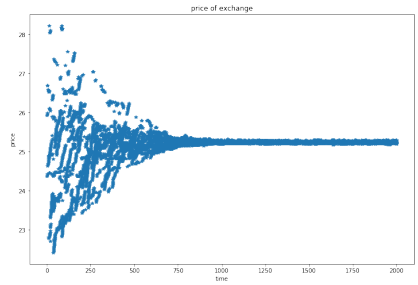
(b) $\lambda = 0.3$, $D = 10^{-2}$



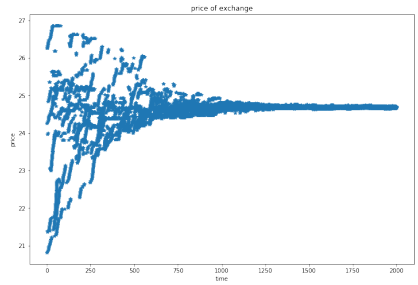
(c) $\lambda = 0.3$, $D = 10^{-3}$



(d) $\lambda = 0.3$, $D = 10^{-4}$

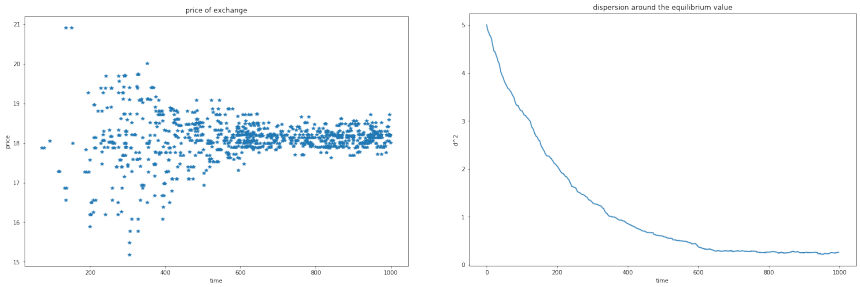


(e) $\lambda = 0.03$, $D = 10^{-2}$



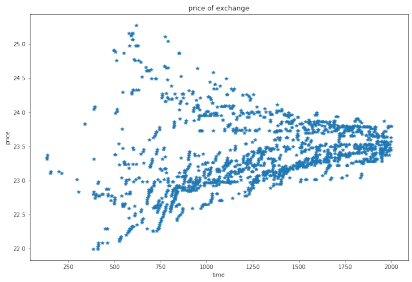
(f) $\lambda = 0.03$, $D = 5 * 10^{-3}$

Figure 8.7: Exchange price.For all $r_p = 0.1$

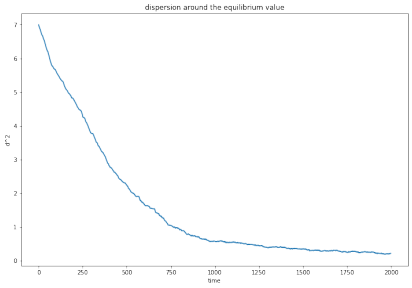


(a) $\frac{\lambda_p}{\lambda_c} = 1$

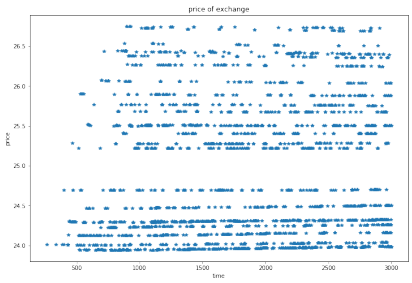
(b) $\frac{\lambda_p}{\lambda_c} = 1$



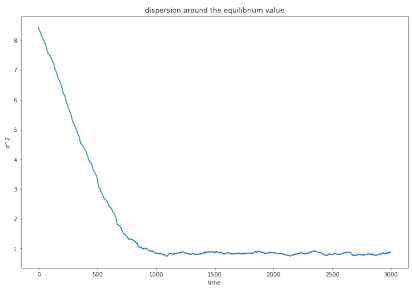
(c) $\frac{\lambda_p}{\lambda_c} = 0.1$



(d) $\frac{\lambda_p}{\lambda_c} = 0.1$

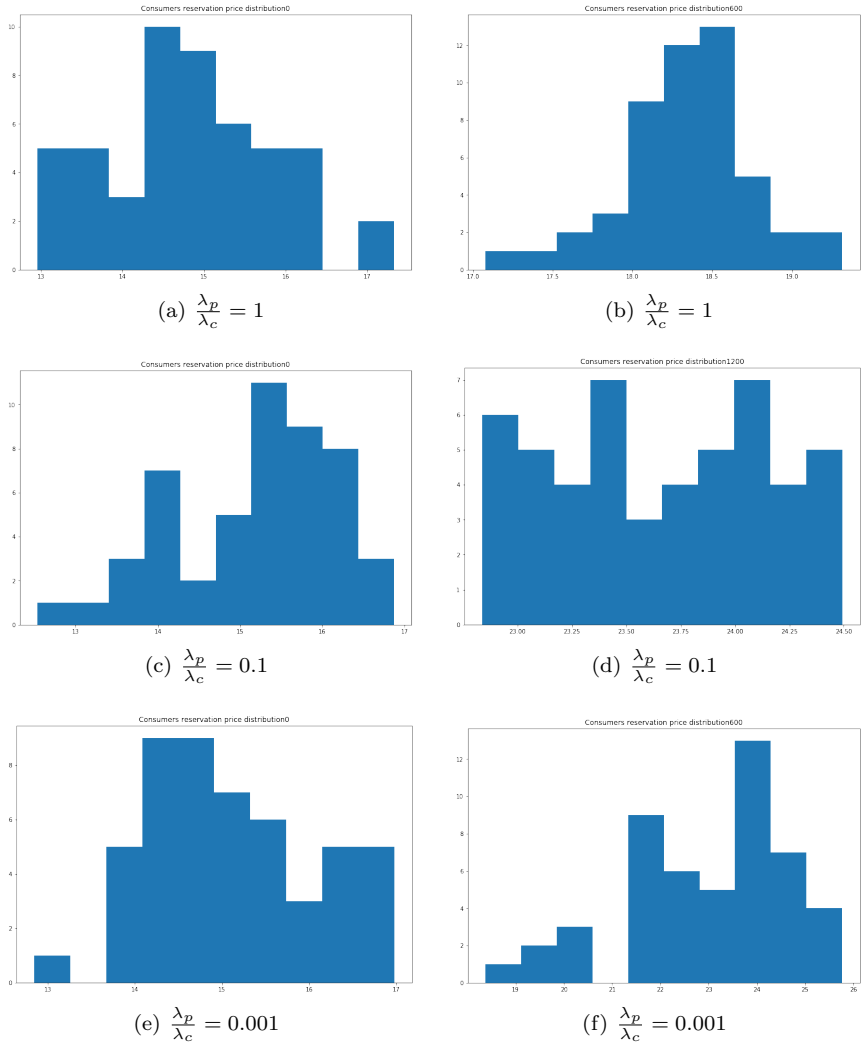


(e) $\frac{\lambda_p}{\lambda_c} = 0.001$



(f) $\frac{\lambda_p}{\lambda_c} = 0.001$

Figure 8.8: Exchange price. For all we have $\lambda_c = 0.3$ and $r_p = 0.05$



(a) $\frac{\lambda_p}{\lambda_c} = 1$ (b) $\frac{\lambda_p}{\lambda_c} = 1$

(c) $\frac{\lambda_p}{\lambda_c} = 0.1$ (d) $\frac{\lambda_p}{\lambda_c} = 0.1$

(e) $\frac{\lambda_p}{\lambda_c} = 0.001$ (f) $\frac{\lambda_p}{\lambda_c} = 0.001$

Figure 8.9: Consumers' price distribution. For all we have $\lambda_c = 0.3$ and $r_p = 0.05$

Chapter 9

Second model: imitation

9.1 Model description

It is a common practice in socio economical simulations to consider imitation between agents. Then, in the second model we have considered direct interaction between consumers, allowing them to imitate each other.

We have considered two different schemes of imitation, both are based on physical proximity: a consumer can imitate another only when the distance between the two agents is less than a given quantity. We interpret this quantity as the radius of the circle representing the field of vision of each consumer.

According to the first scheme, the targets of imitators are only those consumers that have bought in the previous step. In the second we allow consumers to imitate also refutation of producers' offer. In both schemes, the imitating consumer copies the reservation price of her target. We imagine that the reservation price is directly communicated by the target after a request of the imitator.

With respect to the first model the weighted sum of the prices is not conserved. To show this we focus on three agent: the target consumer, the producer with which the target has interacted, and second consumer that imitate the first and we indicate their reservation prices respectively with θ_T, θ_j and θ_i. Figure 9.1 shows the changing of prices after the first interaction involving only the target and the producers (green and red circles) and after the imitation.

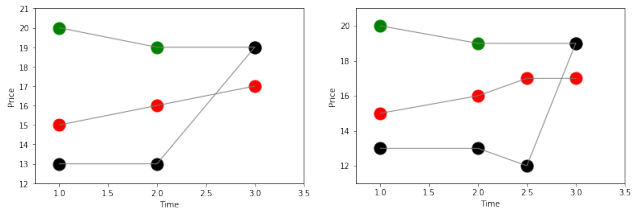


Figure 9.1: Imitation consequence
The green circle represents the target consumer's price, the red circle that of producer, and the black one represents the imitating consumer's price.

After the first interaction, where the target consumer has bought, we have

$$\theta_T(t+1) = \theta_T(t) - \lambda_T \tag{9.1}$$
$$\theta_j(t+1) = \theta_j(t) + \lambda_j \tag{9.2}$$
$$\theta_i(t+1) = \theta_i(t) \tag{9.3}$$

After the second, where the imitating consumer too buys from the same producer

$$\theta_T(t+2) = \theta_T(t+1) \tag{9.4}$$
$$\theta_j(t+2) = \theta_j(t+1) + \lambda_j \tag{9.5}$$
$$\theta_i(t+2) = \theta_T(t+1) \tag{9.6}$$

So we obtain for the differences $\Delta\theta_x = \theta_x(t+2) - \theta_x(t)$

$$\Delta\theta_T = -\lambda_T \tag{9.7}$$
$$\Delta\theta_j = 2\lambda_j \tag{9.8}$$
$$\Delta\theta_i = \theta_T(t) - \theta_i(t) - \lambda_T \tag{9.9}$$

Finally we find for the quantity θ^H defined in eq.(8.17)

$$\Delta\theta^H = \frac{\theta_T(t) - \theta_i(t)}{\lambda_i} \tag{9.10}$$

where we have used the assumption $\lambda_T = \lambda_i$.

For the second scheme of imitation we expect that, on average, successive variation of θ^H cancels out, but not for the first scheme, since imitation occurs only for purchases.

9.2 Code

For the second model we have add methods only to the class "Consumer". The additional data methods are

- imitate_old: If there are consumers inside a circle of radius "sradius" (data member of the class described in section 8.2) ask to that consumer the value of her variables "bought" and "mySeller". If "bought" is one imitation take place: through the target consumer's variable "mySeller", the imitator retrieve the producer that has sold to the target, and she buys from that producer.

- imitate_old2: the same of the previous function but imitation occurs also when "bought" is equal -1, i.e. when the target consumer has interacted with a producers but she has refused producer's offer.

- reset: restore to 0 the variables "bought" and "mySeller".

We report the relative code below.

```
    def imitate_old(self):

        copyList = self.agentList.copy()
        np.random.shuffle(copyList)

        for consumer in copyList:
            if(type(consumer) is not Producer and not self.number == consumer.number):

             distance = pow((self.xPos-consumer.xPos),2) + pow((self.yPos-consumer.yPos),2)
             distance = np.sqrt(distance)
             if(distance <= self.sradius):
                  #print('I am trying to imitate...')
                  if(consumer.bought == 1):
                     # print(self.number, "imitating", consumer.number)
                      self.price = consumer.price


                      consumer.myseller.change(consumer.bought)

                      self.myWorldState.purchaseList.append((common.cycle,
                                consumer.last_purchase))
                  break


    def imitate_old2(self):

        copyList = self.agentList.copy()
        np.random.shuffle(copyList)

        for consumer in copyList:
            if(type(consumer) is not Producer and not self.number == consumer.number):

             distance = pow((self.xPos-consumer.xPos),2) + pow((self.yPos-consumer.yPos),2)
             distance = np.sqrt(distance)
             if(distance <= self.sradius):
                  #print('I am trying to imitate...')
                  if(not consumer.bought == 0):
                     # print(self.number, "imitating", consumer.number)
                      self.price = consumer.price


                      consumer.myseller.change(consumer.bought)
                      if(consumer.bought == 1):
                        self.myWorldState.purchaseList.append((common.cycle,
                              consumer.last_purchase))
                      break

    def reset(self):

        self.myseller = 0
```

```
self.bought = 0
```

The event schedule is:

	A	B	C	D
1	#	1		
2	WorldState	computationalUse	buildLandscape	
3				
4	#	1	100000	
5	tasteA	evolve		
6	tasteA	mantain_pbc		
7	tasteA	record_position		
8				
9	tasteA	search_producerV2		
10				
11	tasteA	0,5	imitate_old	
12				
13	all	record		
14	tasteA	reset		
15				

The two differences between the schedule of the first model are the instruction "imitate_old" and "reset". The number written on the left of "imitate_old" instruction represent the probability with which the method is called.

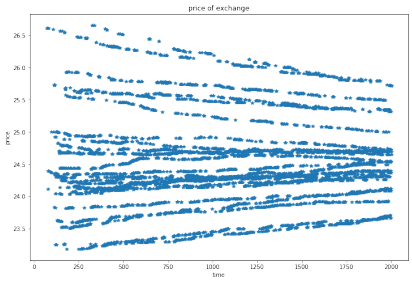
9.3 Results

In all the simulations of the second model producers and consumers initial prices are picked from two Gaussian with different mean; the mean of the producers Gaussian is greater than that of consumers. We have used only periodic boundary condition. Unless otherwise specified we have used $N_c = 50$ and $N_p = 25$.

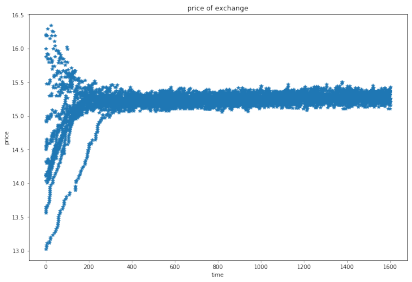
In the following r_c stands for consumers radius and p the probability with which the imitation methods is activated.

Results:

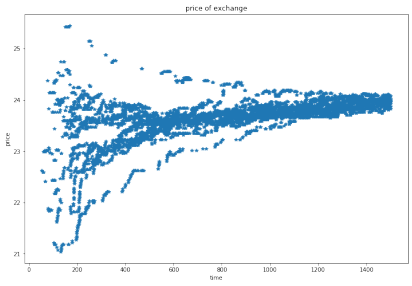
- Using the first imitation scheme we see that exchange prices grow over time(figure 9.2). As said in the model description, this is due to a mean increase in θ^H. This effect is more pronounced with the increasing of r_c or p. In figure 9.3 we can see that agents' prices follow θ^H variations.
- In the second scheme instead we see only fluctuations of the purchase price (figure 9.4), since θ^H does not change on average (orange lines in figure 9.5). We can observe only fluctuations around the mean. Bigger are r_c or p, bigger are fluctuations.



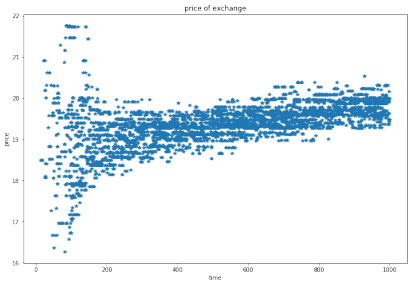
(a) $\lambda_r = 0.01$, $r_c = 0.1, p = 0.05$



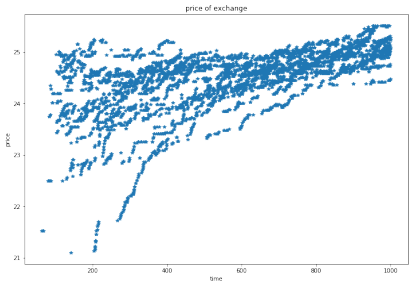
(b) $\lambda_r = 0.1$, $r_c = 0.1, p = 0.05$



(c) $\lambda_r = 0.1$, $r_c = 0.1, p = 0.5$



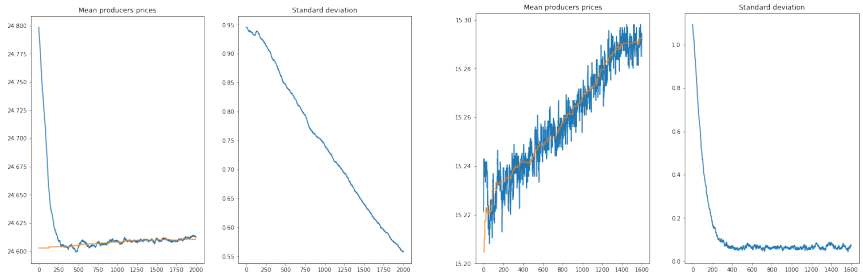
(d) $\lambda_r = 1$, $r_c = 0.1, p = 0.5$



(e) $\lambda_r = 0.1$, $r_c = 0.1, p = 0.5$

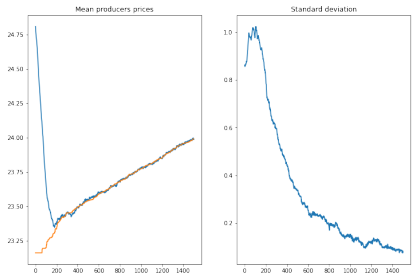
(f) $\lambda_r = 0.1$, $r_c = 0.2, p = 0.5$

Figure 9.2: Exchange price. For all the simulation we have $\lambda_c = 0.3$ and $D = 0.01$

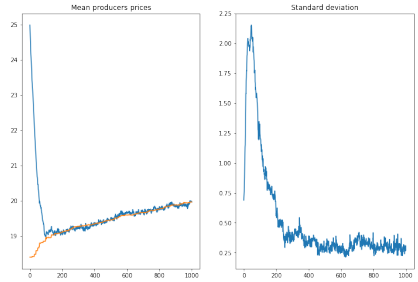


(a) $\lambda_r = 0.01$, $r_c = 0.1, p = 0.05$

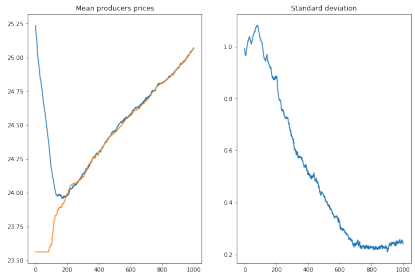
(b) $\lambda_r = 0.1$, $r_c = 0.1, p = 0.05$



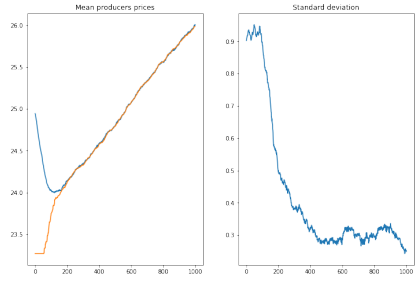
(c) $\lambda_r = 0.1$, $r_c = 0.1, p = 0.5$



(d) $\lambda_r = 1$, $r_c = 0.1, p = 0.5$

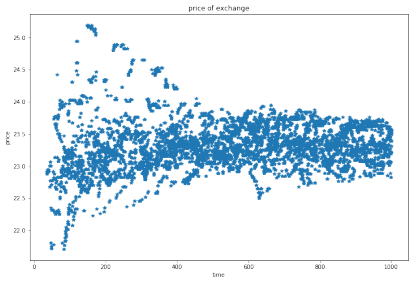


(e) $\lambda_r = 0.1$, $r_c = 0.1, p = 0.5$

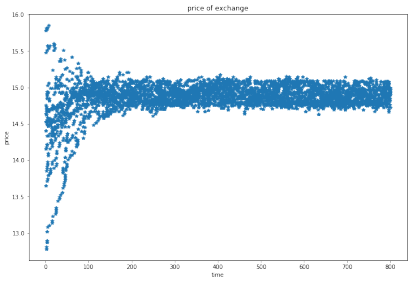


(f) $\lambda_r = 0.1$, $r_c = 0.2, p = 0.5$

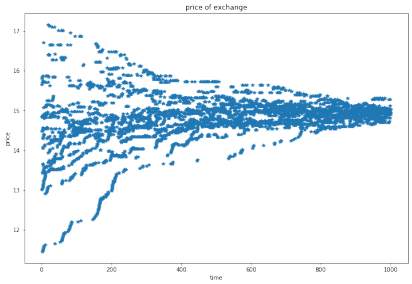
Figure 9.3: Mean producers price. The orange line represents the equilibrium value θ^H of the first model. For all the simulation we have $\lambda_c = 0.3$ and $D = 0.01$



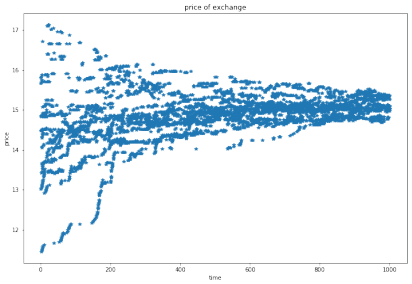
(a) $\lambda_r = 0.1,\ D = 0.01, p = 1$



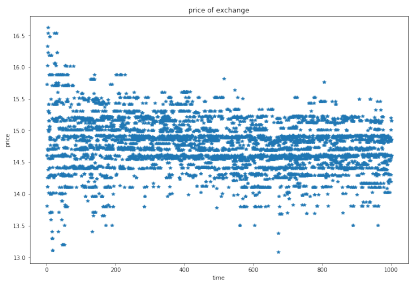
(b) $\lambda_r = 0.1,\ D = 0.1, p = 0.5$



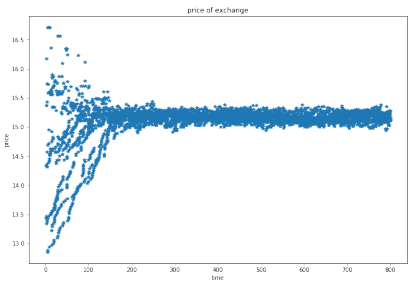
(c) $\lambda_r = 0.1,\ D = 0.01, p = 0.05$



(d) $\lambda_r = 0.1,\ D = 0.01, p = 0.5$

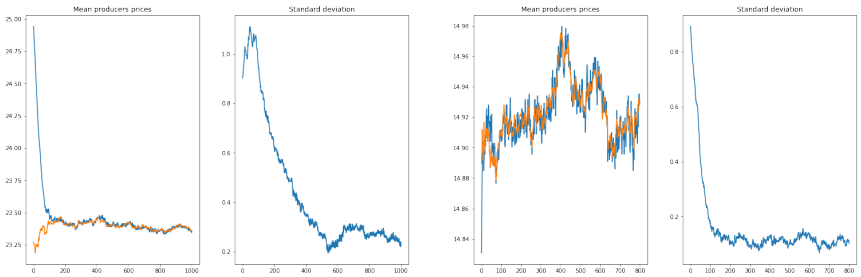


(e) $\lambda_r = 1,\ D = 0.01, p = 0.5$



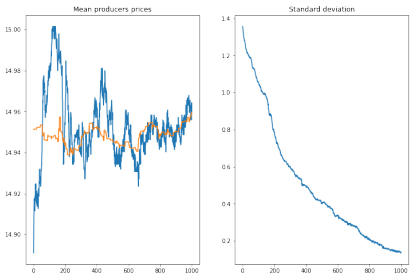
(f) $\lambda_r = 0.1,\ D = 0.1, p = 0.5$

Figure 9.4: Exchange price. For all the simulation we have $\lambda_c = 0.3$ and $r_c = 0.1$

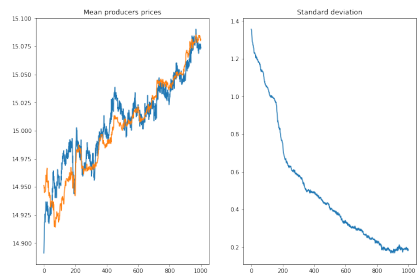


(a) $\lambda_r = 0.1,\ D = 0.01, p = 1$

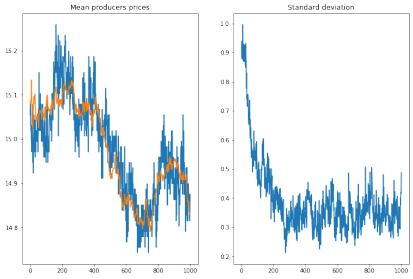
(b) $\lambda_r = 0.1,\ D = 0.1, p = 0.5$



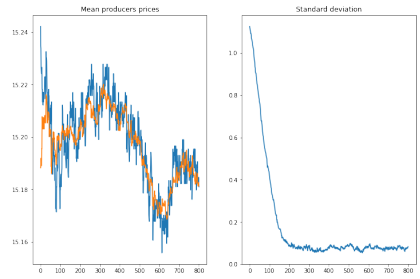
(c) $\lambda_r = 0.1,\ D = 0.01, p = 0.05$



(d) $\lambda_r = 0.1,\ D = 0.01, p = 0.5$



(e) $\lambda_r = 1,\ D = 0.01, p = 0.5$



(f) $\lambda_r = 0.1,\ D = 0.1, p = 0.5$

Figure 9.5: Mean producers price. For all the simulation we have $\lambda_c = 0.3$ and $r_c = 0.1$

Chapter 10

Third model: Motion-trade feedback

10.1 Model description

In our models, consumers motion represents their search activity. With the previous two models we have considered the influence of motion characteristics on the trading activity, but we have forbidden the opposite way. The purpose of the third model is to study the feedback between consumers' search activity and their motion. To this end we will use the adaptive landscape tool mentioned in the third section of the chapter on BA.

We deal with a one component field with a slight modification: we assume that the field is not constantly generated by agents' variable, but it is updated by consumers only when they interact with producers. Simulations start with a plain landscape (it is null in every point). When a consumer buy from a producer she decreases the field value relative to its position creating a valley; when she refuses producer's offer she increases the field value creating a mountain.

Since consumers experience a force proportional to the field gradient changed of sign, a valley will attract sufficiently close consumers; they, in turn, will buy from the producer (if her price is sufficiently low) in the valley, and they will update the local field value, deepening the valley. The opposite happens for mountains. Then, we expect that once a producer has sold, for all the time that her price remains lower than those of the nearest consumers, she will sell more and more, thanks to the interaction between consumers and field.

However, the field influence strongly depend on its eigendynamic parameters and on consumers motion. The landscape evolve according to the equation

$$\frac{\partial h(\overrightarrow{r},t)}{\partial t} = \sum_{i=1}^{N_c} s_i \alpha_i(t) \delta(\overrightarrow{r} - \overrightarrow{r_i}(t)) - k_h h(\overrightarrow{r},t) + D_h \nabla^2 h(\overrightarrow{r},t) \qquad (10.1)$$

Here, $\alpha_i(t)$ is a variable representing the action of the i-th consumer; it is equal to -1 when the consumer has bought at time t, it is 1 when she refuses producer offer and it is equal to 0 when the consumer has not interacted with producers. We maintain that s_i represents the strength with which the i-th consumer influences the field.

As explained in the chapter on BA, the second term represents a decay process of the field value and the third one its diffusion through space. They both determine the lifetime of field dishomogeneity, but while the decay term determines only valleys extinction, the diffusion term influence depend on the value of the diffusion coefficient. On one hand, if it is too high it determines the extinction of gradients; on the other end, if it is too low, the field has limited influence on consumers motion since a valley, no matter how deep, remains confined to a small region.

To observe field structure formation with a significant lifetime the decay rate has to be smaller than the rate at which consumers interact with producers updating the field. Further, the diffusion coefficient of the field have to be high enough to diffuse information (to influence consumers' future actions) in a smaller time compared with the decay time. But also it has to be sufficiently small: the time needed to flatten the landscape has to be higher than the mean time between two interactions. Finally, the strengths s_i have to be high enough to lead non negligible gradient field.

10.2 Code

To use the Adaptive Landscape we have to discretize the space where agents live and we have to assign a number to each box of the partition. We have represented the landscape as a matrix whose entries are in correspondence with the boxes of the world partition. To better present this class we report below the python notebook that we have used to test class functioning.

The data members of the class are:

- L: a number representing the side of the square world.
- N: the number of boxes in a side of the square.
- dec: a number representing the decay term k_h
- diff: a number representing the diffusion term D_h
- grid_size: a number, $\frac{L}{N}$, representing the side of partition boxes.
- t_step: the time step used by the method "evolve"

Its methods are:

- get_indices: takes two float (representing a point) as argument and return the matrix indices associated with the box that includes that point.
- get_field: return the current value of the field at the position given.
- gradient: calculate the gradient of the field through the formulas

$$\frac{\partial h}{\partial x} = \frac{h(x+l,y) - h(x-l,y)}{2*l} \tag{10.2}$$

$$\frac{\partial h}{\partial y} = \frac{h(x,y+l) - h(x,y-l)}{2*l} \tag{10.3}$$

where l=grid_size.

- get_gradient: return the gradient corresponding to the position passed as argument
- evolve: update the filed through the formula

$$h(i,j,t+1) = h(i,j,t)(1-k_h*\Delta t)+step*(h(i-1,j,t)+h(i+1,j,t)+h(i,j-1,t)+h(i,j+1,t)-4*h(i,j,t)) \tag{10.4}$$

  where $\Delta t = t\_step$ and $step = \frac{D\Delta t}{l^2}$.
- mantain_pbc: assure periodic boundary condition
- update: update the field at the position passed as argument by the quantity passed as argument. This method is used by consumer to update the field.
- show2d: plot the field values.

AdaptiveLandscape

April 3, 2018

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
import matplotlib.colors as colors
from mpl_toolkits import mplot3d
```

In [3]:
```
class AdaptiveLandscape():
    def __init__(self,size, Npoints, decay, diffusion, time_step):

        self.L = size
        self.N = Npoints
        self.dec = decay
        self.diff = diffusion
        self.t_step = time_step

        self.grid_size = size/(Npoints)
        self.h = np.zeros((self.N+2,self.N+2))


    def set_initial_condition(self):
        z = int((self.N+1)/2)
        self.h[z,z] = 1/self.grid_size
        print(z)



    def get_indices(self,x,y):

        if(x<-1 or x>1 or y<-1 or y>1):
            print('Error, position out of the square of the field')
        #position of the first box (let-top of the square)
        r = np.array([(-self.L+self.grid_size)/2, (self.L-self.grid_size)/2])

        j = int((x-r[0])/self.grid_size)
        i = int((r[1]-y)/self.grid_size)
        rs = r+np.array([j*self.grid_size,-i*self.grid_size])
        if(x - rs[0] >self.grid_size/2):
```

```
            j = j+1
        if(rs[1]-y > self.grid_size/2):
            i = i +1

        return i+1, j+1


    def get_field(self,x,y):
        return self.h[self.get_indices(x,y)]


    def gradient(self,i,j):

        grad = np.zeros(2)
        if(i not in np.arange(1,self.N+1) or j not in np.arange(1,self.N+1) ):
            print('Error, field indices out of range')
            return
        else:
            grad[0] = (self.h[i,j+1] - self.h[i,j-1])/(2*self.grid_size)
            grad[1] = (self.h[i+1,j] - self.h[i-1,j])/(2*self.grid_size)

        return grad


    def get_gradient(self,x,y):
        i,j =self.get_indices(x,y)
        return self.gradient(i,j)



    def evolve(self):
        step = self.diff*self.t_step/pow(self.grid_size,2)

        #To update the field we use an artificial one to avoid confusion between updated
        # and still not updated field value. Information flow in each point can come onl
        h_temp = np.zeros((self.N+2,self.N+2))

        # update of the internal square of the matrix field
        # The first and the last rows and the first and the last columns are not updated
        # they are used to assuring periodic boundary conditions (p.b.c.)
        for i in range(1,self.N+1):
            for j in range(1,self.N+1):
                h_temp[i,j] = (1-self.dec*self.t_step)*self.h[i,j] + step*(self.h[i-1,j]

        self.mantain_pbc()
        self.h = h_temp

    def mantain_pbc(self):
```

```
        # Here we update boarders to assure p.b.c.
        for i in range(1,self.N+1):
            self.h[i,0] = self.h[i,self.N]
            self.h[i,self.N+1] = self.h[i,1]
        for j in range(1,self.N+1):
            self.h[0,j] = self.h[self.N,j]
            self.h[self.N+1,j] = self.h[1,j]


    def update(self,x,y,a):

        self.h[self.get_indices(x,y)] += a



    def show2d(self):
        x = np.array([-self.L/2+self.grid_size*(i) for i in range(self.N+1)])
        y = np.array([self.L/2-self.grid_size*(i) for i in range(self.N+1)])
        X,Y = np.meshgrid(x,y)
        Z = self.h[1:self.N+1,1:self.N+1]
        #print(x)
        #print(np.shape(Z))
        zm = 1 #Z.max()#0.1
        #if(Z.max()>1): zm=Z.max()
        #levels = MaxNLocator(nbins=30).tick_values(-zm, zm)


        # pick the desired colormap, sensible levels, and define a normalization
        # instance which takes data values and translates those into levels.
        cmap = plt.get_cmap('PiYG')#cm.coolwarm
        norm = colors.SymLogNorm(linthresh=0.03,vmin=Z.min(), vmax=Z.max())#BoundaryNorm

        #plt.subplot(2, 1, 1)
        im = plt.pcolormesh(X, Y, Z, cmap=cmap, norm=norm)
        plt.colorbar()
        plt.show()
```

1 Testing wright pbc

```
In [4]: h = AdaptiveLandscape(2,5,0,1, 0.0001)
```

```
In [5]: for i in range(h.N):
            for j in range(h.N):
                h.h[i+1,j+1]  = i*h.N + j +1

        h.h
```

```
Out[5]: array([[  0.,   0.,   0.,   0.,   0.,   0.,   0.],
               [  0.,   1.,   2.,   3.,   4.,   5.,   0.],
               [  0.,   6.,   7.,   8.,   9.,  10.,   0.],
               [  0.,  11.,  12.,  13.,  14.,  15.,   0.],
               [  0.,  16.,  17.,  18.,  19.,  20.,   0.],
               [  0.,  21.,  22.,  23.,  24.,  25.,   0.],
               [  0.,   0.,   0.,   0.,   0.,   0.,   0.]])

In [6]: h.mantain_pbc()
        h.h

Out[6]: array([[  0.,  21.,  22.,  23.,  24.,  25.,   0.],
               [  5.,   1.,   2.,   3.,   4.,   5.,   1.],
               [ 10.,   6.,   7.,   8.,   9.,  10.,   6.],
               [ 15.,  11.,  12.,  13.,  14.,  15.,  11.],
               [ 20.,  16.,  17.,  18.,  19.,  20.,  16.],
               [ 25.,  21.,  22.,  23.,  24.,  25.,  21.],
               [  0.,   1.,   2.,   3.,   4.,   5.,   0.]])
```

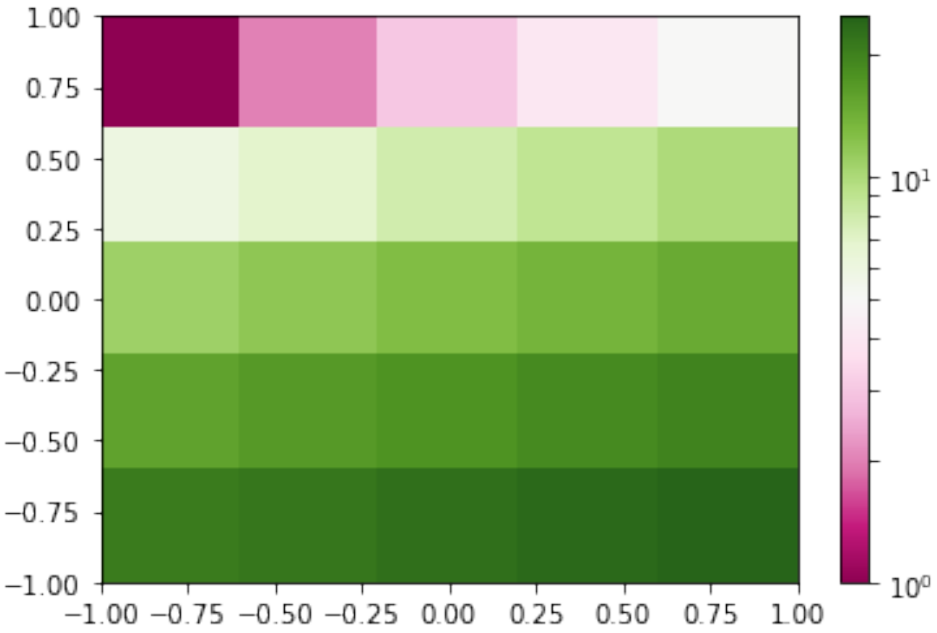
2 Test of indexing

```
In [7]: ix = 2
        jx = 2
        x = (-h.L+h.grid_size)/2 + (jx-1)*h.grid_size
        y = (h.L-h.grid_size)/2 - (ix-1)*h.grid_size
        print(x,y)
        h.get_indices(x,y), h.get_field(x,y),h.get_gradient(x,y)

-0.4 0.4


Out[7]: ((2, 2), 7.0, array([  2.5,  12.5]))

In [8]: x1 ,y1 = -0.0179782121656 , -0.00947202123752

        print(x1,y1)
        h.get_indices(x1,y1), h.get_field(x1,y1),h.get_gradient(x1,y1)

-0.0179782121656 -0.00947202123752


Out[8]: ((3, 3), 13.0, array([  2.5,  12.5]))
```

3 Matrix and image correspondence

```
In [9]: print(h.h[1:h.N+1,1:h.N+1])
        h.show2d()
```

```
[[  1.   2.   3.   4.   5.]
 [  6.   7.   8.   9.  10.]
 [ 11.  12.  13.  14.  15.]
 [ 16.  17.  18.  19.  20.]
 [ 21.  22.  23.  24.  25.]]
```




```
In [10]: h.update(x,y,1)
         h.h[h.get_indices(x,y)]
```

```
Out[10]: 8.0
```

```
In [11]: h.update(x,y,-1)
         h.h[h.get_indices(x,y)]
```

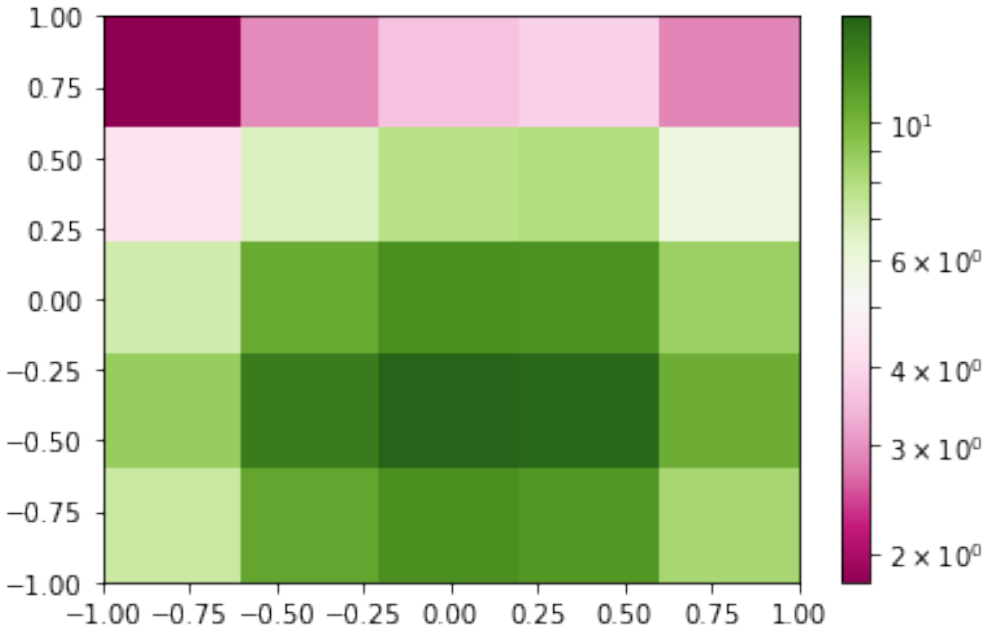
```
Out[11]: 7.0
```

4 EigenEvolution

```
In [12]: for i in range(1000):
             h.evolve()
```

```
In [13]: print(h.h[1:h.N+1,1:h.N+1])
         h.show2d()
```

```
[[  1.79428452   2.93015353   3.61524343   3.83897373   2.89441982]
 [  4.28959362   6.68365584   7.84007785   7.99588405   5.87813386]
 [  7.00906105  10.64352492  12.11773613  12.04524846   8.70596986]
 [  8.83369459  13.24479691  14.84869554  14.55702512  10.42223483]
 [  7.29496104  10.87285472  12.09978748  11.78167492   8.39509634]]
```




```
In [14]: print(h.h[1:h.N+1,1:h.N+1])
```

```
[[  1.79428452   2.93015353   3.61524343   3.83897373   2.89441982]
 [  4.28959362   6.68365584   7.84007785   7.99588405   5.87813386]
 [  7.00906105  10.64352492  12.11773613  12.04524846   8.70596986]
 [  8.83369459  13.24479691  14.84869554  14.55702512  10.42223483]
 [  7.29496104  10.87285472  12.09978748  11.78167492   8.39509634]]
```

The event schedule for this model is:

	A	B	C	D
1	#	1		
2	WorldState	computationalUse	buildLandscape	
3				
4	#	1	100000	
5	tasteA	evolve		
6				
7				
8	tasteA	search_producerV2		
9				
10	tasteA	update_Landscape		
11				
12	WorldState	computationalUse	updateBoundaries	
13	WorldState	computationalUse	updateLandscape	
14				
15	all	record		
16	tasteA	reset		

First, "WorldState" build a plain landscape. Then consumers moves, they interact with producers (through "search_producerV2" method), changing the variable "bought" according the the outcome of interaction. Then they update the landscape through the method "update_Landscape"of Consumer class, that we have add for the third model. The method change the field value relative to the position of the consumer by the amount $-s * bought$, where s is the strength of the consumer. We remember that the variable "bought" is one is the consumer has bought and -1 in the opposite case. If the consumer has not interact, is "bought" is zero. The code for the above mentioned method is

```
def update_Landscape(self):
    a = -self.strength*self.bought
    self.myWorldState.AL.update(self.xPos, self.yPos, a)
```

Then "WorldState" calls the method "mantain_pbc" and "evolve" of the class AdaptiveLandscape through the methods "updateBoundaries" and "updateLandscape".

Finally, consumers reset the "bought" variable to zero.

10.3 Results

The behavior of the system strongly depend on initial conditions. To better show the adaptive landscape influence we have performed simulations with initial producers' prices taken from a distribution which results from the sum of two Gaussian with different mean and equal variance; consumers' prices are picked from one of the two Gaussian used for producers, that with the lowest mean.

For all the simulations, producers are at the nodes of a square lattice; we have used periodic boundary conditions and the following parameters values (unless otherwise specified):

Results:

- The main result of our third model simulations is that producers coming from the highest mean Gaussian do not sell during the observation time (or during a significative fraction).
  - Figure 10.1 shows λ_c influence. Increasing λ_c make selling easier for high price producers; for a sufficiently large value of λ_c they are able to sell during the observation time.

Param.s	Value
N_c	50
N_p	36
Δt	0.01
$D_{consumer}$	0.1
λ_r	0.01
r_p	0.1

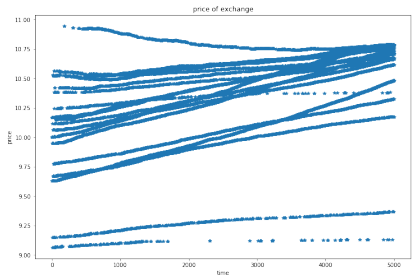
  - Figure 10.2 shows that the landscape influence is reduced increasing D_h as claimed in the model description.
  - Figure 10.3 shows k_h influence. To increase k_h reduce the landscape influence.

- To make clear the previous point we have compared a simulation of the third model with a corresponding one (same parameters values) of the first model. The comparison is shown in figure 10.4

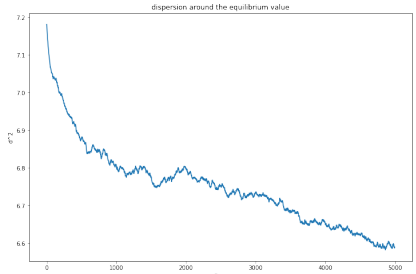
- Figure 10.5 shows graphs relative to a long observation time simulation. We can see that, for long times, higher price producers are able to sell, and that prices evolve towards the Hayekian value (eq.((8.17))).

  To explain this point we describe the "history" of the simulation. At the beginning, consumers move as free Brownian particle in a plain landscape. After the first interactions, consumers have build mountains in correspondence of higher price producers' position, and they have created valleys at lower price producers position. Mountains repel consumers, determining higher price producers isolation. Valleys attract consumers, that interact more frequently with lower price producers. This situation goes on until consumers price are in equilibrium with those of lower price producers: ones their prices are similar, consumers stop to update the field (in a step they will rise it up, and in the next they will decrease it). We have seen that field diffusion and decay lead the field gradient to extinguish. Then, after a sufficiently long period, the field will become a plain landscape, allowing consumers to interact again with higher price producers.

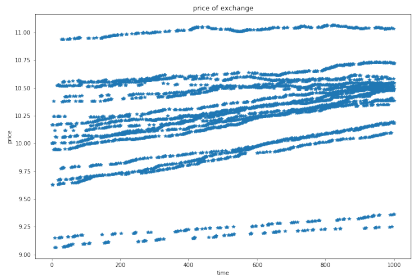
  However, this strongly depends on the simplicity of our models: we allow producers to survive after a long period in which they have not sold. In an agricultural market, a producer that does not sell is compelled to throw away her products; sooner or later she will close her firm. Further, we are not considering the birth of new firms. We believe that, adding death and birth processes to our model will prevent the landscape leveling.



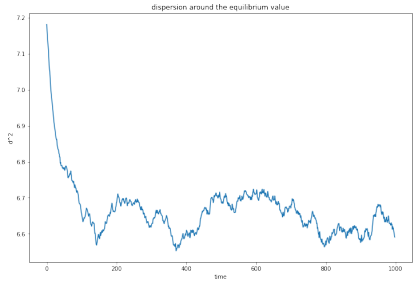
(a) $\lambda_c = 0.03$



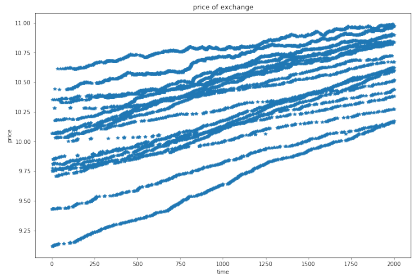
(b) $\lambda_c = 0.03$



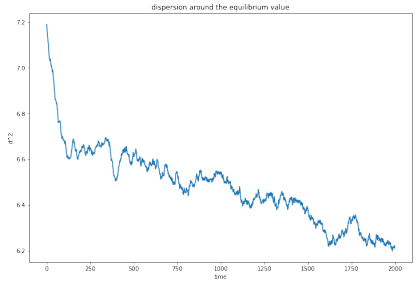
(c) $\lambda_c = 0.15$



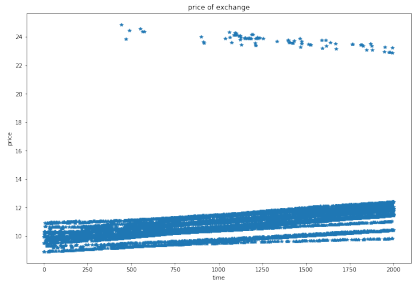
(d) $\lambda_c = 0.15$



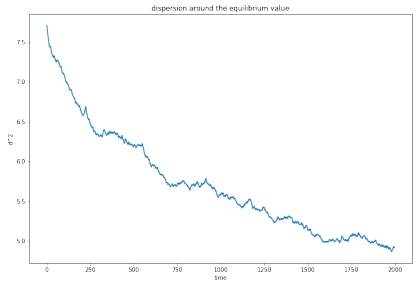
(e) $\lambda_c = 0.15$



(f) $\lambda_c = 0.15$

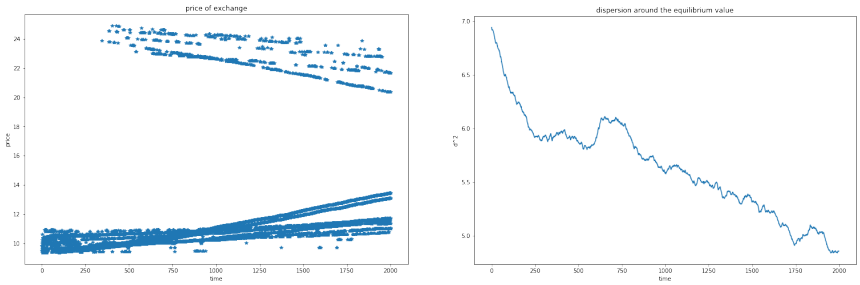


(g) $\lambda_c = 0.2$



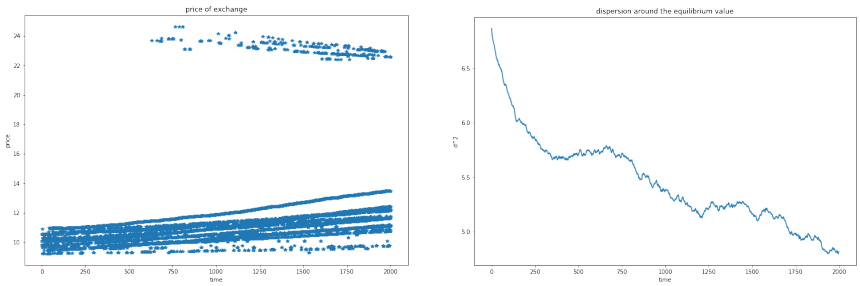
(h) $\lambda_c = 0.2$

Figure 10.1: Left figures are graphs of exchange prices, those on the rigth show the equilibrium function f^{eq} . For all the simulation we have $D_h = 10^{-3}$ and $k_h = 10^{-5}$.



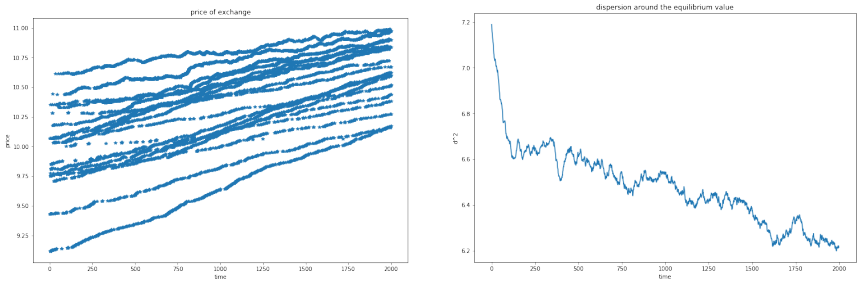
(a) $D_h = 0.02$

(b) $D_h = 0.02$



(c) $D_h = 0.01$

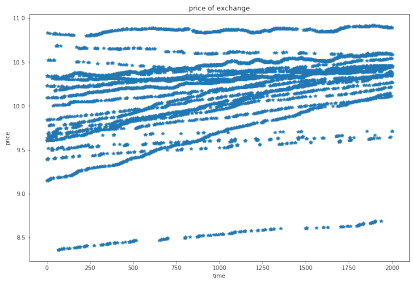
(d) $D_h = 0.01$



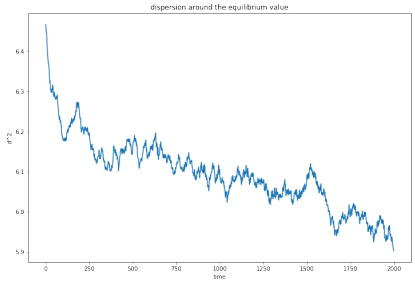
(e) $D_h = 0.001$

(f) $D_h = 0.001$

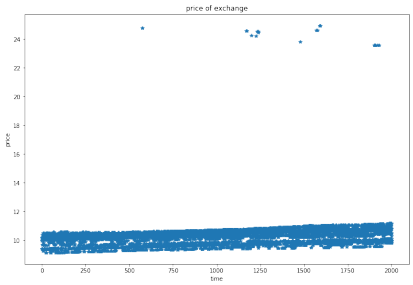
Figure 10.2: Left figures are graphs of exchange prices, those on the rigth show the equilibrium function f^{eq}. For all the simulation we have $\lambda_c = 0.15$ and $k_h = 10^{-3}$



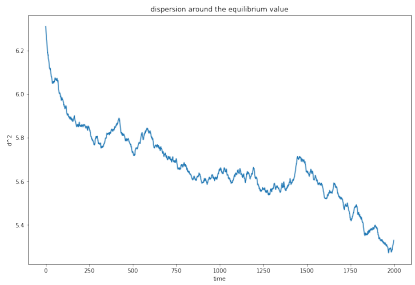
(a) $k_h = 0.1$



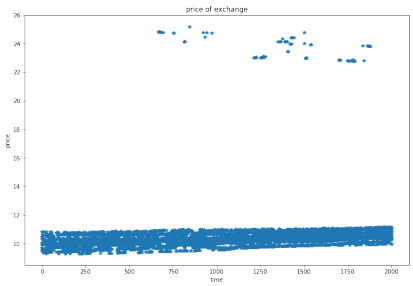
(b) $k_h = 0.1$



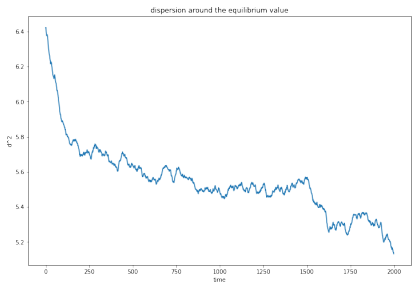
(c) $k_h = 1$



(d) $k_h = 1$

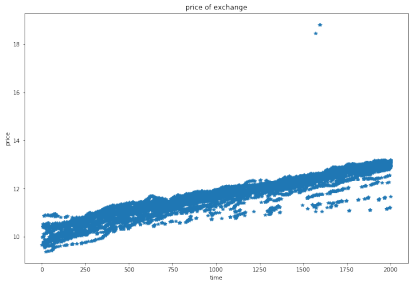


(e) $k_h = 10$

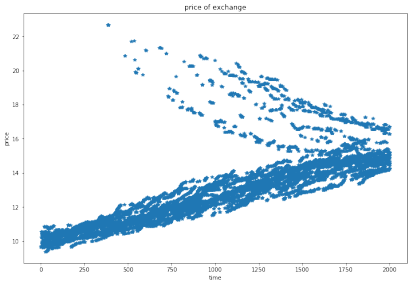


(f) $k_h = 10$

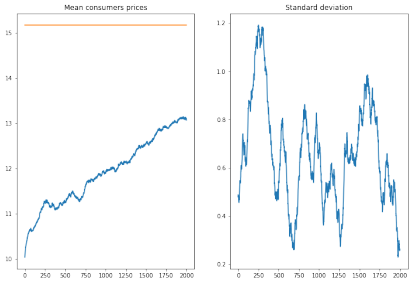
Figure 10.3: Left figures are graphs of exchange prices, those on the rigth show the equilibrium function f^{eq}. For all the simulation we have $\lambda_c = 0.15$ and $D_h = 10^{-3}$



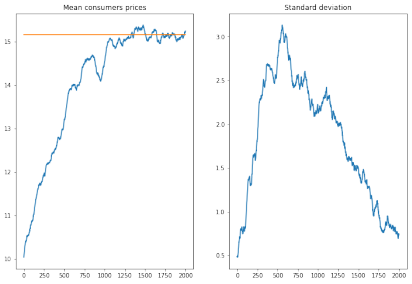
(a) Exchange prices



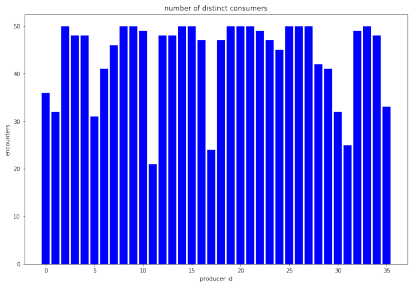
(b) Exchange prices



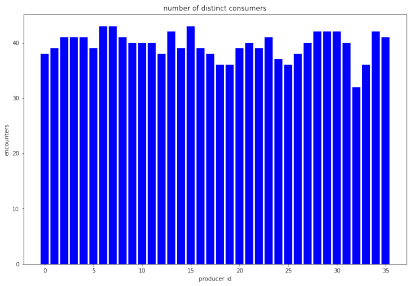
(c) Mean consumers price and standard deviation



(d) Mean consumers price and standard deviation

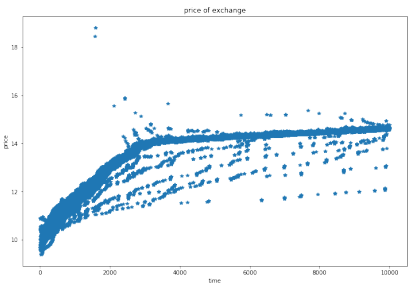


(e) Number of distinct consumers met by producers VS producers identity number

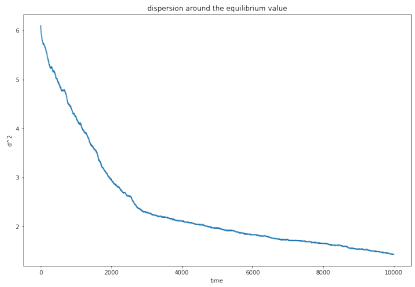


(f) Number of distinct consumers met by producers VS producers identity number

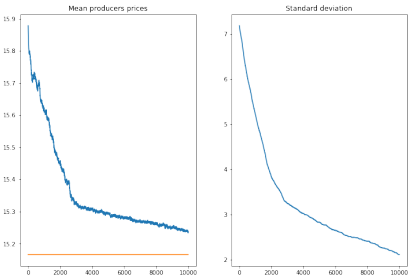
Figure 10.4: Left figures are relative to simulations of the third model, those on the right are relative to the first model. Here $\lambda_c = 0.15$, $\lambda_r = 0.1$, $D_h = 10^{-3}$, $k_h = 0.01$



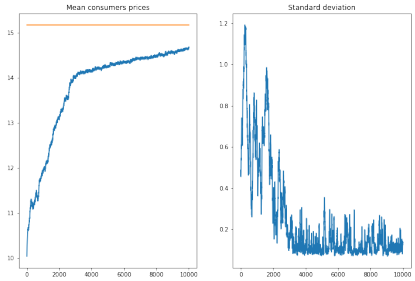
(a) Exchange prices



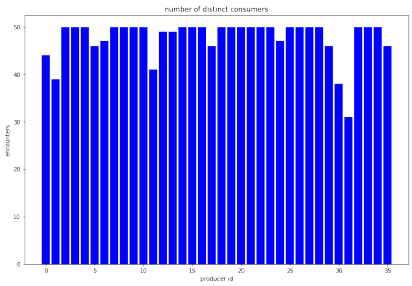
(b) f^{eq}



(c) Mean producers price and standard deviation



(d) Mean consumers price and standard deviation



(e) Number of distinct consumers met by producers VS producers identity number

Figure 10.5: $\lambda_c = 0.15$, $\lambda_r = 0.1$, $D_h = 10^{-3}$, $k_h = 0.01$

Chapter 11

Conclusions

In our work we have studied three different market models. In chapter 2 we have described the real world problem that have inspired the thesis.

In chapter 3 we have described complex systems with particular attention to those features that cause the failure of traditional modeling approach in economics. These give also the reason to prefer ABM to classical modeling approach.

In chapter 4 we have presented ABM. In particular, we have described the drawbacks encountered in building and using agent based models.

In chapter 5 we have described Brownian Agent framework, and in particular the adaptive landscape tool, which constitutes a very powerful way to explore feedback among agents.

In chapter 6 we have presented the software we have used to build our models: SLAPP.

In chapter 7 we have described Hayek's vision of market, which differ from traditional approach and better adapt to complex system paradigm.

In chapter 8 we presented the first model. Although extremely simple, this model in part confirm Hayek hypothesis on the formation of a unique price. But it also shows that, even in a system where an equilibrium condition is attainable in principle, the trading mechanism features may cause an evolution out of equilibrium.

In chapter 9 we have presented our second model, in which we have studied how imitation among agents can change the market dynamic.

Finally, in chapter 10 we have studied the feedback between search activity, represented by Brownian motion, and purchases dynamics through the adaptive landscape tool. The main result is that the feedback implies a different dynamic with respect to the first model: for a long initial period consumers buy mainly from those producers that offer the lower price. After this long initial period consumers stop, on average, to update the field that, due to its eigendynamic, become a plain landscape. However, this is due to the excessive simplicity of the models: birth and death processes can sustain the field existence preventing its leveling.

Future developments includes inserting the birth of new producers and the death of those that do not sell for long periods.

As a further improvement of our models, we can add a vector of characteristics to products to account their quality or the distance they have traveled from

the production place. Modifying the rule by which consumers judge producers offer, we can study market dynamic in presence of diverse choices. To exploit the adaptive landscape tool, we can subdivide consumers in two classes, one that judge only the offer price, the other that judge also environmental quality of products. Each of the class generates a component of the field, and it is influenced only by that component. Fixing transition rates between the two classes, we can study the competitions between the two judging rules looking at the influences of the field.

Chapter 12

Appendix A: Brownian motion

Brownian particles are particles characterized by random motion. They move in a viscous material, eventually in presence of an external force field. The casualty of the trajectories is due to the collisions with the smaller particles of the material.
Following the Langevin approach, we can write the Newtonian equation of motion of a Brownian particle adding a stochastic force which represents the contribution of random collisions

$$\frac{d\overrightarrow{r}}{dt} = \overrightarrow{v} \tag{12.1}$$
$$\frac{d\overrightarrow{v}}{dt} = -\gamma\overrightarrow{v} + \sqrt{2S}\xi_i \tag{12.2}$$
$$\tag{12.3}$$

where $\gamma_0 = m\gamma$ is the drift coefficient, m is the mass, $S = \gamma\frac{k_B T}{m}$ and $\hat{\xi}_i$ is a Gaussian white noise process with

$$\langle\xi(t)\rangle = 0 \tag{12.4}$$
$$\langle\xi(t)\xi(t')\rangle = \delta(t-t') \tag{12.5}$$

In our models we have dealt with the over-damped limit, that occur for $\gamma_0 \gg 1$. In this limit, accelerations are suppressed in a much smaller time than those required to appreciate sensible variation of the position. Then, we can set $\frac{d\overrightarrow{v_i}}{dt} \approx 0$ (adiabatic approximation) to find

$$\overrightarrow{v_i} = \sqrt{2D}\xi_i \tag{12.6}$$

Substituting in the position equation we find

$$\frac{d\overrightarrow{r_i}}{dt} = \sqrt{2D}\xi_i \tag{12.7}$$

The Fokker-Planck equation considering, only one spatial dimension is (see [15])

$$\frac{\partial p}{\partial t} = D\frac{\partial^2 p}{\partial x^2} \tag{12.8}$$

with initial and boarders conditions

$$p(x,0) = \delta(x - x_0) \tag{12.9}$$

$$p(-\frac{L}{2}, t) = p(\frac{L}{2}, t) \forall x, t \tag{12.10}$$

for periodic boundary conditions, or

$$\frac{\partial p(x,t)}{\partial x} = 0 \quad \text{if} \quad |x| > \frac{L}{2} \tag{12.11}$$

for reflecting walls at $\pm\frac{L}{2}$.

To solve for periodic boundary conditions we can take the Fourier series

$$p(x,t) = \sum_{n=-\infty}^{\infty} \hat{a}_n(t) e^{i\omega n x} \tag{12.12}$$

where $\omega = \frac{2\pi}{L}$ and

$$\hat{a}_n(t) = \frac{1}{L}\int_{-\frac{L}{2}}^{\frac{L}{2}} e^{-i\omega n x} p(x,t) dx \tag{12.13}$$

Inserting into equation ((12.8)), we find

$$\frac{\partial \hat{a}_n}{\partial t} = -D\omega^2 n^2 t \hat{a}_n \tag{12.14}$$

Solving with the initial condition

$$\hat{a}_n(0) = \frac{1}{L} e^{-i\omega n x_0} \tag{12.15}$$

we obtain

$$\hat{a}_n(t) = \frac{1}{L} e^{i\omega n x_0} e^{-D\omega^2 n^2 t} \tag{12.16}$$

Substituting in ((12.12)) we find

$$p(x,t) = \frac{1}{L}\sum_{n=-\infty}^{\infty} e^{i\omega n(x-x0)} e^{-D\omega^2 n^2 t} \tag{12.17}$$

We can see that for $t \gg \frac{1}{D\omega^2}$ all the terms with $n \neq 0$ vanish, and the probability density function is time independent and uniform:

$$p(x,t) = \frac{1}{L} \tag{12.18}$$

The generalization to the two dimensional case is straightforward and we have

$$p(x,y,t) = \frac{1}{L^2} \tag{12.19}$$

For the case of reflecting walls boundary condition, we have to write the Fourier series in terms of sine and cosine functions, and we have to retain only the cosine terms. Then a similar procedure shows that also in this case, the probability density functions is time independent and uniform for times $t \gg \frac{L^2}{d(2\pi)^2}$.

Bibliography

[1] R Boero, M Morini, M Sonnessa, and P Terna. *Agent-based Models of the Economy: From Theories to Applications*. Springer, 2015.

[2] Paul L Borrill and Leigh Tesfatsion. Agent–based modeling: the right mathematics for the social sciences? *The Elgar companion to recent economic methodology*, 228, 2011.

[3] Samuel Bowles, Alan Kirman, and Rajiv Sethi. Retrospectives: Friedrich hayek and the market algorithm. *Journal of Economic Perspectives*, 31(3):215–30, 2017.

[4] Douglas Bridges and Erik Palmgren. Constructive mathematics. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 2016 edition, 2016.

[5] In-Koo Cho and Thomas J Sargent. Neural networks for encoding and adapting in dynamic economies. *Handbook of computational economics*, 1:441–470, 1996.

[6] Rosaria Conte, Nigel Gilbert, Giulia Bonelli, Claudio Cioffi-Revilla, Guillaume Deffuant, Janos Kertesz, Vittorio Loreto, Suzy Moat, J-P Nadal, Anxo Sanchez, et al. Manifesto of computational social science. *The European Physical Journal Special Topics*, 214(1):325–346, 2012.

[7] Joshua M Epstein. *Generative social science: Studies in agent-based computational modeling*. Princeton University Press, 2006.

[8] Magda Fontana. Can neoclassical economics handle complexity? the fallacy of the oil spot dynamic. *Journal of Economic Behavior & Organization*, 76(3):584–596, 2010.

[9] José Manuel Galán, Luis R Izquierdo, Segismundo S Izquierdo, José Ignacio Santos, Ricardo Del Olmo, Adolfo López-Paredes, and Bruce Edmonds. Errors and artefacts in agent-based modelling. *Journal of Artificial Societies and Social Simulation*, 12(1):1, 2009.

[10] N Gilbert. Agent-based models (quantitative applications in the social sciences) sage publications. 2008.

[11] Alan Kirman. Complexity and economic policy: A paradigm shift or a change in perspective? a review essay on david colander and roland kupers's complexity and the art of public policy. *Journal of Economic Literature*, 54(2):534–72, 2016.

[12] Rolf Landauer. Computation: A fundamental physical view. *Physica Scripta*, 35(1):88, 1987.

[13] Nelson Minar, Roger Burkhart, Chris Langton, Manor Askenazi, et al. The swarm simulation system: A toolkit for building multi-agent simulations. 1996.

[14] Diana Richards, Brendan D McKay, and Whitman A Richards. Collective choice and mutual knowledge structures. In *Modeling Complexity In Economic And Social Systems*, pages 273–289. World Scientific, 2002.

[15] Hannes Risken. The fokker-planck equation. methods of solution and applications, vol. 18 of. *Springer series in synergetics*, 1989.

[16] Frank Schweitzer. *Brownian agents and active particles: collective dynamics in the natural and social sciences*. Springer Science & Business Media, 2007.