

0000362785

ALMA MATER STUDIORUM - UNIVERSITÀ DI
BOLOGNA

FACOLTÀ DI ECONOMIA

LAUREA MAGISTRALIS IN ECONOMICS

Rational or smart?

An auction study with simulation.

PIETRO BATTISTON

GIACOMO CALZOLARI

Topics in Economic Theory

SESSIONE PRIMA

ANNO ACCADEMICO 2009/2010

Chapter 1

Quantal response functions

The concept of *quantal response function* is a central tool of the *quantal response equilibrium*, which has been first proposed by McKelvey and Palfrey in [3] to model games in which out of (Nash) equilibrium decisions are assumed to play an important role, too relevant to be kept out of the picture.

1.1 Definition

We know that, given a discrete game of N players, a Nash equilibrium in mixed strategies is a set of individual mixed strategies

$$s^* = (s_1^*, \dots, s_n^*)$$

such that $\forall i$, and for any (mixed) strategy \tilde{s}_i available to player i , we have that

$$u_i(s_i^*, s_{-i}^*) \geq u_i(\tilde{s}_i, s_{-i}^*)$$

where by $u_i(s_i, s_{-i})$ we denote the expected payoff of an agent playing strategy s_i against other agents playing strategies s_{-i} .

Under those assumptions, and given a zero-mean error distribution f_i over \mathbb{R}^J for each player,¹ the definition of a quantal response equilibrium implies the construction of a “perturbed payoff function” \hat{u} defined as follows: if

$$u_i(s_i, s_{-i}) = \sum_{j \in S_i} s_{i_j} \cdot u_i(j, s_{-i}),$$

then²

$$\hat{u}_i(s_i, s_{-i}) = \sum_{j \in S_i} s_{i_j} \cdot (u_i(j, s_{-i}) + \epsilon_{i_j}) = u_i(s_i, s_{-i}) + s_i \cdot \epsilon'_i$$

¹For simplicity - and with no loss of generality - assuming that $|S_i| = J$ for each $i \in \{1, \dots, N\}$.

²Denoting the transposed of a vector A as A' .

and in particular, for pure strategies,

$$\hat{u}_i(j, s_{-i}) = u_i(j, s_{-i}) + \epsilon_{ij},$$

with $\epsilon_i = (\epsilon_{i1}, \dots, \epsilon_{iJ})$ distributed according to f_i .

For each player i and for each possible strategy j , there is a region $\mathcal{R}_{ij}(s_{-i}) \subset \mathbb{R}^J$ such that if $\epsilon_i \in \mathcal{R}_{ij}$, then j “looks like” the best pure strategy to i , that is, formally, $\hat{u}_i(j, s_{-i}) \geq \hat{u}_i(\tilde{j}, s_{-i})$ for any \tilde{j} . The probability measure f_i hence induces a probability measure on $\{\mathcal{R}_{i1}(s_{-i}), \dots, \mathcal{R}_{iJ}(s_{-i})\}$:

$$\sigma_{ij}(s_{-i}) \stackrel{\text{def}}{=} \mathbb{P}(\mathcal{R}_{ij}(s_{-i})) = \int_{\mathcal{R}_{ij}(s_{-i})} f_i(\epsilon_i) d\epsilon_i.$$

We can finally define a quantal response equilibrium as a vector σ^* such that

$$\sigma_{ij}^* = \sigma_{ij}(\sigma_{-i}^*) \quad \forall i \in \{1, \dots, n\} \quad \forall j \in \{1, \dots, J\}.$$

Intuitively, the quantal response equilibrium is based on the same rationale of the Nash equilibrium, except that each agent makes some evaluation error, and by knowing which distribution those error terms follow, they are able to internalize their (expected) effect on participants’ behaviour.

1.2 What’s behind

Quantal response equilibria converge to Nash equilibria as the variance of the error term tends to 0.³ They are a useful tool both for selecting *among* Nash equilibria, and for studying the “robustness”, or “plausibility” of selected equilibria, by looking at how far they move away from Nash equilibria as the *variance*⁴ grows: an equilibrium is made of mixed strategies combining the given possible pure strategies.

In this work, instead, quantal response functions will be purposely crafted *around* predicted Nash equilibria, and this will allow us to see the “choice” of one or the other as a *deviation* from that prediction. I will not discuss in detail what is the economic interpretation of this *error*: though the most obvious meaning it can be attributed is the simple “mistake in pressing the button”, that is certainly not the only motivation behind the development of the new equilibrium concept, which

³Though it may seem intuitive to think that QRE converge to *trembling-hand perfect* equilibria, this is not the case, as pointed out for instance by [3].

⁴Usually denoted with λ : when $\lambda \rightarrow 0$, we get the regular Nash equilibrium

well on the contrary has gained importance also for its apparent ability to more realistically predict the outcome of some games - for instance, repeated coordination games - suggesting hence that it may better reflect a type of rationality that we find “more natural”. I will hence not focus on the search for quantal response equilibria; I will however borrow the basic instrument of this equilibrium concept, namely the quantal response function, to give a *shape* to *deviation from theoretical predictions* (in particular, Nash equilibria) on behalf of players.

Quantal response functions (σ_{ij} , in the above formalization) in general depend on the error distribution they originate from (for instance, the widely used *logistic quantal response function* is the result of choosing an error distribution with cumulative density function

$$F(x) = e^{-e^{-\lambda x - \gamma}},$$

where γ is such that $F(0) = .5$). They are however a higher level tool which already assigns a probability to every possible strategic choice of the opponents - depending on the probability distribution of other players' actions. In this work, instead, the symmetry between agents will be broken and while some players' response functions will only depend on *their own* errors, some others (often only one) will play deterministically, by maximizing the expected utility, which internalizes the (known) probabilistic behaviour of opponents.

This is precisely the meaning that I will give to the term “distrust”: *the expectancy, on the behalf of some given player(s), that some opponent(s) will not act accordingly to a Nash equilibrium*. Similarly, I will often design decisions as “irrational” insofar as they deviate from those same theoretical predictions. It is totally evident that the usage of this term, when intended with this meanings, completely abstains from any pejorative judgment - well on the contrary, I will present some deviations as “necessary” in order to get the best expected result (given the particular set of informations/assumptions of the player behind them).

One more technical detail has to be faced: the concepts of quantal response equilibrium and quantal response function are traditionally conceived as valid approaches in cases where the space of strategies is *discrete*. Though generalizations to the continuous case exist, to study auctions, which luckily are characterized by the fact that variables of interest range on *intervals*, I will target the problem simply by introducing *ticks*, which will provide, in any situation, a finite number of choices among which a player can select one. It is important to stress that my aim is not to study the *relevance* of tick size, as already done

in many existing papers (i.e. in empirical or theoretical measures on double auction markets): well on the contrary, the tick size will be kept to a value as little as possible to make its effect negligible.

The concept itself of quantal response functions may seem at first sight a very poor modelization of *irrationality* - in the end, this general idea can describe such a complex space of strategies that the only possible way to meaningfully model it is to start from observational data and in this way construct empirically justified deviations. While I certainly don't mean to assert that quantal response functions put the final word to the question, I will simply adopt the same reasoning that has motivated the concept of quantal response equilibrium: deviations are not simply a consequence of some exogenous hidden causes, but also of the belief that each agent has about competitors.

Translating this idea into my study, which will target mainly the effect of *distrust*, not just of "errors", I will assume that quantal response functions are not a model of *irrationality*, but instead a plausible model of how a particular player may think that his competitors will deviate from the classical prediction. The reason why I won't adopt the concept of quantal response equilibrium as a whole is that if a player has *distrust* in some opponents' rationality, it's hard to claim that she will assume they "feel the same", in that they also internalize the effect of deviations. More simply, if I expected other players to think rationally *and then possibly make errors*, I would internalize them and assume they do the same, but if instead (as is the case here) I expect other players to simply make irrational choices, I will *not* expect them to internalize them (or any possible response to them, on my or some other agent's behalf).

1.3 An example

A simple numeric example may clarify the differences between the approaches. Let us assume that two agents, a and b , enter a first price sealed bid auction. Each of them attributes a private value, picked from a uniform distribution on $[0, 1]$, to the object.

- The only symmetric Nash equilibrium is known to be of the form $\beta_a(v) = \beta_b(v) = \beta(v) \stackrel{\text{def}}{=} \frac{v}{2}$. The expected utility for each agent is also symmetric and is equal to

$$\mathbb{P}\{\beta(v_i) > \beta(v_{-i})\} \cdot (v_i - \beta(v_i)) = \mathbb{P}\{v_i > v_{-i}\} \cdot \frac{v_i}{2} = \frac{v_i}{4}.$$

- Now let us assume that agents make evaluation errors of the following form: at some moment they must choose between playing $\beta_1(v) = \frac{v}{2}$ or $\beta_2(v) \neq \beta_1(v)$. They first build mixed strategies by estimating the expected revenue of each option, and it is at this level that the errors happen: denoted with p_b (respectively p_a) the probabilities that b (respectively a) chooses the first option (obviously $p_a \leq 1$ and $p_b \leq 1$), and with $u_{i,j}(v, p_{-i})$ (for $i \in \{a, b\}$, $j \in \{1, 2\}$) the expected utility of each option, a will not simply maximize u_j , but $\hat{u}_j = u_j + \epsilon_j$ instead, and the decision making process of b will be perfectly specular.

This time we can hence write:⁵

$$\mathbb{P}\{\beta_i(v_i) > \beta_{-i}(v_{-i})\} = p_{-i} \mathbb{P}\left\{\beta_i(v_i) > \frac{v_{-i}}{2}\right\} + (1 - p_{-i}) \mathbb{P}\left\{\beta_i(v_i) > \frac{v_{-i}^2}{2}\right\}$$

which gives the following expected payoffs:

$$\begin{aligned} u_{a,1}(v_a, p) &= \mathbb{P}\{\beta_1(v_a) > \beta_b(v_b)\} \cdot (1 - \beta_1(v_a)) + \epsilon_1 \\ &= [pv_a + (1 - p)\mathbb{P}\{\beta_1(v_a) > \beta_2(v_b)\}] \cdot (1 - \beta_1(v_a)) + \epsilon_1 \\ u_{a,2}(v_a, p) &= \mathbb{P}\{\beta_2(v_a) > \beta_b(v_b)\} \cdot (1 - \beta_2(v_a)) + \epsilon_2 \\ &= [p\mathbb{P}\{\beta_2(v_a) > \beta_1(v_b)\} + (1 - p)v_a] \cdot (1 - \beta_2(v_a)) + \epsilon_2 \end{aligned}$$

and symmetrically (replacing p, a, b with q, b, a respectively) for the other agent. Given that, the distribution of ϵ_i generates a distribution of probability on $\{1, 2\}$:

$$\begin{aligned} \mathbb{P}_p\{1\} &= \int_{\epsilon_1, \epsilon_2: u_{a,1} > u_{a,2}} dF \\ \mathbb{P}_p\{2\} &= \int_{\epsilon_1, \epsilon_2: u_{a,2} > u_{a,1}} dF. \end{aligned}$$

where F is the distribution of ϵ_1, ϵ_2

A quantal response equilibrium will be a pair (p, q) such that $q = \mathbb{P}_p\{1\}$ and $p = \mathbb{P}_q\{1\}$.

⁵It must be noticed that if b plays the “overbidding” strategy, a will have no chance to win the auction by playing the Nash equilibrium, and vice versa.

- In this work, to model distrust I will assume that a estimates u_1 and u_2 as above, but instead b calculates

$$\begin{aligned} u_{b,1}(v_b) &= \mathbb{P} \{ \beta_1(v_b) > \beta_1(v_a) \} \cdot (1 - \beta_1(v_b)) + \epsilon_1 \\ &= v_b \cdot (1 - \beta_1(v_b)) + \epsilon_1 \\ u_{b,2}(v_b) &= \mathbb{P} \{ \beta_2(v_b) > \beta_1(v_a) \} \cdot (1 - \beta_2(v_b)) + \epsilon_2; \end{aligned}$$

again, the distribution of ϵ_i induces a distribution of probability on $\{1, 2\}$. This probability will be considered by a in the search for the optimal (pure) strategy.

Chapter 2

Building distrust

The crucial point in the simulation is the construction of the strategy profile of the agent(s) assuming evaluation errors on behalf of the other participants.

The study is made through the already mentioned discretization: the range of the value, which I will always assume (with no loss of generality) being the interval $[0, 1]$, is subdivided into a given number of ticks. Based on the distribution chosen for the value, each tick gets a finite probability mass.

This construction must be done deterministically, and not, for instance, via a simulated learning mechanism on behalf of the “smart” player, since that would imply a very high computational cost - linear in the number of ticks, the number of players and the number of iterations of the auction - to keep the variance of results low.

First, the functional form for the *expected payoff as function of bid and private value*, as estimated by an agent assuming opponents play the Nash equilibrium, is constructed: let us call it $u(b, x)$. For instance, in the case of a first price auction with uniformly distributed private values, we have (see for instance [1]):

$$u(b, x) = \left(b \frac{N}{N-1} \right)^N [x - b].$$

Then, the distribution of probability of bids is calculated as - intuitively (since this is evidently not a real probability, which would otherwise evaluate to 0 everywhere) - the probability for any given value \tilde{b} that:

$$\hat{u}(\tilde{b}, x) = \max_b \hat{u}(b, x)$$

where $\hat{u}(b, x)$ is simply $u(b, x) + \epsilon_b$ and ϵ_b is our random evaluation error.

More formally, fixed $x = \tilde{x}$, in the case in which ϵ_b is distributed uniformly on $[\epsilon_-, \epsilon_+]$, after defining the set

$$U_b = \{b' \in [0, 1] \mid u(b', \tilde{x}) \geq b\}, \quad (2.1)$$

and denoting the indicator function of a set S as I_S , the distribution of probability can be obtained as

$$p(b) \propto \int_{b+\epsilon_-}^{b+\epsilon_+} \frac{1}{\int_0^1 I_{u^{-1}(U_b)}} du$$

where since U_b is an interval (u is single peaked), containing \bar{b} for which $u(\bar{b})$ is maximum, then $u^{-1}(U_b)$ is an interval too,¹ and hence we can write:

$$p(b) \propto \int_{b+\epsilon_-}^{b+\epsilon_+} \frac{1}{|u^{-1}(U_b)|} du$$

Remark 2.1. *It may be interesting to notice that, with a first order approximation, this process of assuming that agents make evaluation errors (under/overestimating the expected output of a given strategy) is analogous to just assuming that they choose the right strategy but then make a random (and independent from the private value) error around that choice. In other terms, since in \tilde{b} we have $u'(\tilde{b}) = 0$, then for $[\epsilon_-, \epsilon_+] \rightarrow [0, 0]$ we could equivalently assume that the bid of the agent simply follows the rule $\tilde{b} + \epsilon$, with ϵ uniformly distributed. However, this is instead not possible for non-infinitesimal errors, since the distribution of ϵ would not be uniform and more importantly would depend upon x , the private value.*

2.0.1 Discretization

In the discrete environment, this translates into the following: there is a finite number (K) of ticks, regularly distributed at distances of $\tau = \frac{1}{K}$, from $0\tau = 0$ to $K\tau = 1$. Let k_0, \dots, k_K be (all distinct) indexes in $\{0, \dots, K\}$ such that

$$u(k_0\tau) \geq u(k_1\tau) \geq \dots \geq u(k_K\tau).$$

¹I'm making the assumption that the domain of definition of u is only $[0, 1]$. The reason why I do that is formally not obvious, but derives from the very simple hypothesis that no participant in the auction will act in such a way that will make him loose wealth *with certainty*.

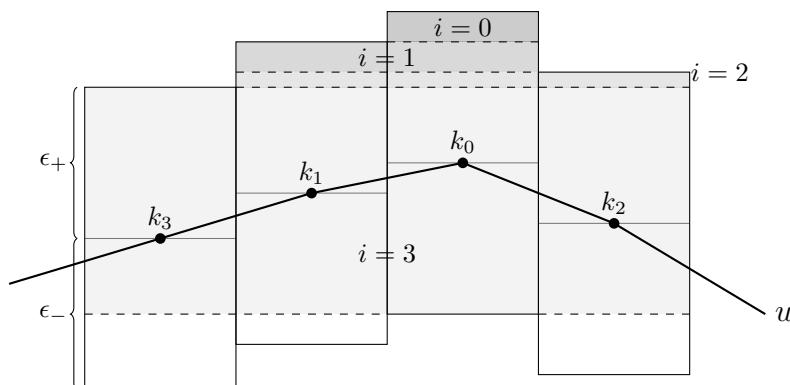


Figure 2.1: The mechanism behind the distribution of probability. Darker zones increase in a higher measure the probability mass of ticks they “intersect”.

Then for each $k \in \{0, \dots, K\}$, I define²

$$p'(k_j) = \sum_{i=j}^K |[u(k_{i+1}), u(k_i)] \cap [u(k_K) + \epsilon_-, u(k_K) + \epsilon_+]| \cdot (i + 1). \quad (2.2)$$

The above intersection is an interval for all i such that

$$u(k_i) + \epsilon_+ > u(k_K) + \epsilon_-, \quad (*)$$

and is empty otherwise. If we denote by $k_{\underline{i}}$ the *biggest* index for which (*) holds, then (2.2) can be rewritten as

$$p'(k_j) = \begin{cases} \sum_{i=j}^{\underline{i}} [(u(k_i) - u(k_{i+1})) (i + 1)] + (u(k_{\underline{i}}) - u(k_K) + \epsilon_-) & \text{for } j \leq \underline{i} \\ 0 & \text{for } j > \underline{i} \end{cases}$$

It is evident, as can be seen from figure 2.1, that $p'(k_j)$ has a maximum in $j = 0$.

p' is still not a probability: in particular, its total mass depends on the particular shape of u . Next step is hence to normalize it, deriving:

$$p(k_j) = \frac{p'(k_j)}{\sum_{i=0}^K p'(k_i)}.$$

This is finally the distribution of probability of the bids *for a given private value*. But our “smart” agent does not know the private value of

²In the sum below, the term $u(k_{K+1})$ can appear, which is not defined; we can give it the value $u(k_K) + \epsilon_-$, or lower, as this will not affect the analysis in any way.

the opponents, and what he really cares about is the aggregate probability, that is, all the probabilities seen so far weighted on the distribution of private values. In the continuous case, this would mean

$$P(b) = \int_0^1 p(x) dx;$$

in the discretized framework, this corresponds to

$$P(k_j) = \sum_{\tilde{h}=0}^K \tau \cdot p_{h\tau}(k_j)$$

where $p_{h\tau}$ is calculated fixing $\tilde{x} = h\tau$ back in (2.1).

2.1 Summing up

Once the probability that the “smart” player attributes each one of his opponents is determined, as seen so far, the construction of the strategy proceeds by:

1. calculating a discretized cumulative distribution function, simply by summing the mass of ticks t_k with k lower than a given k' ,
2. calculating the cumulative distribution of the highest order statistics, as the simple product of the vectors of cumulative distribution function (which are one for each player),
3. obtaining the expected gain by multiplying the payoff obtained in case of winning by bidding a certain value t_k by the cumulative distribution function calculated above for t_k ,
4. finding the $t_{\hat{k}}$ which maximizes the expected gains.

Chapter 3

Private value

Several different types of auctions are used around the world, and ironically the most popularly associated with the term "auction", the open English one, is probably the one on which studies on strategic behaviour have less to say: while several behavioural works have highlighted deviations from theoretical predictions - even striking ones, as in [2] - it is virtually impossible to make errors *once the private value given to the auction is known with certainty*: the optimal strategy - increasing the bid until it reaches the private value assigned, then stopping - is very simple and natural.

The other three mechanisms traditionally considered by auction theory are:

- sealed-bid *first price auction*, which differs from the English one in the fact that participants don't get any information during the auction, and don't have the possibility of filing a *sequence* of bids: they just decide for one, and file it,
- sealed-bid *second price auction*, which is similar to the first price auction, but designs as the winner not the one filing the highest offer, but instead the one bidding the *second highest* value,
- *Dutch auction*, which is in some way the reverse of the English one: the price (often tracked by a sort of clock) starts very high and keeps lowering, until one of the participants *accepts*, winning the auction and paying the current price.

The Dutch auction is *strategically equivalent* to the first price one: that means, for instance, that an auction house running a Dutch auction could in principle accept anticipated sealed offers, with an employee of the house acting as proxy, offering the sum written in the sealed envelope on behalf of the remote participant, and in this situation *partici-*

*pating directly or through the proxy would make virtually no difference.*¹

So to get a picture of effects of distrust in private value auctions, we are down to *two mechanisms*, which will be now separately studied.

3.1 Second price auctions

Second value auctions in the classical (“undisturbed”) case are characterized by particularly simple strategies: *truthtelling* (weakly) dominates (it is at least as convenient, whatever other participants do - and strictly more convenient in some cases) all the others:

$$\beta^*(v_i) = v_i.$$

The dominance automatically implies that this optimal strategy is unaffected by any possible expected deviation of the other players - that is, *distrust* has no effect in this case.

It may be worth mentioning that the effective presence of noise in other players’ strategies can have important fallbacks on some characteristics of the auction - for instance, it can easily be seen that even a 0-mean white noise applied ex-post to bids can on the one hand increase expected revenues for the auctioneer, and on the other decrease both *chances of winning* and *expected gain from a win* for players playing non disturbed strategies. This reflection is however out of the scope of this work, and moreover general effects of irrationality on auctions have been already studied quite exhaustively in a number of papers: for instance [2] interestingly considers the effects of overbidding in eBay auctions, and the influence of an even small number of overbidders.

3.2 First price auctions

In the case of first price, private value, auctions instead benchmarking the effect of distrust against classical predictions is not trivial, and will require the approach described in chapter 2.

Figure 3.1 shows the result of the construction of the “distrusting” profile with an error term of variance zero - in other terms, of the “trust-

¹In the real world, the important difference between the two mechanisms, which makes the Dutch auction particularly favourable in situations where speed matters, is the time factor, that I will simply not consider here.

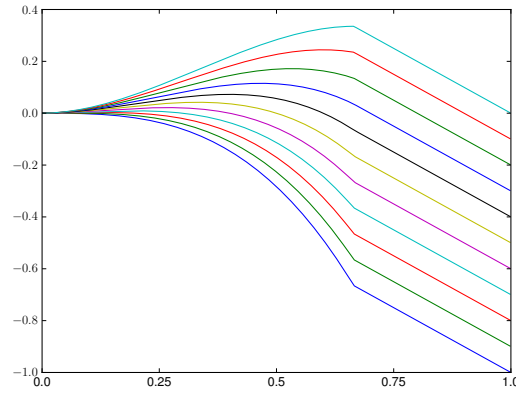


Figure 3.1: Expected payoff of possible responses (bids) to different private values, in an auction with 3 players and “clean” Nash equilibrium strategies (no evaluation errors).

ing” profile, the Nash equilibrium prediction itself.

In the plot, each line corresponds to a given private value: for instance, the lowest one is the plot of expected payoffs (y axis) in function of the bid (x axis) when the private value is 0, while the top one is the same plot when the private value is 10, and all others correspond to private values of 1, 2, 3. . .

As expected, the maxima of the different profiles exactly correspond to the Nash predictions: from the picture, in particular, it can be easily noticed that with a private value of 0 the best choice consists indeed in playing 0, while with a private value of 1 the maximum is $\frac{2}{3} = \frac{N}{N+1}v_i$.

Figure 3.2 shows the next step: it represents the strategy profile for an individual that attributes to opponents an *ex post* error, that is, assumes that they do find the best possible response, but then their effective bid is perturbed. The error term follows an uniform distribution of width 0.353 centered in 0. The result is similar, but the noise has the effect to smooth the expectation profiles.

The interesting point is the difference between the kind of error that is assumed in quantal response functions and a bare noise around the first best bid, as in figure 3.2: since the (clean) expected payoffs function is not symmetric around its peak (intuitively, it grows “slowly” and decreases “faster”, as can be seen in figure 3.1), the effect of an evaluation error will not be symmetric neither, but instead will shift the distribution of probability toward alternatives which have a higher expected value, and those are more present on the left than on the right

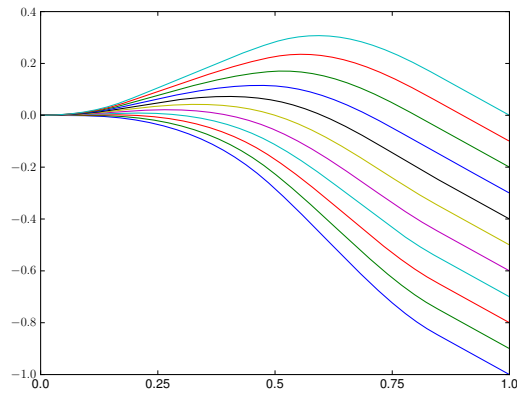


Figure 3.2: The same as above, but now internalizing the expected error of the others 2 players, which is still not structured as an *evaluation* error, but simply an error term distributed uniformly on an interval of width 0.353, applied *after* the choice of the bid.

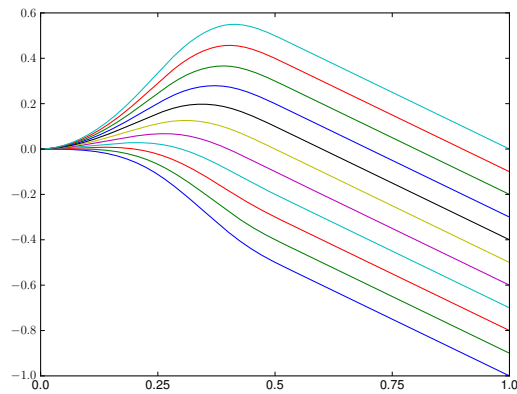


Figure 3.3: Here, the (expected) error finally intervenes at the step of evaluating the different options, in the form of an uniformly distributed error term: the difference from the above plot is evident in the fact that the expected bids, and hence the shape of the expected payoffs, are shifted to the left.

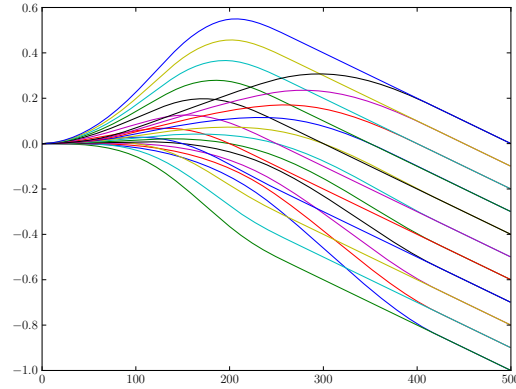


Figure 3.4: The two above plots compared. Since the effect of the shift to the left is due to lower expected bids on the behalf of opponents, the expected net relative benefit is positive, as is suggested by the higher peaks which can be seen on the left.

of the peak.

It must be observed that the difference between the two error concepts does *not* consist simply in a higher or lower *expected influence* on the final bid choice of “naive” agents: the value 0.353 in figure 3.2 is chosen precisely in order to compare two error distributions of bids which have the *same expected deviation from optimal bid*. Indeed, with ϵ uniformly distributed on $[-0.1, 0.1]$, \hat{u} defined as $\hat{u}(v) = u(v) + \epsilon$, and $\beta'(v) = \max_v \hat{u}(v)$ we get that

$$\mathbb{E}[|\beta'(v) - \beta(v)|] = 0.088$$

which is precisely the standard deviation of a uniform distribution of width 0.353.

3.2.1 The strategy profile

I now proceed to the simulation of an auction in which one participant feels distrust toward the others, that is, he plays with the strategy just described.

The very first simulation ran simply shows that distrust does not pay: if others players do play according to classical predictions, the player that assumes noise in their actions gets an overall damage from this assumption (figure 3.5).

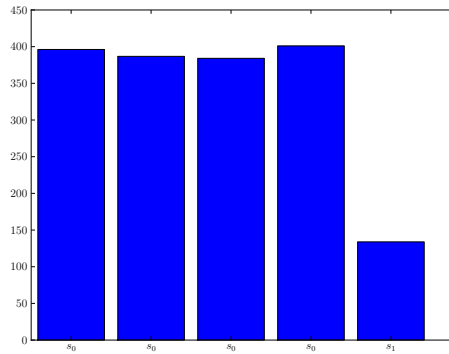


Figure 3.5: Results of a simulation with 5 agents, of which 4 play according to Nash equilibrium and one - the last - plays attributing to the others an uniform error of width 0.353 around the Nash equilibrium. Plotted are the cumulative payoffs after running the auction ten thousands times.

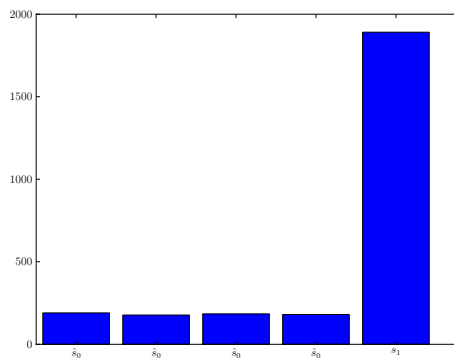


Figure 3.6: Results of the same simulation, where now however the first 4 players *indeed make* the error which the fifth attributes them.

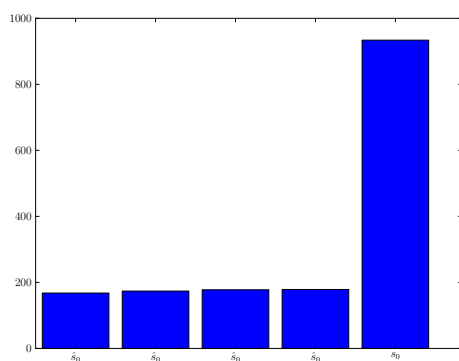


Figure 3.7: The same as above, but here the “smarter” player does not internalize the noise, and plays the ordinary Nash equilibrium.

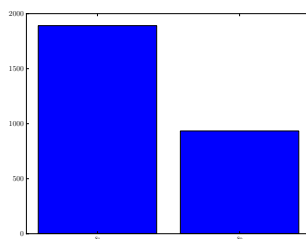


Figure 3.8: A closer comparison of the outcome for the “smart” player from Figure 3.6 (left) versus playing according to the Nash equilibrium as in Figure 3.7 (right).

Subsequent figures, and in particular 3.8, instead show that, as we expect, the positive effect of distrust can be very relevant if the opponents do make the error which is attributed to them, as can be seen from figure.

Chapter 4

Rebuilding trust

Nash equilibrium has been often criticized as being based on too strong assumptions, and in particular on a form of rationality that assumes perfect symmetry and availability of knowledge about all players' strategies. One of the most popular defenses of the concept of Nash equilibrium is however that, though the strong hypothesis may make it, in many situations, an implausible "point of agreement" between agents that meet occasionally, it is much more realistic if seen as the point of convergence of an iterated game.

Let us call β_0 the theoretical Nash equilibrium prediction, and $\hat{\beta}_0$ the same strategy, perturbed with the evaluation error as described in Chapter 2. Let β_1 be the best response (introduced in 3.2) in an auction where all other agents are assumed to follow $\hat{\beta}_0$. Let β_{n+1} be the best response in an auction where all other agents are assumed to follow β_n .¹

Those strategies can be interpreted as an iterated construction of trust, in the following sense: let us assume that when they meet, n "smart" agents simply *do not trust each other*, and think each opponent will play $\hat{\beta}_0$. After the first auction is ran, they discover that instead each one played β_1 . So if the same agents happen to meet again in another auction, they may decide to act according to β_2 . Again, assuming that every agent updates his belief after every auction, based on what he has observed on the behalf of opponents, after n auctions all agents will play β_n . It is hence natural to be curious about what β_∞ looks like.

¹The numerical construction of the various β_i for $i > 1$ is not described, as it is much simpler than for β_1 , because it is the best response to a *deterministic strategy*, and hence, knowing the strategy profile in function of private value, it is relatively easy to reconstruct the CDF and finally build a strategy as described in section 2.1.

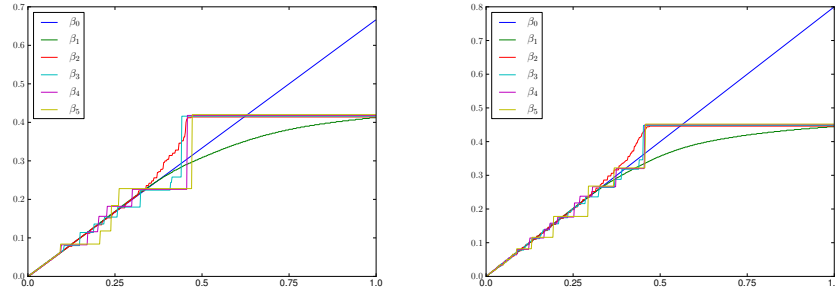


Figure 4.1: Here is represented, in a first price private value auction with 3 (on the left) and 5 (on the right - notice the different scale) players, β_i , for various values of i . β_0 is the straight line (since the Nash equilibrium involves a linear response function). β_1 is the lower, green, one, β_2 is the one red above, and then come β_3 (light blue) and β_4 (purple). Though as i increases the strategies show higher and higher variations, due to the increase of calculations errors, it seems that a form of convergence is observed, which however does not tend toward the Nash equilibrium.

The answer, which can be observed from Figure 4.1, may be surprising: even in our case of an homogeneous evaluation error, as n increases indefinitely, β_n does *not* converge to β_0 , the Nash equilibrium: it tends instead to stay lower. In particular, while the sharpness of higher order profiles is to be attributed to the discreteness of the numeric process involved, the evident effect is that *assuming the presence of initial distrust, high level mentalizing and/or learning will not bring the situation back to "normality"*, to be intended as Nash equilibrium; or if such a convergence will take place, it will be *extremely* slow.

For comparison, 4.2 represents the result of the same study starting from an *ex post* error applied *starting from* the best bid. We can see that the effect is less relevant, but still present: high order strategies would seem to converge, but not on the Nash equilibrium-compliant strategy, and since we know the latter is the only symmetric equilibrium, they *can not* converge to any other functional form. This is even more relevant when we recall, from figure 3.4, that, differently from the case of quantal response functions, *ex post* noise obviously *does not* imply a change of the *expected value of the bid coming from the untrusted opponent*.

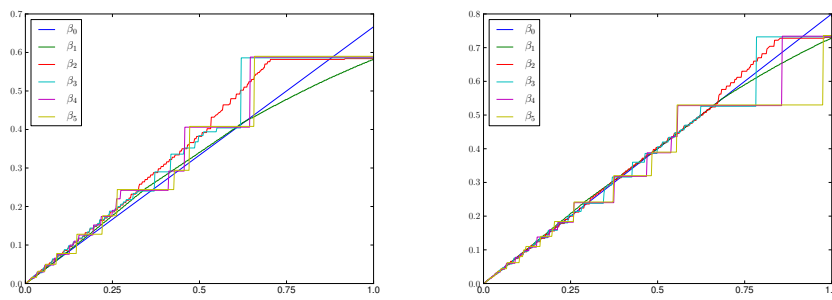


Figure 4.2: Here, β_1 is calculated assuming that (homogeneous, expected value 0) noise is applied *ex post* to the best possible bid, and the other strategies are calculated from β_1 as in the previous figure. Again, the left plot considers an auction with 3 participants, the right one with 5.

Chapter 5

The contest

So far, the target has been on studying optimal adaptation of “smart” players to *clean* situations - such as the one in which *all* other players are playing a given strategy, with an error of a given form.

In this final part, I will consider the results of a “contest” in which strategies seen so far are compared among them by simply playing in an automatized tournament. Though the idea is similar to what can be found, for instance, in [4], given the higher sensibility to the number of participants that is intrinsic in virtually any bidding strategy (which in turn comes from the fact that there is - at least in the framework studied so far - a single object which can be acquired by participants)¹ it would not be interesting to put a high number of automated players in a single, repeated, auction: instead the approach taken is to run a high number of auctions, characterized by the same number of agents, each time extracted from a larger pool of agents.

This does not imply dropping the evolutionary approach used by Rust, Palmer and Miller in their paper - namely, agents with successful strategies become more and more present, while those with less successful ones drop out, and this will be reflected in the strategy distributions of auctions that are run. More precisely, in their 1992 paper they identify a trader’s *fitness* with the amount of capital owned, which evolves as

$$K_i(t) = K_i(t - 1) + \Pi_i(t) - S_i(t)$$

where $S_i(t)$ is the total value of shares received by player i at the

¹By the way, aside from the theoretic reasoning, there is the empirical observation that in reality double auction markets can easily see hundreds of participants not just *taking part*, but *actively participating*, in a single long-lasting auction market; instead auctions last a finite time and are usually ran *ad-hoc* for any single item to be sold.

beginning of game t , $\Pi(t)$ is the total value of assets owned at the end of that same game and $K_i(t)$ is evidently the stock of capital. They then let the fraction $p_j(t)$ of traders of type j be defined as

$$p_j(t) = \frac{K_j(t)}{\sum_{i=1}^I K_i(t)}. \quad (5.1)$$

5.1 Adapting to auctions

While I will in line of principle take a similar approach, the limited number of agents that can reasonably take part in an auction without diluting excessively the variance and significance of observable results leads me to consider each t not as a single auction but instead as a *set* of virtually contemporaneous auctions. Each player, at each time, participates in a randomly chosen one, and the *population share* for each type of players is determined as in (5.1) - the whole approach is simplified by the fact that players don't have any status variable associated with them: capital will be associated only with strategies, and players will, at each time, be assigned strategies according to it.

One point, however, needs particular attention due to the subject of observation being auctions instead than double auction markets: the fact that they are not *zero-sum games*. In the "evolutionary tournament" described in [4], there is no intrinsic motivation why we should expect that as a given strategy becomes more or less popular, it becomes respectively more or less powerful: well on the contrary, we can see a saturation effect which attenuates the growth of more aggressive and successful strategies as they become monopolistic. Instead, in the case of auctions, where the profit is almost always (at least in the majority of studied strategies) positive (or null), the simple fact that a strategy is, at a given moment, more popular can mean that, *on a parity of strategic behaviour*, its expected growth for periods to come is higher.

The solution adopted to face this problem is to *transform* auctions in zero-sum games: given an auction, let \hat{i} be the highest bidder, b_i and v_i respectively the bid and private value of the generic bidder $i \in \{1, \dots, N\}$. Then, the final payoff of the auction is defined as:

$$\hat{u}(i) = \frac{-\frac{u(\hat{i})}{N} + u(i)}{2} = -\frac{v_i - b_i}{2N} + \begin{cases} \frac{v_i - b_i}{2} & \text{if } i = \hat{i} \\ 0 & \text{otherwise} \end{cases};$$

all addends are halved so that the *net aggregate expected impact* on

stocks of a given auction is the same as in non normalized ones. This is done only for the purpose of simplifying comparison among normalized and non normalized in the next section, not for intrinsic motivations.

For the sake of clarity, it must be observed that nevertheless there *can* (and will) very well be *scale effects*, since single strategies may perform well or poorly depending on the composition of the population of auction participant. In particular, “high order” strategies which were defined in chapter 4 are constructed precisely to behave in the best possible way against a *given kind* of opponent. The aim of the compensation mechanism that I just presented is only to reduce sensitivity to low initial (random) perturbations. This can be motivated for the sake of realism - the situation of a market that is exploding and in which acting now has much more influence than acting tomorrow is possible, but we would probably not call it “standard” - but is even more important for reproducibility (low variance of final results) of the simulation itself.

5.2 The results

Figures 5.1 to 5.5 show the outcome of simulations ran with the “clean”, Nash equilibrium-compliant, strategy, denoted as β_0 , the “erroneous” one, denoted with $\hat{\beta}_0$, and higher order ones presented in chapter 4, introduced one at a time.

Given the arguments exposed in the above section, it may be striking to observe, from the comparison of plots of simulations ran with and without normalization, that the former tend to feature apparently similar qualitative phenomena to the latter (though I will focus later on the important differences), but at a much faster pace.

As seen in the previous section, the reason can not be that the *net aggregate impact* of each auction is higher in one case than in the other. The explanation must be searched instead in the fact that with normalization, auction participants which do not win *erode* their stock, hence accelerating the decline of strategy they follow. In fact, dynamics of normalized simulations are very sensible to starting stock quotas: while in the figures 5.1 and following it was set to 100 (recall that the private values in each auction range from 0 to 1), higher values slow down arbitrarily the observed phenomena. The curvature of the trajectories also shows the effect of this chain reaction: the lower stocks go,

the higher is the impact of a single loss auction. But it is important to stress that this *does not mean* that less popular strategies will have more difficulty in winning: it just means that if they loose, the effects will be more evident (and obviously vice versa for more popular ones).

However, another difference between the two kinds of setup deserves much more attention: in all normalized contests, the “clean” (Nash equilibrium-compliant) strategy is the clear winner, while in the others higher order strategy mostly take the lead.² The interpretation of this difference is facilitated by observing the behaviour of non-highest order strategies in the last 4 contests: it is evident that, as could be imagined, high order strategies remain very effective only as long as lower order ones remain present (and even when the presence of the latter declines, they tend to decline very slowly), then have a sharp decrease in efficiency. In non-normalized strategies, where the initial moments are particularly important, having acquired a predominant position facilitates them in growing even more, and this is the reason, together with the fact that inferior strategies never disappear completely, why they result as the best.

In fact, figure 5.2, left, shows the only non-normalized contest in which the higher order strategy does not *immediately* perform better than the clean one.

While the result itself is, as already stated, very dependent on the parameters chosen - namely, the initial stock of 100 - there is an interesting message that we can get by comparing it with the subsequent figures, which feature that same parameters: higher order strategies do not just get an advantage from the presence of bidders with the “one step lower” strategy, but in general they also perform better, with respect to the clean one, if there are more “modified” ones. We could already have an intuition of this fact by observing, in Chapter 4, that those strategies are relatively similar among them, hence may at least partially share the capacity to exploit given situations.

This adds other, slightly more general, evidence that distrust is a self-preserving mechanism.

Simulation exposed in figure 5.6 finally differs from the others in the fact that it does not feature the whole spectrum of strategies, but instead a particularly prominent population of “order 3 thinkers”, together with “order 4” and “order 0” ones. The fact that lines associated with β_3 (one can hardly be seen, because the other superposes to it)

²Those qualitative results came out in all repetitions of the simulations.

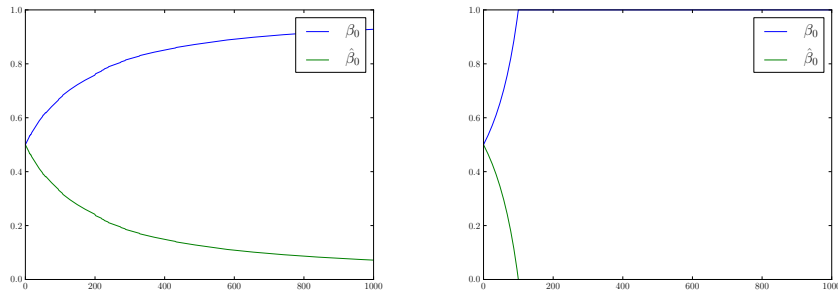


Figure 5.1: First contest: on the right, the normalized version.

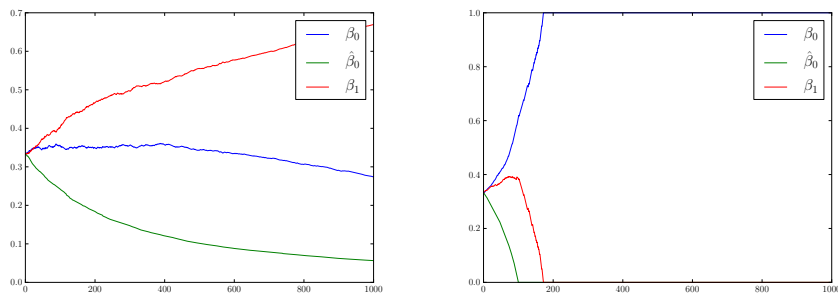


Figure 5.2: Second contest.

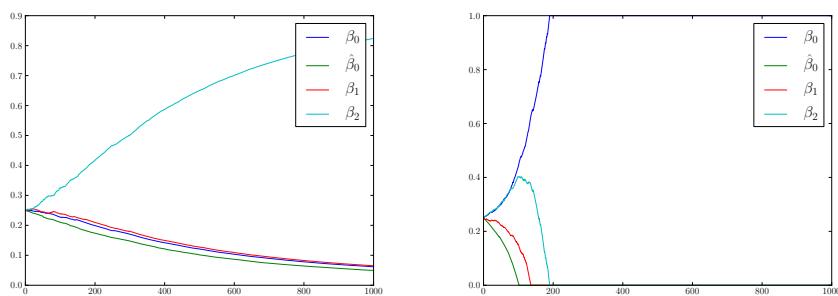


Figure 5.3: Third contest.

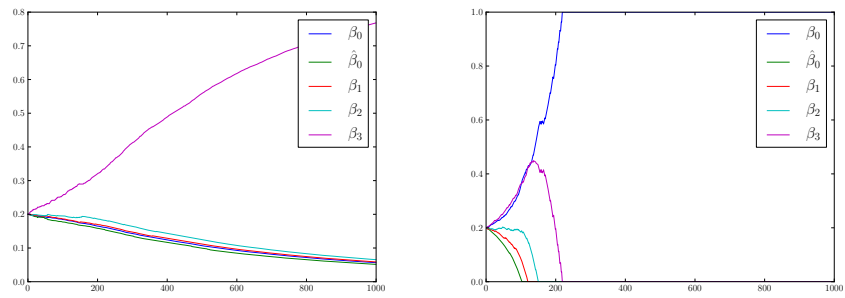


Figure 5.4: Fourth contest.

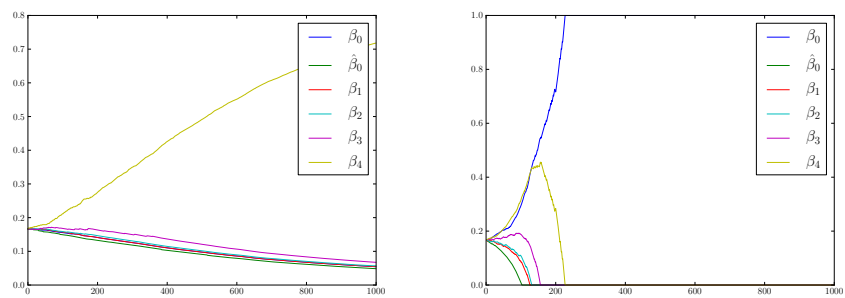


Figure 5.5: Fifth contest.

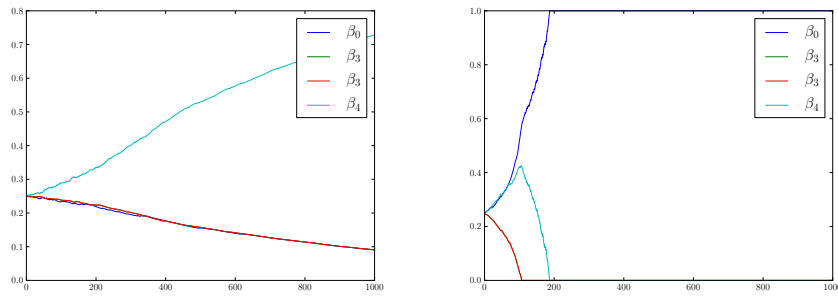


Figure 5.6: Contest with higher β_3 population share.

always decrease does not come as a surprise: what is interesting is the behaviour of β_0 and β_4 , which confirm even more clearly the observations made on the other contests.

Appendix A

The code

This appendix contains the code behind the simulations and drawings. It consists of 5 files:

1. *strategies.py*, which contains the construction of all strategic profiles,
2. *auction.py*, in which the auction process is set up,
3. *contest.py*, which builds the evolutionary contest of chapter 5,
4. *utils.py*, containing few helper functions used in the other modules,
5. *buildfig.py*, a script that creates all the figures present in this study.

All the code is written in the *Python* programming language, and to run it the libraries *scipy* and *sympy* are required.

Moreover, the script *buildfig.py* needs the *matplotlib* library to create plots.

A.1 strategies.py

```
# -*- coding: utf-8 -*-
from __future__ import division

from sympy.interactive import n, x, y, z, integrate, N
from scipy import array, prod, trunc, zeros, arange, average
from utils import intersection, frac_simpl
from random import choice

from pylab import plot

class Debug(object):
    def __call__(self, who, what):
        setattr(self, who, what)

deb = Debug()

N_TICKS = 500
TICK_WIDTH = 1 / N_TICKS

class Strategy(object):
    calculated_dists = {}

    # From Krishna, pag. 18
    FPA_expected = z**n * (x-z) - integrate(y**n, (y, x, z))
    # However, that was in terms of \beta(z)
```

```

FPA_expected = FPA_expected( {z : z * n / (n-1)} )

val_CDF = {}
val_freq = {}
deviations = {}
means = {}
players_n = None
deterministic = True

def __init__(self, **kwargs):
    for i in kwargs:
        setattr( self, i, kwargs[i] )

def reset(self, **kwargs):
    if 'players_n' in kwargs:
        self.players_n = kwargs['players_n']

def profile(self):
    if not self.deterministic:
        raise NotImplementedError

    prof = zeros( N_TICKS )
    for i in range( N_TICKS ):
        prof[i] = self( i * TICK_WIDTH )
    return prof

def build_CDF(self):
    """For non-deterministic strategies, this is only an approximation.
    """
    CDF = zeros( N_TICKS )

    prof = self.profile()

    # prof is sorted (since the strategy is increasing)
    cursor = 0
    for i in range( N_TICKS ):
        while cursor < N_TICKS and prof[cursor] < i * TICK_WIDTH:
            cursor += 1
        if cursor == N_TICKS:
            CDF[i] = 1
        else:
            # Linear interpolation:
            CDF[i] = ( cursor - 1 + ( i * TICK_WIDTH - prof[cursor - 1] ) \
                / ( prof[cursor] - prof[cursor - 1] ) ) * TICK_WIDTH

    return CDF

def build_error_distr(self, expected, error_CDF, tick):
    try:
        error_distrs = self.calculated_dists[expected, error_CDF]
    except KeyError:
        error_distrs = {}
        self.calculated_dists[expected, error_CDF] = error_distrs

    try:
        all_deviations = self.deviations[expected, error_CDF]
    except KeyError:
        all_deviations = self.deviations[expected, error_CDF] = {}

    try:
        deviations = all_deviations[N_TICKS]
    except KeyError:
        deviations = all_deviations[N_TICKS] = zeros( N_TICKS )

    try:
        all_means = self.means[expected, error_CDF]
    except KeyError:
        all_means = self.means[expected, error_CDF] = {}

    try:
        means = all_means[N_TICKS]
    except KeyError:

```

```

        means = all_means[N-TICKS] = zeros( N-TICKS )

    try:
        return error_distrs[frac_simpl( tick , N-TICKS )]
    except KeyError:
        pass

    try:
        error_extrema = error_CDF.extrema
    except AttributeError:
        error_extrema = error_CDF.extrema = self.CDF_extrema( error_CDF )

    val = tick * TICK_WIDTH

    players_n = self.players_n

    exp_val = zeros( N-TICKS )

    for bid_tick in arange( N-TICKS ):
        bid = bid_tick * TICK_WIDTH
        # Expected utility of playing "bid" with value "val":
        exp_val[bid_tick] = N( expected({n:players_n , x:val , z:bid}) )

    deb('exp_val'+str(tick), list(exp_val))
    nash_eq_exp_val = max( exp_val )
    best_resp = list(exp_val).index( nash_eq_exp_val )
#    print "best resp", best_resp

    val_pairs = filter( (lambda x : x[1] + error_extrema[1] > nash_eq_exp_val + error_extrema[0] ),
                        [(t, exp_val[t]) for t in range( N-TICKS )] )

    val_pairs.sort( cmp=(lambda x, y: cmp( x[1], y[1] ) ) )

    tick_distr = zeros( N-TICKS )

    # List of already (partially) processed ticks:
    prev = []
    prev.append( val_pairs.pop() )
    i = 1
    while val_pairs:
        cur = val_pairs.pop()
        for j in prev:
            tick_mass = (prev[-1][1] - cur[1] ) / i
            tick_distr[j[0]] += tick_mass
        prev.append( cur )
        i += 1

    for j in prev:
        tick_distr[j[0]] += (prev[-1][1] + error_extrema[1] - nash_eq_exp_val - error_extrema[0]) / i

    # Normalize so that it is a probability
    tick_distr *= 1 / sum(tick_distr)

    # Calculate standard deviation (from first best, not from mean!):
    deviation = sum( [tick_distr[i] * abs(i-best_resp) * TICK_WIDTH for i in range(N-TICKS)] )

    # Calculate mean:
    mean = sum( [tick_distr[i] * i * TICK_WIDTH for i in range( N-TICKS )] )

    deb('start'+str(tick), list( tick_distr ) )
    deviations[tick] = deviation
    means[tick] = mean
    error_distrs[frac_simpl( tick , N-TICKS )] = tick_distr
    return tick_distr

def cumul_error_distr(self, expected, error_CDF):
    try:
        error_distrs = self.calculated_dists[expected, error_CDF]
    except KeyError:
        self.calculated_dists[expected, error_CDF] = error_distrs = {}

    try:

```

```

        return error_distrs[N-TICKS]
    except KeyError:
        pass

    # If we're here, it wasn't cached. Let's build it.
    bids = zeros( N-TICKS )
    for tick in range( N-TICKS ):
        # The probability correlated to each private value, weighted with
        # the probability of each private value.
        bids += TICK.WIDTH * self.build_error_distr( expected, error_CDF, tick )

    deb( 'start', list(bids) )
    deb( 'stdev', self.deviations[expected, error_CDF][N-TICKS] )
    return bids

def CDF_extrema(self, CDF):
    if CDF(0) < 1:
        max_cursor = 2
        while CDF(max_cursor) < 1:
            max_cursor *= 2
    else:
        max_cursor = -2
        while CDF(max_cursor) >= 1:
            max_cursor *= 2
        max_cursor /= 2

    if CDF(0) > 0:
        min_cursor = -2
        while CDF(min_cursor) > 0:
            min_cursor *= 2
        min_cursor /= 2
    else:
        min_cursor = 2
        while CDF(min_cursor) <= 0:
            min_cursor *= 2

    space = max( abs(min_cursor), abs(max_cursor) )

    # Let's position the cursor at the right extrema of the interval
    # containing the value we're searching:
    if min_cursor < 0:
        min_cursor = 0
    if max_cursor < 0:
        max_cursor = 0

    # Let's do a binary search in that interval:
    while space:
        space /= 2
        if CDF( max_cursor - space ) >= 1:
            max_cursor -= space

        if CDF( min_cursor - space ) > 0:
            min_cursor -= space

    return min_cursor, max_cursor

class Truthful(Strategy):
    # The simplest strategy: bid your signal
    def __call__(self, value, **kwargs):
        return value

class FPNashHomogeneous(Strategy):
    # (2.4) in Krishna, with  $X_i$  being homogeneous
    # Notice the real space from which values are extracted doesn't matter:
    # the agent will play assuming his signal is the highest.
    def __init__(self, start=0, **kwargs):
        Strategy.__init__(self, **kwargs)
        self.start = start
        self.noise = None

    def __call__(self, signal, **kwargs):
        if not self.players_n:
            self.players_n = len( self.auction.players )
        # Given that "signal" is the highest, the expected value of the second

```

```

    # highest is:
    bid = (self.players_n - 1) / self.players_n * signal
    if self.noise:
        bid += self.noise()
    return bid

class FPNashHomogeneousNoisy(FPNashHomogeneous):
    deterministic = False
    def reset(self, **kwargs):
        if 'noise' in kwargs:
            self.noise = kwargs['noise']

class FPNashHomogeneousError(FPNashHomogeneous):
    deterministic = False
    def __init__(self, **kwargs):
        FPNashHomogeneous.__init__(self, **kwargs)
        self.exp = self.FPA_expected

    def reset(self, players_n=10, error_CDF=None, **kwargs):
        if not error_CDF:
            raise

        self.error_CDF = error_CDF
        self.players_n = players_n

    def __call__(self, value, **kwargs):
        if not self.error_CDF:
            return FPNashHomogeneous.__call__(self, value, **kwargs)

        val_tick = int( value / N_TICKS )

        if val_tick not in self.val_CDF:
            distr = self.build_error_distr( self.FPA_expected, self.error_CDF, val_tick )
            deb('distr ', distr)
            distr_CDF = [0]
            for tick in distr:
                distr_CDF.append( distr_CDF[-1] + tick )
            distr_CDF.pop( 0 )
            distr_CDF.append( 1 )

            self.val_CDF[val_tick] = distr_CDF

            self.val_freq[val_tick] = []

            cur = 0
            for i in arange(0, 1, .01):
                while i > distr_CDF[cur]:
                    cur += 1

            self.val_freq[val_tick].append( cur )

        return choice( self.val_freq[val_tick] ) / N_TICKS

    def profile(self):
        """This strategy is nondeterministic; here, we return as "profile" the
        expected value of the response to a given private value."""
        assert self.error_CDF
        self.cumul_error_distr( self.FPA_expected, self.error_CDF )
        return self.means[self.FPA_expected, self.error_CDF][N_TICKS]

class FPNashBestResponseNumeric(Strategy):
    # Initialized with an (n_players-1)-uple of strategies, the distribution of
    # values and (possibly) the distribution of signals (w.r.t. values), it
    # calculates the best response numerically.

    def reset(self, players_n=10, error_CDF=None, noise_shape=(lambda x : x==0), noise_lenght=.2, **kwargs):
        self.N_TICKS = N_TICKS
        self.TICK_WIDTH = TICK_WIDTH = 1 / N_TICKS
        self.players_n = players_n

        noise_ticks = int( noise_lenght // TICK_WIDTH )
        # We must be sure it's an odd number:
        noise_ticks += 1 - noise_ticks % 2

```

```

# Noise shape (sum = 1, odd length, the middle is "no error"):
## Homogeneous of total width 9 ticks
noise_ampl = noise_ticks // 2
noise = array( [noise_shape((i - noise_ampl) * TICK_WIDTH) for i in range( noise_ticks )] )

# Discrete distributions:
distrs = []
n_antagonists = players_n - 1

for antagonist in range( n_antagonists ):

    strategy = FPNashHomogeneous(players_n = players_n)

    # Homogeneous distribution of value (FIXME: add signals != values):
    ticks = array( [TICK_WIDTH] * N_TICKS )
    strat = zeros( N_TICKS + 1 )

    # Evaluate strategy:
    # FIXME: fix things (probability mass) with looking at ticks+1
    strat = [strategy( TICK_WIDTH * tick ) for tick in range( len( ticks ) + 1 )]

    if error_CDF:
        bids = self.cumul_error_distr(self.FPA_expected, error_CDF)
    else:
        bids = zeros( N_TICKS )
        for tick in range( len( ticks ) ):
            # Evaluate strategy:
            interested_tick = int( strat[tick] // TICK_WIDTH )

            # What follows is based on the assumption that strategies
            # will always be increasing.

            while interested_tick * TICK_WIDTH <= strat[tick+1]:
                # Calculate the intersection between intervals
                # [interested_tick, interested_tick+1] and
                # [strategy[tick], strategy[tick+1]]
                inters = intersection( interested_tick * TICK_WIDTH,
                                     ( interested_tick + 1 ) * TICK_WIDTH,
                                     strat[tick],
                                     strat[tick+1] )

                assert inters >= 0, (inters, interested_tick * TICK_WIDTH,
                                     (interested_tick+1) * TICK_WIDTH,
                                     strat[tick], strat[tick+1])
                bids[interested_tick] += ticks[tick] * inters /\
                    (strat[tick + 1] - strat[tick])

                interested_tick += 1

        ticks = bids

    deb('ticks', list(ticks))

    noisy = array( [0.0] * N_TICKS )
    # Distribution of signal (a sort of discrete convolution):
    for tick in range( len( ticks ) ):
        # Cut noise that would go outside of the value distribution:
        useful = ( max( 0, noise_ampl - tick ),
                  min( len( noise ), len(ticks) - tick + noise_ampl ) )
        #
        print "useful", noise[useful[0]:useful[1]]

        # Normalize so that it still is a distribution
        scaling = 1/sum(noise[useful[0]:useful[1]])

        # Apply noise:
        start = max( 0, tick - noise_ampl )
        end = min( len( ticks ), tick + noise_ampl + 1 )
        for i in range( end - start ):
            noisy[start+i] += ticks[tick] * noise[useful[0] + i] * scaling

    ticks = noisy
    deb('noisy', noisy)

```

```

        CDF = [0]
        for tick in ticks:
            CDF.append( CDF[-1] + tick )
        CDF.pop(0)

        distrs.append(CDF)

    self.build_cum_distr(distrs)

def reset_from_CDF(self, CDF):
    """ Accept a list of CDFs and build the best response function to them. """
    self.build_cum_distr(CDF)

def build_cum_distr(self, distrs):
    # CDF of highest bid:
    try:
        self.cum_distr = array( [ prod( [ CDF[i] for CDF in distrs ] ) for i in range( N-TICKS ) ] )
    except IndexError, msg:
        print distrs
        raise IndexError, msg
    deb('cumul', self.cum_distr)

def __call__(self, signal):
    winning_payoff = array([signal - TICK_WIDTH * tick for tick in range( N-TICKS ) ])
    deb('if-win ', winning_payoff)
    expected_payoff = winning_payoff * self.cum_distr
    deb('expected ', expected_payoff)

    best_choice = list(expected_payoff).index( max( expected_payoff ) )

    return best_choice * TICK_WIDTH

def test(self, signal=.5, n_antagonists=10, error_CDF=(lambda x : 10*(x+.05)),
        noise_shape=(lambda x : x==0), noise_lenght=.2):
    # If everything works fine, then when there is no noise this should just
    # return the Nash Equilibrium:

    res = self(signal, n_antagonists=n_antagonists, noise_shape=noise_shape,
               noise_lenght=noise_lenght)
    print "n_ant", n_antagonists+1
    res2 = FPNashHomogeneous(players_n=n_antagonists+1)(signal)

    print "error of", res-res2

```

A.2 auction.py

```

from distributions import *
from random import choice

class Auction(object):
    def __init__(self, mechanism='FPA', evaluation='private',
                 value_distribution=Homogeneous( 0, 1 ),
                 signals_distribution=Homogeneous( 0, 0 )):
        self.mechanism = mechanism
        self.evaluation = evaluation
        self.value_distribution = value_distribution
        # For now, non-noisy signals:
        self.signals_distribution = signals_distribution
        self.players = []
        self.debugging = False

    def add_player(self, strategy):
        self.players.append( strategy )

        strategy.auction = self

    def run(self):
        self._players_n = len( self.players )

        values = self.extract_values()

```

```

        bids = self.ask_bids( values )
        payoffs = self.payoffs( values , bids )
        return payoffs

def extract_values(self):
    if self.evaluation == 'common':
        # Determine real value:
        values = [self.value_distribution()] * self._players_n
    else:
        # Reshuffle for each player:
        values = [self.value_distribution() for i in range( self._players_n )]

    return values

def ask_bids(self , values):
    """Players interplay."""
    players_n = len( self.players )

    self.debug( values )

    # Determine signals:
    signals = []
    for agent in range( players_n ):
        # Distributions must accept the optional parameter "around":
        signals.append( self.signals_distribution( around=values[agent] ) )

    # "Ask" for bids:
    bids = []
    for i in range( len( self.players ) ):
        # For now, FPA:
        bids.append( self.players[i]( signals[i] ) )

#     print bids

    return bids

def payoffs(self , values , bids):

    highest = max( bids )
#     print highest
#     print bids
#     print self._players_n
    winners = filter( (lambda i : bids[i] == highest), range( self._players_n ) )
    # Solve possible ties:
    winner = choice( winners )

    payoffs = [0] * len( values )

    if self.mechanism == 'FPA':
        payoffs[winner] = values[winner] - bids[winner]
    elif self.mechanism == 'SPA':
        self.debug( "non-winners", filter( (lambda b : b != highest), bids ) )
        second_highest = max( filter( (lambda b : b != highest), bids ) )
        self.debug( "second_highest", second_highest )
        payoffs[winner] = values[winner] - second_highest

#     print payoffs
    return payoffs

def debug(self , *args):
    if self.debugging:
        for arg in args:
            print arg,
        print

```

A.3 contest.py

```
#!/usr/bin/python
```

```

from __future__ import division
from auction import Auction
from utils import AuctionCounter
from random import choice, randint
from scipy import array, average

def contest(strategies, pls_per_auction, iterations=1000, normalize=False):
    strat_n = len( strategies )

    # The number of contemporaneous auctions:
    cont_auct = 10
    total_players_n = pls_per_auction * cont_auct

    # The stock "owned" by each strategy - all the same, at start:
    stock = array( [100] * strat_n )

    c = AuctionCounter()
    c.reset(iterations, ticks=100)

    # This records stock time series for each strategy:
    historical = []

    while c():
        # Here starts a given "time" t.
        players = []

        # Inside a given time unit, payoffs are remembered by strategies:
        for strategy in strategies:
            strategy.payoffs = 0

        total_stock = sum( stock )
        stock_percentages = array( stock ) / total_stock
        historical.append( array( stock_percentages ) )

        # Each "place" in an auction corresponds to a given percentage:
        price = 1 / total_players_n

        # Each strategies "buys" all places it can afford:
        for i in range( strat_n ):
            places = stock_percentages[i] // price
            players.extend( [strategies[i] for j in range( int( places ) )] )
            stock_percentages[i] = stock[i] % price

        # Remaining strategies are attributed to "best offerers":
        while len( players ) != total_players_n:
            best_offer = max( stock_percentages )
            best_offerers = filter( (lambda i : stock_percentages[i] == best_offer),
                                   range( strat_n ) )
            i = choice( best_offerers )
            players.append( strategies[i] )

        # Now players are chosen, let's put them into auctions:
        for auct in range( cont_auct ):

            a = Auction('FPA')

            for i in range( pls_per_auction ):
                extraction = randint( 0, len( players ) - 1 )
                player = players.pop( extraction )
                a.add_player( player )
            payoffs = a.run()
            # Attribute payoffs
            if normalize:
                opportunity_cost = average( payoffs ) / 2
                payoffs = array( payoffs ) / 2
            else:
                opportunity_cost = 0
            for i in range( pls_per_auction ):
                a.players[i].payoffs += payoffs[i] - opportunity_cost

        for i in range( strat_n ):
            stock[i] += strategies[i].payoffs

```

```

    return historical

def demo():
    from strategies import FPNashHomogeneous, FPNashBestResponseNumeric, FPNashHomogeneousError

    strtgs = []

    p = FPNashHomogeneous()
    p.reset(players_n=5)

    strtgs.append( p )

    p = FPNashHomogeneousError()
    error_CDF = (lambda x : 20*(x+.025))
    p.reset( players_n=5, error_CDF=error_CDF )

    strtgs.append( p )

    p = FPNashBestResponseNumeric()
    p.reset( players_n=5, error_CDF=error_CDF )

    strtgs.append( p )

    p2 = FPNashBestResponseNumeric()
    p2.reset_from_CDF( [p.build_CDF()*4 ] )

    strtgs.append( p2 )

    p3 = FPNashBestResponseNumeric()
    p3.reset_from_CDF( [p2.build_CDF()*4 ] )

    strtgs.append( p3 )

    p4 = FPNashBestResponseNumeric()
    p4.reset_from_CDF( [p3.build_CDF()*4 ] )

    strtgs.append( p4 )

    return contest(strtgs, 5)

if __name__ == '__main__':
    demo()

```

A.4 utils.py

```

from scipy import sqrt

def intersection(a, b, c, d):
    """ Positive: length of intersection of [a,b] and [c,d].
        Negative: opposite of their distance."""
    return min(b, d) - max(a, c)

def frac_simpl(a, b):
    # Return the integer ratio a/b simplified.

    assert( b >= a )
    assert b % 1 == a % 1 == 0 , "%d/%d" % ( a, b )

    # Euler:

    p = b
    q = a
    while q:
        r = p % q
        p = q
        q = r

    return a/p, b/p

class AuctionCounter(object):
    def __init__(self):
        self.count = 0

```

```
def __call__(self):
    self.count += 1
    if not self.count % self.ticks:
        print "%d of %d" % (self.count, self.auct_iterations)
    return self.count <= self.auct_iterations

def reset(self, auct_iterations, ticks=1000):
    self.auct_iterations = auct_iterations
    self.count = 0
    self.ticks = ticks
```


Bibliography

- [1] V. Krishna. *Auction theory*. Academic press, 2009.
- [2] U. Malmendier, A. Szeidl, UC Berkeley, and N.U.C. Berkeley. Fishing for Fools. Technical report, Citeseer, 2008.
- [3] R.D. McKelvey and T.R. Palfrey. Quantal response equilibria for normal form games. *Games and Economic Behavior*, 10(1):6–38, 1995.
- [4] J. Rust, R. Palmer, and J. Miller. Behaviour of trading automata in a computerized double auction market. *The Double Auction Market: Institutions, Theories, and Evidence*, pages 155–198.