Non-linear externalities: A computational estimation method

Giulio Bottazzi [*]
Fabio Vanni [*]

[*] Institute of Economics and LEM, Scuola Superiore Sant'Anna, Pisa, Italy

2014/01 November 2015
ISSN(ONLINE) 2284-0400

A numerical estimation method for discrete choice models with non-linear externalities

Giulio Bottazzi [*]

Scuola Superiore Sant'Anna
Institute of Economics
Pisa, Italy.

Fabio Vanni [†]

Scuola Superiore Sant'Anna
Institute of Economics
Pisa, Italy.

January 2014

Abstract

This paper presents a stochastic discrete choice model with non-linear externalities and a related methodology to carry out inferential analysis on its parameters. Such framework allows to disentangle the role of the intrinsic features of alternatives from the effect of externalities in determining individual choices. The stochastic process underlying the model is demonstrated to be ergodic, so that numerical methods are adopted to estimate the parameters by χ^2 minimization and evaluate their statistical significance. In particular, optimization rests on computational procedures that consider also iterative methods such as successive parabolic interpolation. Comparisons among various computational techniques and simulation implementations are analyzed. For completeness, the χ^2 is compared to the approach of maximum likelihood optimization, underling the advantages of the first in this model.

JEL codes: C12, C13, C46, C52, R12.
Keywords: Externalities, Heterogeneity, Computational methods, Firm localization.

[*]E-mail: `giulio.bottazzi@sssup.it`; Corresponding author
[†]E-mail: `fabio.vanni@sssup.it`

1 Introduction

A variety of individual choices are determined by the intrinsic features of the object of choice as well as by the choices of other individuals. For instance, a firm might choose where to place its plants looking also at the localization of other firms, be it to cluster with them in a Silicon Valley style, or to be as far as possible from them so as to escape local competition. In all these cases, the number of agents that make a certain choice contributes to determining how many other agents will make the same choice, thus generating interdependencies in the decisions of agents. The present model belongs to the class of generalized Polya urn schemes (see Garibaldi and Scalas, 2010). The probability for an alternative to be chosen, however, will depend also on its intrinsic features, represented by some positive value. As a consequence, the model is characterized as an irreducible Markov process that converges to a unique asymptotic distribution,irrespective of initial conditions.

However the purpose of this paper is to address a more precise specification of the computational methods found in the previous paper Giulio Bottazzi and Vanni (2014) in which the authors present a more detailed introduction on this topic and a particular application to firm localization.

2 Model

The discrete choice model presented here extends the one in Bottazzi and Secchi (2007) to the case of quadratic externalities. In parallel, also the type of inferential analysis introduced in Bottazzi and Gragnolati (2012) is here modified to account for non linearities. Those papers deals with the general framework where a population of N agents has to choose among L alternatives, and each generic agent i does so maximizing individual utility. Labeling as n_l the number of agents that have chosen alternative l and as the configuration $\boldsymbol{n} = (n_1, \ldots, n_L)$ the corresponding occupancy vector, the utility that agent i associates to alternative l depends on the choices made by other agents. In terms of firms localization agents are the firms and the locations are the alternatives. In an operational point of view the linear-case model consists in new firms entering the economy, and selecting a site in which to place their activities. Conversely incumbent firms from every location face some probability of leaving them (random dying process). Once a firm has left the economy the probability of each location to be selected is proportional to:

$$p_l \sim a_l + bn_l, \tag{1}$$

where a_l represents the intrinsic attractiveness captures the intrinsis feature offered by the location l (higher demand, lower marginal costs, better infrastructures, etc.). On the other hand b represents the interdependence factor that captures the presence of positive externalities of localization which generate and act within the sector, rather than in single locations.

The extension of such model to a non-linear coefficients relies on the function shape of the probability :

$$p_l \sim a_l + bn_l + cn_l^2, \tag{2}$$

where the c coefficient represents the non- linear externalities. To get the complete probabilities eqs (2) has to be normalized as:

$$p_l = fraca_l + bn_l + cn_l^2 A + bN + c \sum_{l=1}^{L} n_l^2, \tag{3}$$

where $A = \sum_{l=1}^{L} a_l$, $N = \sum_{l=1}^{L} n_l$, with $a_l \geq 0 \forall l$ and $b \geq 0$. A positive estimate of the quadratic parameter would signal that the probability for a firm to choose a certain location grows more than linearly in the number of rms already settled in the same location. Vice versa, a negative estimate would signal the presence of spatial congestion.

While still allowing to estimate the presence of externalities in high dimensional problems, the extension to the quadratic case impedes to derive a closed-form likelihood function with the aim to evaluate the best parameters that fit the observations. Therefore, the parameter estimates are here obtained computationally, and so is their variance. In particular, point estimates are the result of χ^2 minimization resting on successive parabolic interpolation Brent (1973), while their variance is estimated through Monte Carlo simulations. It follows that the statistical significance of the parameter estimates can be evaluated through their p-values. This entire methodology has been realized in the Matlab program language.

3 Domain definition

The present model describes the evolution of the choices of a population across alternatives in probabilistic terms. Given an initial configuration $\boldsymbol{n}_0$ and the set of parameters $(\boldsymbol{a}, b, c)$, the realization of the stochastic process can be simulated numerically. In the case of non-linear externalities (i.e. $c \neq 0$), Both processes (linear and non-linear) have been demonstrated to be when the transition probabilities are non-zero (see Giulio Bottazzi and Vanni (2014) and Bottazzi and Secchi (2007)). If this condition is attained, then the process converges asymptotically to the equilibrium distribution $\pi(\boldsymbol{n}; \boldsymbol{a}, b, c)$. Therefore, the values of the parameters $(\boldsymbol{a}, b, c)$ must be such as to ensure non-zero transition probabilities. In particular, since $u_l(g_l)$ maps into $\mathbb{R}_{\geq 0}$, it must generally hold that $g_l > 0$ for any $l \in (1, \ldots, L)$. These constraints allow to identify a relation among the values of the unknown parameters, especially for what concerns the lower bound of c. In turn, this will inform numerical simulations.

We can obtain a definition of the lower bound of c straightforwardly from the constraint $g_l > 0$:

$$c > -\frac{a_l + bn_l}{n_l^2} \ , \quad \forall l = 1, \ldots, L \ , \tag{4}$$

$$> -\frac{\min\{a_l\} + b \max\{n_l\}}{\max\{n_l\}^2} \ , \tag{5}$$

$$c_{\min} = -\frac{\min\{a_l\} + bN}{N^2} \ . \tag{6}$$

Equation (6) corresponds to "full concentration", that is when all N agents choose the same alternative l. If the condition $g_l > 0$ is respected for the alternative l that is chosen by all agents under this extreme scenario, then it will also be respected for all other alternatives under any other scenario. In parallel, conditions (4)–(6) imply a restriction also on the denominator of equation (3), namely

$$A + bN + c\sum_{l=1}^{L} n_l^2 > 0 \ , \tag{7}$$

$$A - \min\{a_l\} > 0 \ , \text{with } c \geq c_{\min} \ , \tag{8}$$

where $A = \sum_{l=1}^{L} a_l$.

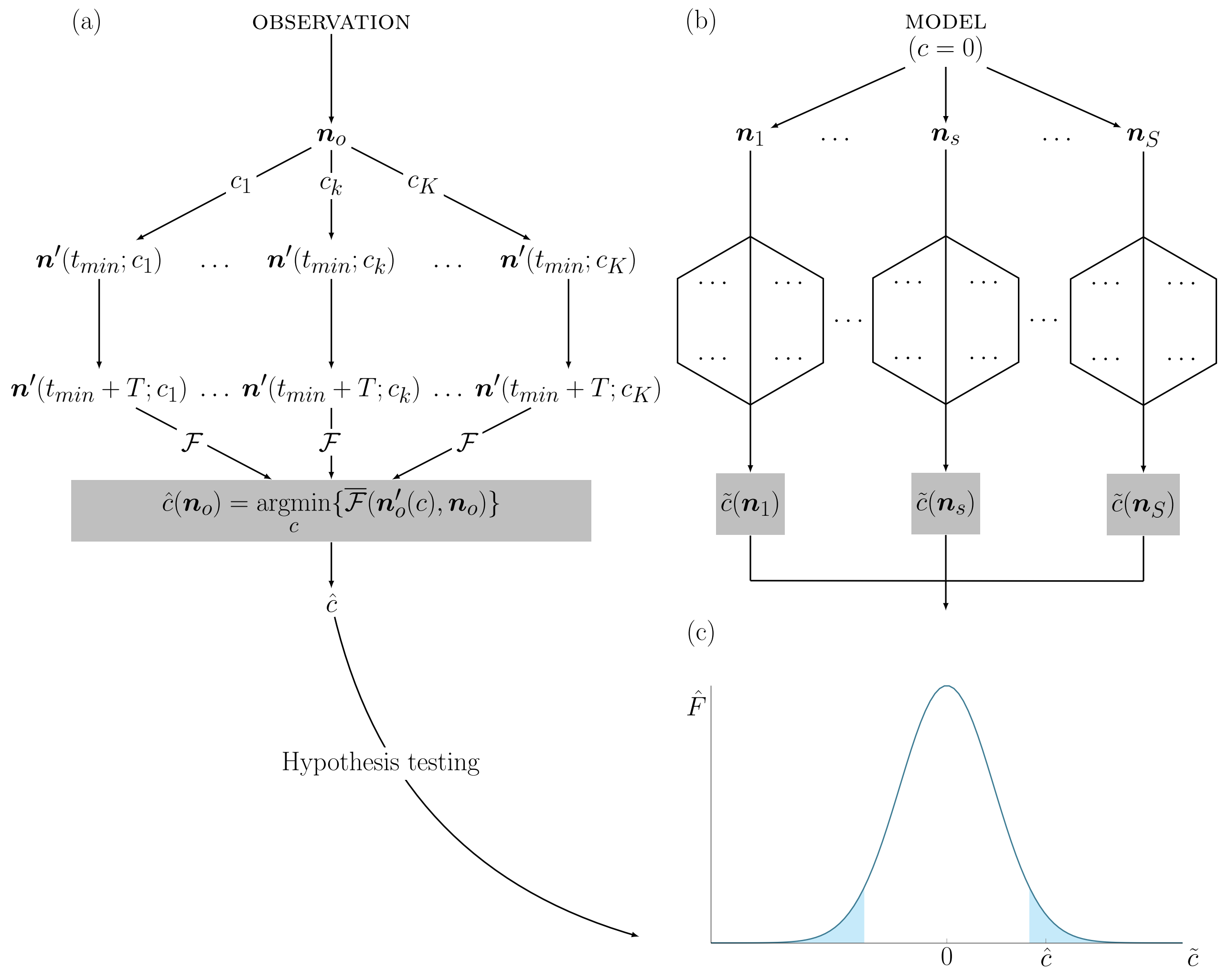
Furthermore, the interest of the present analysis is on small deviations from linearity. Therefore, if an estimate of the "typical" occupancy $\tilde{n}$ were available, the value of the non-linear parameter would be approximately in the order of $1/\tilde{n}$.

4 Estimation and inferential analysis

Given the observed $\boldsymbol{n}_o$, one can investigate whether such distribution is statistically "compatible" with a set of predefined parameters $(\hat{\boldsymbol{a}}, \hat{b}, \hat{c})$. Under the assumption that the model described in Section 2 with a set of parameters $(\boldsymbol{a}, b, c)$ represents the *true* data generating process, testing the statistical "compatibility" of $(\hat{\boldsymbol{a}}, \hat{b}, \hat{c})$ amounts to performing an inferential analysis on the null hypothesis $H_0(\boldsymbol{a} = \hat{\boldsymbol{a}}, b = \hat{b}, c = \hat{c})$ against the alternative. Practically, this translates in estimating the parameters that govern the model through equation (3), while testing also whether these estimates are statistically different from zero. To simplify the exposition, we illustrate the underlying reasoning by referring to the estimate of a single parameter, namely c. Specifically, the inferential analysis will evaluate whether the null hypothesis $c = 0$ can be rejected, at a given significance level, based on the observed distribution $\boldsymbol{n}_o$.

The point estimate of c is obtained by searching numerically for the value $\hat{c}$ which generates an equilibrium configuration closest to the observed $\boldsymbol{n}_o$. The search takes place within a predetermined set $[c_{min}, c_{max}]$, which contains 0. The lower bound of the set, c_{min}, is defined according to equation (6), whereas the definition of c_{max} is set according to the search method in use. In fact, we will implement the numerical search for $\hat{c}$ following different procedures, namely the grid method and successive parabolic interpolations. Each of these approaches is detailed in Section 4.4. Yet, to conduct the reader toward a first generic understanding of the estimation procedure, it is convenient to refer hereby to the grid method. Once this is understood, the rest will follow more easily.

Figure 1a illustrates how the point estimate $\hat{c}$ is determined in the grid method. Starting from the observed configuration $\boldsymbol{n}_o$, the model is simulated with K different values of $c \in [c_{min}, c_{max}]$ and fixed values for the other parameters. For each value of c, the model is run for a transient of t_{min} time steps which are sufficient to reach the equilibrium configurations $\{\boldsymbol{n}'(t_{min}; c_1), \ldots, \boldsymbol{n}'(t_{min}; c_K)\}$. Then, each configuration is further evolved for a sufficiently large number of steps T. Hence, K independent trajectories $\{\boldsymbol{n}'(t_{min} + T; c_1), \ldots, \boldsymbol{n}'(t_{min} + T; c_K)\}$ are obtained. The goal is then to identify the particular value $\hat{c} \in (c_1, \ldots, c_K)$ whose



(a)
OBSERVATION
$\boldsymbol{n}_o$
c_1
c_k
c_K
$\boldsymbol{n}'(t_{min}; c_1)$ … $\boldsymbol{n}'(t_{min}; c_k)$ … $\boldsymbol{n}'(t_{min}; c_K)$
$\boldsymbol{n}'(t_{min}+T; c_1)$ … $\boldsymbol{n}'(t_{min}+T; c_k)$ … $\boldsymbol{n}'(t_{min}+T; c_K)$
$\mathcal{F}$
$\hat{c}(\boldsymbol{n}_o) = \underset{c}{\operatorname{argmin}}\{\overline{\mathcal{F}}(\boldsymbol{n}'_o(c), \boldsymbol{n}_o)\}$
$\hat{c}$
Hypothesis testing
(b)
MODEL
$(c = 0)$
$\boldsymbol{n}_1$ … $\boldsymbol{n}_s$ … $\boldsymbol{n}_S$
$\tilde{c}(\boldsymbol{n}_1)$
$\tilde{c}(\boldsymbol{n}_s)$
$\tilde{c}(\boldsymbol{n}_S)$
(c)
$\hat{F}$
0
$\hat{c}$
$\tilde{c}$

Figure 1: Estimation approach (grid method).

associated trajectory $\boldsymbol{n}'(t_{min} + T, \hat{c})$ is closest as an average over time to $\boldsymbol{n}_o$. In particular, the comparison between any $\boldsymbol{n}'(t_{min} + T)$ and $\boldsymbol{n}_o$ is performed according to an objective function Θ defined in terms of a distance measure $\mathcal{F}$ (see subsection 4.1). Formally,

$$\hat{c}(\boldsymbol{n}_o, t_{min}, T) = \underset{c\in(c_1,\ldots,c_K)}{\operatorname{argmin}} \{\Theta(\boldsymbol{n}'(c), \boldsymbol{n}_o)\} \tag{9}$$

$$\Theta = \overline{\mathcal{F}}(\boldsymbol{n}'(t; c), \boldsymbol{n}_o), \tag{10}$$

where t_{min} is the minimum number of time steps needed to reach convergence. Besides $\overline{\mathcal{F}}$ represented a generic mean function over the time. Overall, this process delivers the point estimate $\hat{c}$, whose statistical significance has then to be tested.

Figures 1b-c illustrate the Monte Carlo approach that is adopted to test the statistical significance of $\hat{c}$. Initially, the model is run under the null hypothesis $c = 0$ for $s \in (1, \ldots, S)$ different realizations, thus obtaining the set of replicas $\{\boldsymbol{n}_1, \ldots, \boldsymbol{n}_S\}$. Then, the same search procedure described above is performed for each replica. Namely, each configuration $\boldsymbol{n}_s$ is further evolved for a time $t_{min} + T$ with different parameter values $c \in (c_1, \ldots, c_K)$, thus reaching the new configurations $\{\boldsymbol{n}'_s(t_{min} + T; c_1), \ldots, \boldsymbol{n}'_s(t_{min} + T; c_K)\}$. One of these will be closest to $\boldsymbol{n}_s$, and consequently its associated value of c will be identified as the solution for

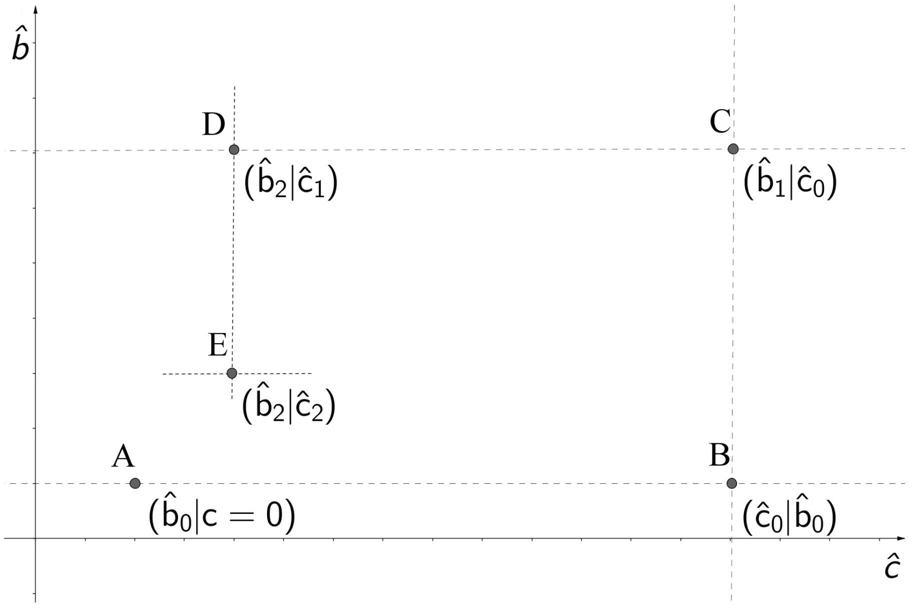


Figure 2: Multistage estimation procedure that follows the line of the coordinate descend method. Each step represents a minimization of a given optimization function. The step forward is realized in terms of staistical significance of the found value.

the same minimization problem as (9). Since this procedure is repeated for each replica, one ends up with S independent estimates $\tilde{c}$ under the null hypothesis $c = 0$. From these it is then possible to calculate the p-value of the point estimate $\hat{c}(\boldsymbol{n_o})$ under the null hypothesis. Let $\hat{F}(c)$ be the empirical distribution associated to the S estimates $(\tilde{c}(\boldsymbol{n}_1), \dots, \tilde{c}(\boldsymbol{n}_S))$ under the null $H_0 : c = 0$; then, the p-value of $\hat{c}(\boldsymbol{n}_o)$ is given by $\hat{F}(\tilde{c} \leq -|\hat{c}|) + \hat{F}(\tilde{c} \geq |\hat{c}|)$.

If focusing on the estimate of a single parameter might have eased the exposition thus far, the present estimation method relies on a multistage procedure which allows to estimate multiple parameters. In fact, the kind of estimation exercises that are relevant for the model presented in Section 2 concern a multidimensional space, where each dimension corresponds to one of the parameters to be estimated. This leads to a multistage procedure, in which each single parameter estimate is obtained conditionally to an initial value of the other parameter(s). Such an iterative cycle stops as soon as convergence is reached. Figure 2 gives an illustration of this procedure for a two dimensional case. Point **A** identifies the estimate $\hat{b}_0$ obtained under the parametric restriction $c = 0$. From there the algorithm searches for an estimate of c under the restriction $b = \hat{b}_0$, thus reaching $\hat{c}_0$ and identifying point **B**, and so on to points **C**, **D**, and **E**. In particular, the algorithm stops when the new set of estimates is not statistically different from the set encountered at the previous search round.

Having provided a general picture of the present estimation method, it is now time to focus in greater detail on two of its key elements, that is the objective function and the optimization algorithm through which $\hat{c}$ and the various $\tilde{c}$ are determined.

4.1 Occupancy and Objective function

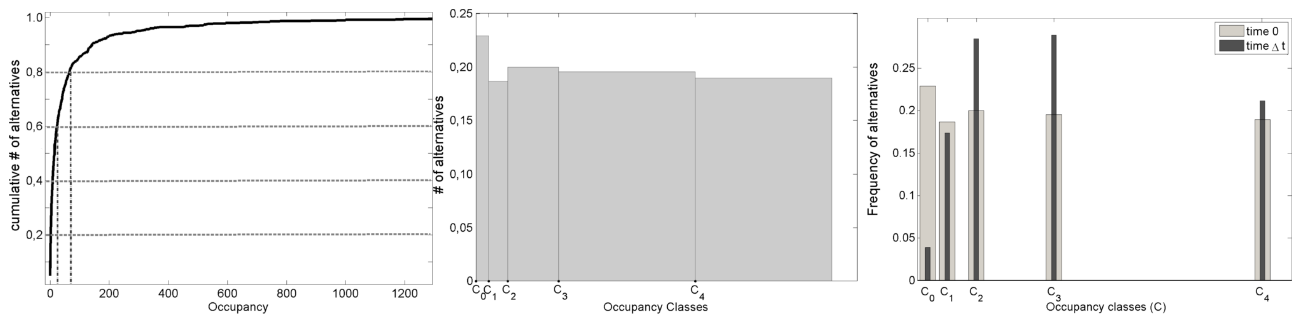
As mentioned above, the objective function Θ compares two configurations of the system. In this respect, we explore two approaches which will be described below. Again, to simplify the exposition, the following description assumes that only the parameter c is unknown.

The first and most natural approach consists in searching for the particular $\hat{c}$ which maximizes the likelihood of the observed configuration $\boldsymbol{n}_o$. In this setting, the objective function compares a simulated configuration $\boldsymbol{n}'(t;c)$ with the observed configuration $\boldsymbol{n}_o$ according to $\mathcal{F}(\boldsymbol{n}'(t;c), \boldsymbol{n}_0) = -\delta_{\boldsymbol{n}'(t;c),\boldsymbol{n}_0}$. Hence, $\mathcal{F}$ has value -1 when $\boldsymbol{n}(t;c)$ and $\boldsymbol{n}_o$ are identical and 0 when they differ. This count yields the negative likelihood that $\boldsymbol{n}_0$ is generated by a particular value of c, and that is the function to be minimized in problem (9).

Unfortunately, the applicability of maximum likelihood estimation turns out to be limited in the present case due to two intertwined issues. Each of them is further detailed in Appendix A, but it is still worth to introduce them here by means of an example. The first issue has to do with ambiguity. To see how, imagine a situation in which $N = 2$, $L = 2$, $\boldsymbol{a} = (1, 1)$, and the observed configuration reads $\boldsymbol{n}_o = (2, 0)$. In this scenario, externalities are clearly at play since one alternative is chosen much more frequently than the other despite their homogeneous intrinsic attractiveness. In particular, the parameter values that generate $\boldsymbol{n}_o = (2, 0)$ are necessarily the same as those that would generate $\boldsymbol{n}' = (0, 2)$. However, $\mathcal{F}(\boldsymbol{n}', \boldsymbol{n}_0) = -\delta_{\boldsymbol{n}'(t;c),\boldsymbol{n}_0}$ regards $\boldsymbol{n}_o = (2, 0)$ and $\boldsymbol{n}' = (0, 2)$ as being different one from the other, although they have the same generating parameters. The second issue reinforces the former through the rise of computational costs when L is large. More precisely, the probability to regard as different two configurations that are actually generated by the same underlying parameters increases with the size of L. To stick to the previous example, when $N = 2$, $L = 2$, $\boldsymbol{a} = (1, 1)$ and $\boldsymbol{n}_o = (2, 0)$, the observed configuration can be "confused" only with $\boldsymbol{n}' = (0, 2)$. As soon as dimensionality increases to $L = 3$, an observed configuration $\boldsymbol{n}_o = (2, 1, 0)$ could be "confused" with $\boldsymbol{n}' = (2, 0, 1)$, $\boldsymbol{n}' = (0, 2, 1)$, $\boldsymbol{n}' = (1, 2, 0)$, $\boldsymbol{n}' = (1, 2, 0)$, or $\boldsymbol{n}' = (1, 0, 2)$. Therefore, the number of time steps for which the model needs to be run in order to generate sufficiently accurate statistics on $\boldsymbol{n}_o$ increases more than linearly with the size of L. In this sense, maximum likelihood estimation entails also high computational costs for large L.

In fact, computational costs can be attenuated by moving from a point to a local estimation of the maximum likelihood. As further detailed in Appendix A, we have explored this road by resorting to different techniques ranging from decimation, to convolution smoothing, to the more advanced k-nearest neighbors estimation. However, while sensibly reducing computation costs, none of these methods can entirely solve the ambiguity that is intrinsic to the numerical maximum likelihood approach. To give an idea of these local maximum likelihood methods, let us turn to the k-nearest neighbors estimation. This method compares the K simulated distributions $\{\boldsymbol{n}'(c_1), \ldots, \boldsymbol{n}'(c_K)\}$ and the observed distribution $\boldsymbol{n}_o$ according to a distance measure, which we define as the euclidean distance. The simulated distributions $\{\boldsymbol{n}'(c_1), \ldots, \boldsymbol{n}'(c_K)\}$ are then ordered according to their distance relative to $\boldsymbol{n}_o$, and only the k that are closest to $\boldsymbol{n}_o$ enter Θ in equation (9).[1] In this sense, the k-nearest neighbors is a local method, which allows to obtain a smoother estimate. Then, the objective function estimate becomes $\Theta(\boldsymbol{n}_o) \propto 1/R_k^L$, where R_k is the euclidean distance between the estimation point $\boldsymbol{n}_o$ and its k-th closest neighbor. Overall, this local maximum likelihood method attenuates some of the "confusion" discussed

[1]Notice that, for k-nearest neighborhood estimation, the minimization problem in equation (9) does not longer correspond to the time average $\Theta = \bar{\mathcal{F}}$ as discussed in the previous paragraph.



(a) (b) (c)

Figure 3: Binning procedure.

in the examples above. For instance, given a uniform intrinsic attractiveness $a_l = a$ as in the previous examples, the configurations $\boldsymbol{n_o} = (2, 1, 0)$ and $\boldsymbol{n'} = (2, 0, 1)$ would now be regarded as closer than they were with the point maximum likelihood. Nonetheless, they are still not considered as identical, even if they are generated by the same underlying parameter values. In this sense, even local maximum likelihood approaches do not manage to solve entirely the problem of ambiguity. And this is even more the case for large L.

That is why we move to a χ^2 minimization approach. In this alternative setting, the distance between two configurations is measured through the χ^2 function applied on occupancy classes. An occupancy $f(n)$ is defined as the number of alternatives chosen by n agents. For instance, $f(0)$ is the number of alternatives chosen by zero agents, $f(1)$ is the number of alternatives selected exactly by one agent, and so on. It follows that the sum of all occupancies is equal to the number of available alternatives, that is $\sum_{n=0}^{N} f(n) = L$. Occupancies are then grouped into classes $C_1, \dots, C_J$ of variable width and constant size:

$$\begin{cases} C_j = [c_j, c_{j+1}) \qquad j = 0, 1, \dots, J, \\ f(C_j) = \sum_{n \in C_j} f(n), \\ f(C_j) = f(C_i) \qquad \forall i, j = 0, 1, \dots, J, \end{cases} \tag{11}$$

The resulting histograms tends to be flat, since each bin counts the same number of occurrences (see Figure 3b). Notice that classes are computed only on the observed configuration $\boldsymbol{n_o}$, and then they are maintained constant for all other simulated configurations. By doing so, it is ensured that the bins are chosen according to the real data while the cost of defining new classes is limited to the moment in which a new observation $\boldsymbol{n_o}$ is considered. Moreover, this also allows to have an immediate visual hindsight on how two configurations may possibly differ (see Figure 3c). Given this definition of occupancy classes, the distance function used to compare configurations via the objective function Θ.

4.1.1 The χ^2 objective function

The reliability of the use of the optimization procedure of the objective function is based on fine settings of several parameters and values. The main numerical issue is the convergence to the ergodicity condition of the time series that represents the realizations of the configuration vector.

Stationarity and ergodicity are properties of the underlying stochastic process and not of a single realization. Stationarity is the property that the expected values of the moments of the process are independent of the temporal index. Ergodicity is the property that the expected values of the moments of the process are equal to the time averages of the moments of the process. Since the expectation operator is the average over all the realizations of the process, in general it is not possible to say anything for sure with just a single realization of the process. If, however, we have that the process is ergodic, we can evaluate ensemble averages from the time average estimated from a single realization. Ergodicity for linear and non-linear models discussed in the present paper has been proved in Giulio Bottazzi and Vanni (2014). Anyway as for time series we need some deeper considerations. The ergodicity of the time series of the configuration vector $\boldsymbol{n} = [n_1, \ldots, n_k, \ldots, n_N]^T$ can be checked via the wide-sense ergodicity criteria

$$n_k(t) = \lim_{T\to\infty} \frac{1}{T} \int_0^T n_k(t')dt' = E[n_k] \ , \tag{12}$$

$$\hat{\gamma}_{n_k}(\tau) = \lim_{T\to\infty} \frac{1}{T} \int_0^T n_k(t')n_k(t'+\tau)dt' = E[n_k(t), n_k(t+\tau)] \ , \tag{13}$$

where $\hat{\gamma}$ is the covariance function. Since we know that the Markov process under study is ergodic, the first-order stationary processes condition is sufficient to detect if the process has reached the steady state. So we can check the first two moments of the distribution $\bar{n}_k = E[n_k] = m_k$ and $\mathrm{Var}(n_k) = E[(n_k - m_k)^2]$.

As a rule of thumb, for our purpose as stationary time series we mean a flat looking series, without trend, constant variance over time, a constant autocorrelation structure over time and no periodic fluctuations. Since we start from configurations that are close to the equilibrium it is necessary to wait a time until the trajectory reaches the expected ensemble variance. This time for equilibrium has be determined according to the starting configuration, the number of locations and the total number of units. This time is what we have called t_{min}.

The evaluation of the objective function rests on numerical estimation of the distance between the compared occupancy distribution.

The ergodic condition is valid in a long run term for time series of our distance between occupancy distributions $\mathcal{F}$. The rate of convergence also in this case helps us to meet the stationarity condition for the time average in the computation of the objective function in terms of occupation distances. That is,

$$\Theta = \overline{\mathcal{F}}(\boldsymbol{n}'(t;c), \boldsymbol{n}_0). \tag{14}$$

In the present work we have used two different pretty equivalent mean function $\overline{\mathcal{F}}$ based on the occupancy frequency :

$$\Theta_{int} = \sum_{j=1}^{J} \frac{\left(h_{o,j} - \frac{1}{T}\sum_{t=t_{min}}^{t_{min}+T} h_{t,j}\right)^2}{\frac{1}{T}\sum_{t=t_{min}}^{t_{min}+T} h_{t,j}} \tag{15}$$

$$\Theta_{ext} = \frac{1}{T}\sum_{t=t_{min}}^{t_{min}+T}\sum_{j=1}^{J} \frac{(h_{o,j} - h_{t,j})^2}{h_{t,j}} , \tag{16}$$

where the first equation is a distance between the observed occupancy and the mean expected one, and the second equation is the mean distances between the observed occupancy and the occupancy calculated at time t. We have use the eq.(15) as the favorite measure because of a better stability on the estimation of the expected occupancy.

Beside the χ^2 function as distance measure we have also used as a double check other kinds of objective functions, that in the case of the use of an inner mean distance are:

$$\Theta = \begin{cases} \chi^2 = \sum_{j=1}^{J} \frac{(h_{o,j}-h_{e,j})^2}{h_{e,j}} , & \chi^2 \text{ distance} , \\ d_H = \frac{1}{\sqrt{2}}\left(\sum_{j=1}^{J}(\sqrt{h_{o,j}} - \sqrt{h_{e,j}})\right)^{1/2} , & \text{Hellinger distance} , \\ d_{KL} = \sum_{j=i}^{J} h_{o,j}\log\frac{h_{o,j}}{h_{e,j}} , & \text{Kullback-Leibler distance} . \end{cases} \tag{17}$$

We have mainly used the first two distances, in particular, we treat the Hellinger distance as a double check for the χ^2 distance and its behavior close to singularities and

By definition, the Hellinger distance is a metric satisfying triangle inequality. The $\sqrt{2}$ in the definition is for ensuring that $h_H(t) \leq 1$ for all probability distributions.

In our optimization procedure we have chosen to use

$$\overline{\mathcal{F}} = \chi^2 = \sum_{j=1}^{J} \frac{(h_{o,j} - h_{e,j})^2}{h_{e,j}} \tag{18}$$

$$h_{e,j} = \frac{1}{T}\sum_{t=t_{min}}^{t_{min}+T} h_{t,j} , \tag{19}$$

where $h_{o,j}$ is the frequency of class j for the observed configuration $\boldsymbol{n}_o$, $h_{t,j}$ is its simulated counterpart at time step t, while $h_{e,j}$ is the frequency class for the mean expected configuration. J is the total number of classes. According to equation (10), the resulting objective function is then $\Theta = \overline{\chi}^2$. And as a distance the χ^2 one.

4.1.2 Discussion on the Empirical Occupancy Frequency

The occupancy frequency $f(n)$ counts the number of alternatives that have to be chosen by exactly n agents:

$$f(n) = \sum_{l=1}^{L} \delta_{n_l,n} , \tag{20}$$

where $\delta_{n_l,n}$ is the Kronecker delta. In general one observes an occupancy distribution with an high modal value close to zero and long tails in the case where the majority of agents choses the same few alternatives while the others remain basically unchosen. A flat occupancy distribution instead, is associated with the case where the choices of agents are evenly distributed in all the alternatives. An intermediate case is represented by a bell-shaped distribution of the occupancies.

This is the basis why we have chosen to select the classes on the basis of the observed data and defining the class with ranges following (11). The more the empirical occupancy classes frequency looks like a flat distribution, the more the simulated configurations are close to the observed configuration vector.

The theoretical prediction for the occupancy class frequency as defined in (11) is

$$f(C_j) = \sum_{n \in C_j} \sum_{l=1}^{L} \pi(n; N, L, a_l, A, b) \ , \tag{21}$$

where π is the marginal distribution of the number of agents choosing an alternative l derived from the theoretical model as in Bottazzi and Secchi (2007). Figure 4 shows the case of the classes definition in terms of the observed data and the respective theoretical occupancy frequency from equation (21).

In order to make an analytical discussion and comparison with the linear case (pure Polya distribution) on paper Bottazzi et al. (2008). It is possible to calculate the binning sizes in such a way the Polya distribution has a flat shape. In this case, the cumulative distribution is computed as

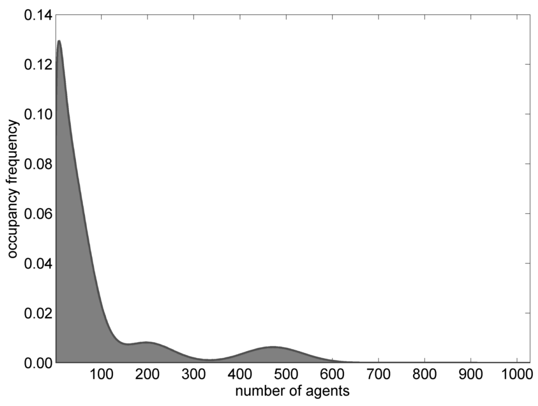
$$F_\pi(n) = \sum_{n=0}^{N} f(n). \tag{22}$$

As consequence, it is straightforward to define the classes $C_1, \cdots, C_j, \ldots, C_J$ as in (11). This choice of class definition makes the bins independent from the observed configuration $\boldsymbol{n}_o$. In this way it is also possible to determine the expected value of the χ^2 distance and its variance that is the amount of fluctuations we expect to observe in the case of $c = 0$. And it can be used as stopping criteria in the iterative optimization method.

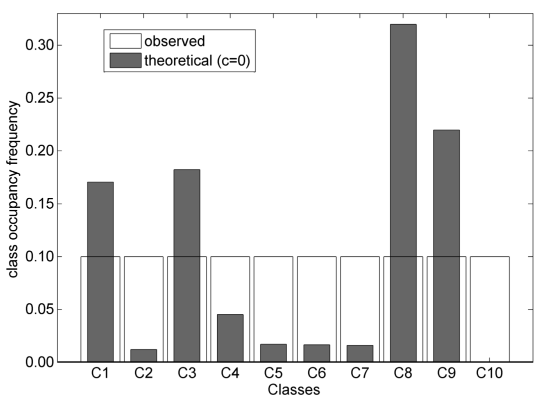
At this point it is useful to study what it is expected to obtain when the objective function between occupancies is at play. To give an analytical estimation, let us now explore the χ^2 distance in terms of occupancy between the ensemble of configurations generated from a Polya distribution h_j and the theoretical occupancy $h_j^{(th)}$. If we take the binning classes procedure starting from the theoretical Polya distribution, we have that

$$h_j^{(th)} = \frac{1}{J},$$

and $\sum_{j=1}^{J} h_j^{(th)} = 1$. We can choose two types of χ^2 distance according to the fact that we take the average of the entire objective function or the objective function of the average occupancies.



(a) The theoretical occupancy frequency for the model with c=0, the Polya case of our null-hypothesis.



(b) The same occupancy frequencies distributed over classes calculated from the the observed data.

Figure 4: (a) Theoretical prediction for the occupancy distribution that counts the number of alternatives chosen by exactly n agents. (b) White bars represents the occupancy class frequency computed on observed data, gray bars represents the occupancy class frequency from the theoretical estimation with the model with $c = 0$, the classical Polya case. It can be noticed how the observed occupancy is flat, that is properly how the bins have been selected.

Specifically,

$$\chi^2_{\text{ext}} := \Big\langle \sum_{j=1}^{J} \frac{\left(h_j - h_j^{(th)}\right)^2}{h_j^{(th)}} \Big\rangle \, , \tag{23}$$

$$\chi^2_{\text{int}} := \sum_{j=1}^{J} \frac{\left(\langle h_j \rangle - h_j^{(th)}\right)^2}{h_j^{(th)}}, \tag{24}$$

where $\langle \cdot \rangle$ is the average operation, and where $h_j^{(th)}$ is a constant value. After some simple calculation we can write

$$\begin{aligned} \chi^2_{\text{ext}} &= \Big\langle \sum_{j=1}^{J} \frac{\left(h_j - \frac{1}{J}\right)^2}{\frac{1}{J}} \Big\rangle, \\ &= 1 + J \sum_{j=1}^{J} \langle h_j^2 \rangle - 2 \sum_{j=1}^{J} \langle h_j \rangle, \end{aligned} \tag{25}$$

$$\begin{aligned} \chi^2_{\text{int}} &= \sum_{j=1}^{J} \frac{\left(\langle h_j \rangle - \frac{1}{J}\right)^2}{\frac{1}{J}}, \\ &= 1 + J \sum_{j=1}^{J} \langle h_j \rangle^2 - 2 \sum_{j=1}^{J} \langle h_j \rangle. \end{aligned} \tag{26}$$

So, the difference between the two ways of calculating the objective function consists of a factor

$$\chi^2_{\text{ext}} - \chi^2_{\text{in}} = J \sum_{j=1}^{J} \left(\langle h_j^2 \rangle - \langle h_j \rangle^2 \right) \ ,$$
$$= J \sum_{j=1}^{J} \text{Var}(h_j), \tag{27}$$

which has been verified numerically with Monte Carlo simulations using the random generator.

If occupancies are not normalized we have:

$$\chi^2_{\text{ext}} = L + \frac{J}{L} \sum_{j=1}^{J} \langle h_j^2 \rangle - 2 \sum_{j=1}^{J} \langle h_j \rangle \ , \tag{28}$$

$$\chi^2_{\text{int}} = L + \frac{J}{L} \sum_{j=1}^{J} \langle h_j \rangle^2 - 2 \sum_{j=1}^{J} \langle h_j \rangle \ . \tag{29}$$

So differently from what has been done in our simulations, if one build the binning procedure from a Polya configuration instead of the observed one, it is possible to have an alanytical evaluation.

4.1.3 Random generator from Polya distribution

The only case in which the model can be analytically described is for the linear case (c=0), where it is possible to write the equilibrium Polya distribution :

$$\pi(\boldsymbol{n}; \boldsymbol{\alpha}) = \frac{N!}{n_1! \cdots n_l!} \frac{\Gamma(\sum_{l=1}^{L} \alpha_l)}{\Gamma(\sum_{l=1}^{L} \alpha_l + n_l)} \prod_{l=1}^{L} \frac{\Gamma(\alpha_l + n_l)}{\Gamma(\alpha_k)} \tag{30}$$

where $\alpha_l = a_l/b$ and $A/b = \sum_{l=1}^{L} \alpha_l$. This equation is the compound probability mass function for the Dirichlet-multinomial (Ng et al., 2011). Consequently, the random number generation can be implemented through the composition technique exploiting the the Multinomial and Gamma random generator in Matlab Statistics toolbox or the corresponding toolbox on Octave (see Gentle, 2004). For the procedure see Code 2. Starting from the gamma random generator, one can then recover the Dirichlet random generator as proved in Devroye (1986). For the purpose of our paper, it can improve computational performance by creating the seeds for our statistics under the null-hypothesis, thus allowing to avoid the time evolution generation for $c = 0$.

4.2 Linear approximation limits

At this point we want to address more precisely the issue of finding an interval of values for c on the optimization problem. Let us start considering the original purpose of the work that is to detect the presence of non-linear perturbation.

In a vectorial form, eq.the numerator of (69) can be written as:

$$\boldsymbol{g}(\boldsymbol{n}) = \boldsymbol{a} + b\boldsymbol{n} + c\boldsymbol{n}^2 \tag{31}$$

The best linear approximation of $\boldsymbol{g}$ near the configuration point $\boldsymbol{m}$ that is the typical configuration of the case $c = 0$ where it is possible derive analytically the expected number of agents that chose an alternative

$$\boldsymbol{m} = \frac{N}{A}\boldsymbol{a}. \tag{32}$$

The general setup is

$$\boldsymbol{g} : \mathbb{R}^L \mapsto \mathbb{R}^L, \qquad \boldsymbol{n} \mapsto \boldsymbol{g} = \boldsymbol{g}(\boldsymbol{n}) \tag{33}$$

The linear approximation condition we are interested in can be estimated through the linear map described by the Jacobian matrix $J_g(\boldsymbol{m})$ near the point $\boldsymbol{m}$. Namely,

$$\boldsymbol{g}(\boldsymbol{n}) = \boldsymbol{g}(\boldsymbol{m}) + J_g(\boldsymbol{m})(\boldsymbol{n} - \boldsymbol{m}) + o(||\boldsymbol{n} - \boldsymbol{m}||) \;, \tag{34}$$

where $\boldsymbol{n}$ is close to $\boldsymbol{m}$, o is the little o-notation, and $\|\mathbf{n} - \mathbf{m}\|$ is the distance between $\boldsymbol{n}$ and $\boldsymbol{m}$.

The matrix J_g is the Jacobian of $\boldsymbol{g}$ at $\boldsymbol{m}$ and is given by

$$J_g = \left[\frac{\partial g_i}{\partial n_k}\right]_{1\leq i\leq L\,,\,1\leq k\leq L} \;. \tag{35}$$

As a first approximation, we can neglect the constraint $\sum n_i = N$ and consider the Jacobian to be a diagonal matrix, so as to have the same expression for each element of the vector $\boldsymbol{g}$. That is,

$$g_k(n_1, \ldots, n_L) = g_k(m_1, \ldots, m_L) + \left.\frac{\partial g_k}{\partial n_k}\right|_{m_k} (n_k - m_k) + \ldots, \tag{36}$$

which yields the first order approximation

$$\tilde{g}_k = a_k - cm_k^2 \; + \; (b + 2cm_k)n_k \qquad \text{for } k = 1, \ldots, L \tag{37}$$

$$= \tilde{a}_k + \tilde{b}n_k \;. \tag{38}$$

To fulfill all the properties for the utility function g, we can impose

$$\tilde{a}_k = a_k - cm_k^2 \geq 0 \;, \tag{39}$$

$$\tilde{b} = b + 2cm_k \geq 0 \;. \tag{40}$$

Conditions (39) and (40) hold true for every k. Hence, the intervals for the linear approximation are

$$I_k = \left[-\frac{b}{2m_k}, \frac{a_k}{m_k^2}\right] \;, \qquad \forall k = 1, \ldots, L \;. \tag{41}$$

the intersection of those intervals, $I_{lin} = \cap I_k$, brings to the condition for linearity

$$I_{lin} = \left[-\frac{A}{2N}\frac{b}{\max(a_k)} \, , \, \frac{A^2}{N^2}\frac{1}{\max(a_k)} \right] \, , \tag{42}$$

where we have used equation (32) for the average configuration vector, and $A = \sum_{k=1}^{L} a_k$ and $N = \sum_{k=1}^{L} n_k$.

It is important to stress that if the function g is an utility function we have to consider the constraint for which $g > 0$. Hence, the search interval is the intersection of the I_{lin} and the previous condition.

4.3 Bias on distance measures

The optimization technique discussed in the paper rests on equation (10), one can be lead to think that the minimum distance is obtained with evolution with $c = 0$ that is “more similar” to the starting configuration.

Actually, small negative variations of the linear parameter b gives smaller fluctuations of the occupancy distributions, resulting in a smaller distance measure relative to the same process with $c = 0$. In fact, in the first order perturbation regime we can approximately write

$$\tilde{g}_l = a_k + (b - \Delta b) n_k \, , \tag{43}$$

since we want $\tilde{a}_k \approx a_k$ so that $\sum \tilde{a}_k \approx A$. From the theoretical distribution of the linear case $c = 0$ we can write the variance of the configuration (see Garibaldi and Scalas, 2010):

$$E[n_l] = N\frac{a_l}{A} \tag{44}$$

$$\mathrm{Var}(n_l) = N\frac{a_l}{A}\frac{A - a_l}{A}\frac{A + bN}{A + b} \tag{45}$$

$$\mathrm{Cov}(n_l, n_m) = -\left(\frac{N + A/b}{1 + A/b}\right)\frac{E[n_l]\,E[n_m]}{N}. \tag{46}$$

It is evident how covariance and variance diminish when A grows, that is when the number of alternatives gets larger.

In the high dimensional case we have $A \gg a_l$ and $A \gg b$, so we can write the variance

$$\mathrm{Var}(n_l) \approx a_l \frac{N}{A^2}(A + bN). \tag{47}$$

From equation (47) it is clear that small negative perturbations reducing the value of b result in a smaller variation of the occupancy frequency inside each bin, while the mean value of the frequency is unaffected. This impacts on the occupancy distributions that has the same shape as in the case $c = 0$ but attenuated fluctuations, thus resulting in smaller distances to the starting occupancy distribution of $\boldsymbol{n}_0$. However, in real simulations, the value of this bias on is usually smaller than the order of magnitude of the stopping criteria.

4.3.1 Baseline discussion

Let us see in detail the previous discussion in a low-dimensional case with a subsample of the real data reported in table 1

Table 1: Example data

$\boldsymbol{n}_o$	58	85	284	17	40	15	280	118	52	139
$\boldsymbol{a}$	6.29	5.45	14.25	1.38	1.27	1.38	21.14	12.76	4.77	6.04

Note: Data used in the example in this appendix. The total number of agents is $N = 1088$ over $L = 10$ alternatives with $\boldsymbol{a}$ and the observed $\boldsymbol{n}_o$ as in the table.

The following examples are serve to explain in a clear way how the computational procedure is supposed to work. The inferential analysis is not considered to be reliable for real data.

Table 2: Baseline results

Parameters	Example
c_{min}	$-0.97 \cdot 10^{-3}$
I_{lin}	$[-1.06 \cdot 10^{-3}\, ,\, 1.46 \cdot 10^{-4}]$
T_{eq}	$\sim 10^4$

In the example, the time for equilibrium results to be $t_{min} = T_{eq} = 1.6 \cdot 10^4$ as also recovered from the theoretical prediction

$$T_{eq} = \frac{N(A + N - 1)}{A/b} \tag{48}$$

. The simulation for a given initial seed for the state $\boldsymbol{n}_0$ is given in Figure 6, and the values in the computations of the averages are given as an example of the method's ability to discriminate.

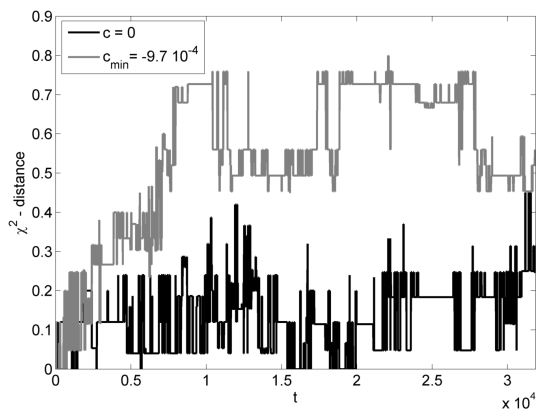
Another important initial parameters to estimate is the stopping criteria, which coincides with the fluctuations measure of the distance. It is evaluated via a Monte Carlo simulations of many trajectories that start from different seeds from a configuration drawn from a Polya distribution and evolved along time with $c = 0$.

4.4 Optimization methods

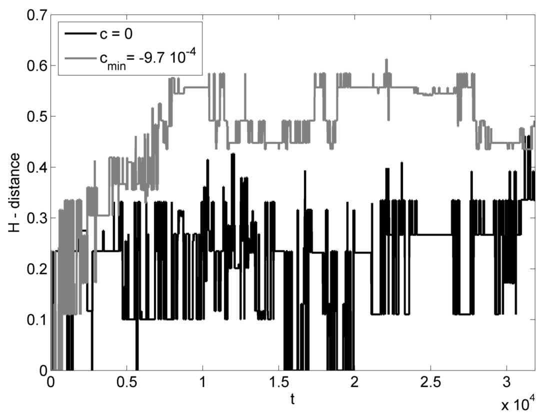
Problem (9) can be solved numerically searching for the argument that minimizes χ^2, either via the classic grid method or via successive parabolic interpolation. It is worth to describe both approaches in some detail to motivate why one is preferable relative to the other in the applications that can be of interest for the model presented here.

4.4.1 Grid

With the classic grid method, problem (9) is tackled as illustrated in Figure 1 and already described in Section 4. In particular, the search for $\hat{c}$ is limited within the interval $[c_{min}, c_{max}]$,

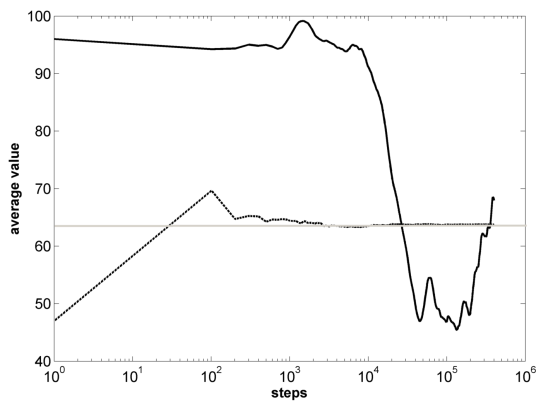


(a) χ^2 distance. The value of the mean distance are $\overline{\chi^2}(c = 0) = 0.14$ and $\overline{\chi^2}(c = c_{min}) = 0.63$.



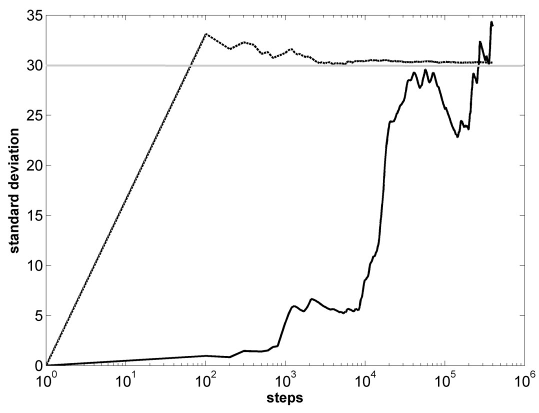
(b) Hellinger distance. The value of the mean distance are $\overline{H}(c = 0) = 0.21$ and $\overline{H}(c = c_{min}) = 0.51$.

Figure 5: Distance in terms of *(a)* the χ^2 and *(b)* Hellinger function have been calculated as in equation (14)), for two values of c, where c_{min} is the maximum negative value allow. The gray line represents the distance measures of the trajectories with $c = c_{min}$ and the black lines the trajectory with $c = 0$. It can be noticed that after a transient of t_{min} the two series tend to their average. The time series with $c = 0$ tends to be the closest in terms of occupancy distribution to the starting configuration $\boldsymbol{n}_0$



(a) Mean value of the number of agents in one location. The fluctuations of the temporal mean (black solid line) are of the order of the expected standard deviation after a transient that is of the order of $T_{eq} = 1/r \approx 1.6 \cdot 10^4$ as estimated in eq.(48)

.



(b) Value of the standard deviation of the number of agents choosing a certain alternative. Again we have that starting from the time T_{eq} the estimated values statistically become equals.

Figure 6: Computational steps to reach equilibrium. Three evaluations have been estimated for (a) the mean value of agents in a given location, and (b) the standard deviation of the number of agents in the same location. The gray straight line is the analytical theoretical prediction (expected value), the dotted black line represents a sample drawn from the Polya random generator (ensemble average via Montecarlo) and the steps are the number of times we have drawn from the Polya generator. The black solid line represents the time evolution of the Polya process (temporal mean via Montecarlo). It can be noticed that more the window for the average calculation becomes big, more the average values tends to the real expected values. It is important underline how the random generator produces faster convergence then the time evolution process, about two orders of magnitude.

where c_{min} is defined by equation (6) and c_{max} is set according to the specific process under scrutiny. Specifically, if one is interested on small deviations from linearity, the value of c_{max}

has to be in the order of $1/\tilde{n}$, where $\tilde{n}$ is some "typical" occupancy. To be safe, here we set $c_{max} = \max(1/n_1, \dots, 1/n_L)$ where the time average is computed for a trial simulation, i.g the typical configuration vector derived from the process with $c = 0$.

Having defined the boundaries $[c_{min}, c_{max}]$, the other choice to be made concerns the step size to be adopted in the search. Intuitively, a finer step size tends to guarantee a higher definition in the measure of the parameter estimates. Nonetheless, the process under scrutiny is stochastic, thus making the value of χ^2 noisy. As a consequence, an excessively fine step size would be useless, since it would not manage to discern values beyond a certain resolution. Figure 7 gives an example in this sense, where the relatively fine step size magnifies the oscillations deriving from the random nature of the process.

In addition, the time of search R is proportional to the number of points in the interval. Therefore, when generating a distribution of the statistics of c_s^* with S samples, the cost of the computation is proportional to $S \cdot R \sim S \cdot O(T)$, where T is the length of the temporal evolution of the stochastic process. Clearly, this implies that the step size cannot be too fine if time-efficiency is to be guaranteed for practical purposes.

On the other hand, a coarser step size entails a loss in definition as well as greater fluctuations around the "true" value of the parameter estimate. Such fluctuations increase the variance of the estimate, and thus the probability not to reject the null hypothesis. Overall, then, the grid method entails a choice in the step size which cannot be easily controlled.

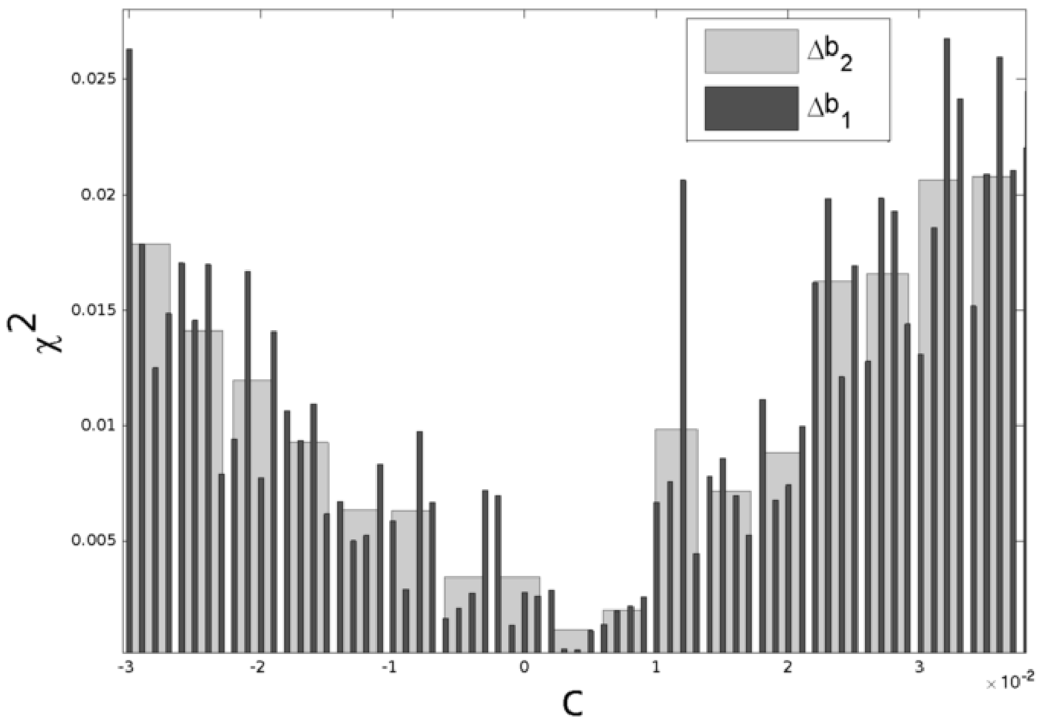
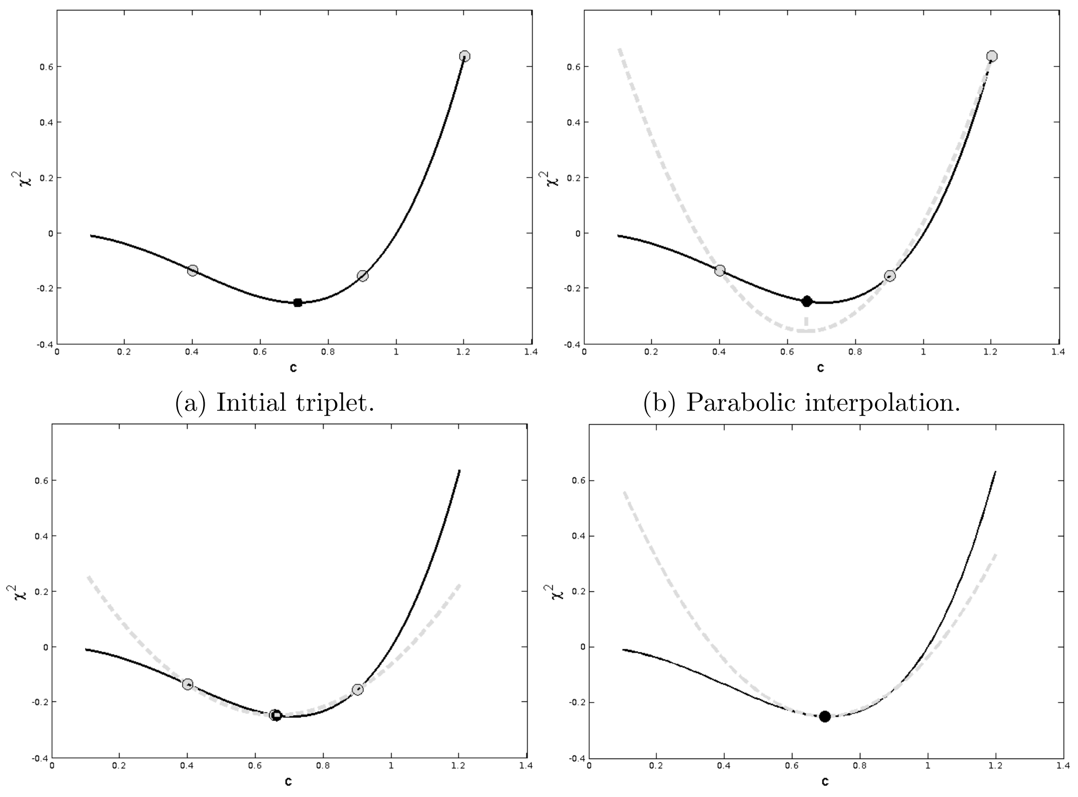


Figure 7: Grid search method to detect the minimum of the χ^2 function. Two dimension of binning size are given. The difference is that decreasing the size allows a finer approximation of the minimum at the cost of a less smooth curve affecting the performance of the optimization procedure.



(a) Initial triplet.

(b) Parabolic interpolation.

(c) New triplet and interpolation.

(d) Convergence.

Figure 8: Successive parabolic interpolation.

4.4.2 Successive parabolic interpolation

An alternative optimization method exploits the quadratic behavior of the distance function χ^2 around its minimum by relying on the method of successive parabolic interpolations. This approach allows to substantially escape the choice of c_{max} required by the grid method, while also guaranteeing a faster convergence. For these reasons, successive parabolic interpolation is generally more convenient than the grid method.

To give a visual intuition, Figure 8 illustrates the method of successive parabolic interpolation in its four salient moments. Suppose that the objective $\overline{\chi}^2$ follows the shape represented by the solid line in Figure 8. Then, the aim is to find the minimum of this function, as represented by the black dot on the solid line in Figures 8a. The function is not required to be differentiable. The search algorithm starts by drawing three points lying on the function to be optimized, as represented by the gray dots in Figure 8a. This initial triplet is then interpolated by a parabola (the dashed line in Figure 8b). Since the functional shape of the interpolated parabola is known, its minimum can be determined analytically. Hence, the abscissa associated to the analytical minimum is used to identify a new point on the objective function (the black dot in Figure 8b). This new point is now taken on board, while dropping the old point that is horizontally furthest from it. By doing so, a new triplet is formed, which leads to a new parabolic interpolation and to the identification of a new analytical minimum (see Figure 8c). This process continues iteratively until it converges to a stable point, which means that the previous and the successive minimum correspond within a certain range of tolerance. The point of convergence is the approximate solution to problem (9).

The fundamental property supporting this method is that a unimodal function can be approximated by a parabola over an interval that includes the minimum. More precisely, having a triplet of points x_0, x_1, x_2 and their function values $f(x_0), f(x_1), f(x_2)$, the second-order Lagrange interpolation is used to construct a second-degree polynomial which approximates the parabolic function:

$$q(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} f(x_0) + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} f(x_1) + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} f(x_2) \ , \tag{49}$$

Clearly, the critical point of equation (49) is given by the first order condition $dq(x)/dx = 0$. For practical purposes, however, equation (49) can also be seen as a quadratic polynomial

$$q(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) \ , \tag{50}$$

where a_0, a_1, a_2 have to be such that equation (50) agrees with $f(x)$ at the three points. That is $f_0 \equiv f(x_0) = q(x_0)$, $f_1 \equiv f(x_1) = q(x_1)$ and $f_2 \equiv f(x_2) = q(x_2)$. It follows that

$$a_0 = f_0 \ , \quad a_1 = \frac{f_1 - f_0}{x_1 - x_0} \ , \quad a_2 = \frac{1}{x_2 - x_1}\left(\frac{f_2 - f_0}{x_2 - x_0} - a_1\right) \ , \tag{51}$$

thus making equation (50) equivalent to equation (49). In this alternative form, the critical point to be sought by the algorithm is

$$\bar{x} = \frac{x_1 + x_0}{2} - \frac{a_1}{2a_2} \ . \tag{52}$$

Expressions (50)-(52) are used in the implementation of the algorithm. In doing so, one needs to consider at least two aspects. First, the sign of a_2 determines the concavity of our parabola, thus making the critical point a candidate for a minimum ($a_2 > 0$) or for a maximum ($a_2 < 0$). Second, the three points involved in equation (52) lie on a line if $a_2 = 0$, thus imposing to change the step size between the triplet.

Given this parabolic interpolation, another important detail of the algorithm concerns the stopping criterion. Having to deal with a stochastic process, the objective function $\overline{\chi}^2$ is noisy, and this introduces the necessity to fix a tolerance criterion based on the variance of the process. To this purpose, the standard deviation of the χ^2 function is calculated computationally. Consequently, the algorithm is set to stop when the fluctuations of our minimum values are of the same order of magnitude of the estimated standard deviation, that is

$$|f(\bar{x}) - f(x_{min})| \leq \sigma(f_{\chi^2}) \tag{53}$$

Finally, convergence is generally fast. As discussed in Brent (1973), the convergence rate for a deterministic function is superlinear (i.e. $r \approx 1.3$). In the stochastic case, however, it can happen that the interpolation parabola has a local convex behavior for a given triplet, as shown in Figure 9. This is due to the random nature of the objective function. In this case, quadratic interpolation is repeated with the same triplet but recalculating the stochastic function value of the central point of the triplet. Such recalculation is repeated until that point is associated

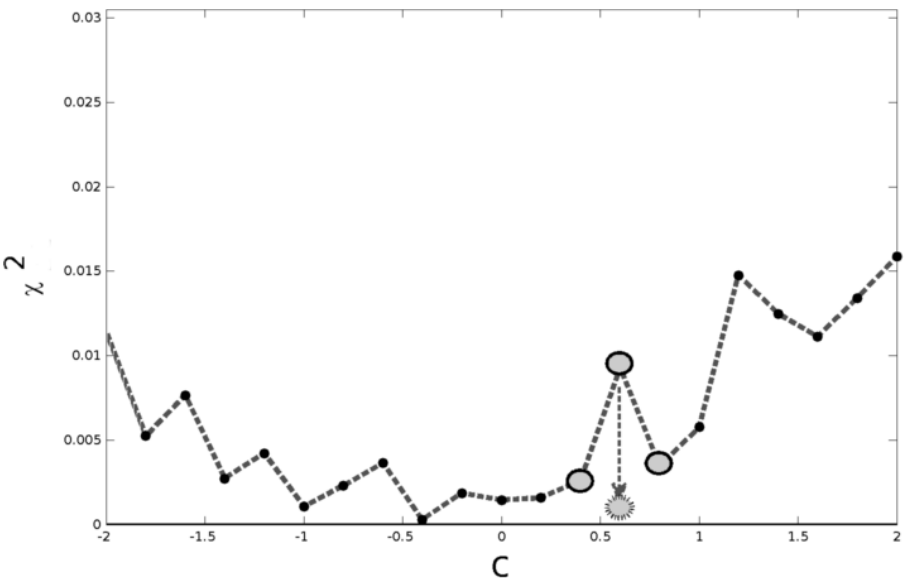


Figure 9: Local convexity problem.

to a function value that is under the straight line passing through the two external points of the triplet.

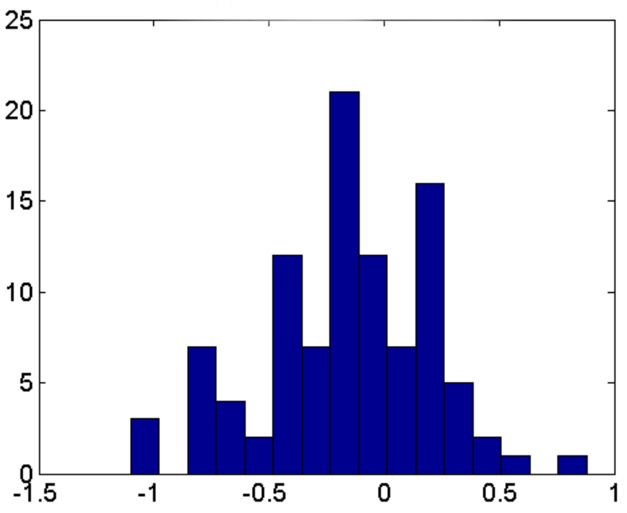
5 Computational costs of the optimization methods

The performance of the two optimization method can be evaluated according to their speed of convergence. As mentioned above, successive parabolic method can have convergence limitations related to far solutions, point alignment, and convexity. Each of these issues needs *ad hoc* control conditions, which also entail some cost in terms of efficiency. Nonetheless, successive parabolic estimation is much faster than the grid method. In particular, we performed a test for a given vector of $\boldsymbol{a} = (a_1, \dots, a_L)$, an elevated number of alternatives $L = 1 \cdot 10^3$, and a conspicuous number of agents $N = 1 \cdot 10^4$. With these parameter values, the grid method manages to construct the histogram of c^* in Figure 10a in 120 minutes. On the other hand, successive parabolic interpolation allowed to construct the corresponding histograms in Figure 10b in 20 minutes, reaching convergence in 8–10 steps on average. Due to this remarkable difference in performance, successive parabolic interpolation is adopted as the default optimization method to carry out estimation.

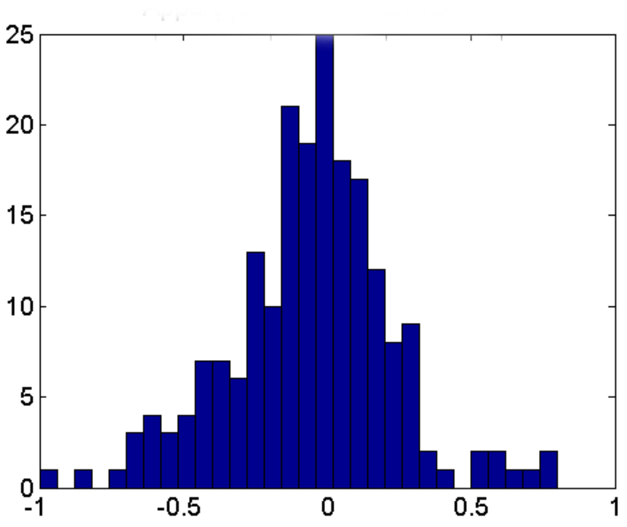
More generally, the algorithm of the stochastic process has asymptotic computational complexity $O(T)$, where T is the length of evolution of the process, which thus generates T configurations. Hence, if the first $t = 10^4$ steps take 0.7 seconds to be simulated, the computation of $t = 10^5$ steps is achieved in 7 seconds.

This section discusses the computational cost associated to the codes proposed in the paper, focusing in particular on the code for the evolution of the choice process described in Section 2. In general, the efficiency of all codes is affected by time and memory costs. The first is the time required to execute the algorithm, while the second amounts to the quantity of memory required by the algorithm.

As for the time cost, we make use of asymptotic analysis to estimate the computational



(a) Grid method.



(b) Successive parabolic interpolation.

Figure 10: Histogram of $\tilde{c}$.

Note: The parameters of this example are $L = 1 \cdot 10^3$ and $N = 1 \cdot 10^4$.

time complexity O of the algorithm. The basic algorithm to take as a reference is the code for the evolution of the choice process described in Section 2. It has an asymptotic cost of $C(L, T) \sim O(T) \cdot O(L)$, where T is the number of time steps for which the simulation runs and L is the number of alternatives in the model. Hence,

$$C(L, T) = (a_0 + a_1 L) \cdot T \ , \tag{54}$$

$$= (a_0 T) + (a_1 T) L \ , \tag{55}$$

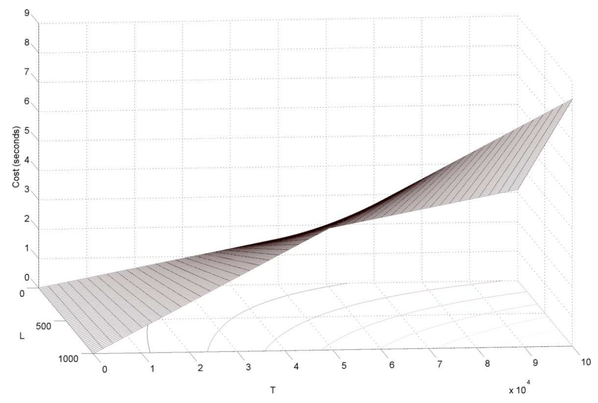
which represents the time cost of the algorithm with the parameters having the real time dimension $[a_0] = [a_1] = [\text{sec}]$. Using linear fit on multiple simulations of the code for fixed T, it is possible to evaluate the parameters (a_0, a_1). These values depend on the machine in use, so that one has to obtain the performance of the program with a preliminary benchmark test for a specific computer architecture, and then compare it with the performance on other machines. On the other hand, the memory cost for the choice process described in Section 2 is

$$M(L, T) = mL \cdot T \ , \tag{56}$$

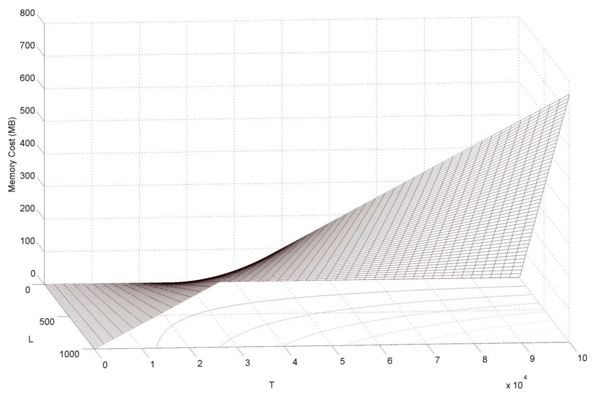
where the dimension of the parameter is $[m] = [\text{MBytes}]$. As an example, we have recovered the computation of (a_0, a_1, m) for two different machines using Matlab as programming language.

The first machine specification (considered here as the reference machine in the paper) mounts an Intel Core i7-3612QM Processor (6M Cache, up to 3.10 GHz) rPGA (with 4 cores and 8 threads), a DDR3 SDRAM of 8 GBytes and an Intel HD Graphics 4000. The operative system is Windows 7 Professional 64-bit and Linux Mint 15 64bit. With this machine architecture, resulting estimates of the time and memory cost parameters are $a_0 \approx 3.1 \cdot 10^{-5}\text{sec}$, $a_1 \approx 5.2 \cdot 10^{-8}\text{sec}$ and $m \approx 7.6 \cdot 10^{-6}\text{MBytes}$.

The second machine (taken as a comparison machine) uses an Intel Core i3-3220 CPU



(a) Time cost for the choice process algorithm.



(b) Memory cost for the choice process algorithm

Figure 11: Computational costs.

@ 3.30GHz (with 2 cores and 4 threads), a DDR3 SDRAM of 4GBytes, and an Intel HD Graphics 2500. The operative system is Windows 7 Basic 64-bitand Ubuntu-Linux 11 64bit. In this case the parameter estimates result to be $a_0 \approx 3.06 \cdot 10^{-5}$sec, $a_1 \approx 4.9 \cdot 10^{-8}$sec and $m \approx 7.6 \cdot 10^{-6}$MBytes. Hence, the two machines give similar results.

To better understand the behavior of time and memory costs, Figure 11 represents the functions $C(L,T)$ and $M(L,T)$. For example In the empirical application presented in Giulio Bottazzi and Vanni (2014), the number of alternatives is given by the data as $L \sim 7 \cdot 10^2$. Given this size of L and setting $T \sim 10^5$, the typical time of execution is $C \approx 6.7$sec, and the required memory 530MB of physical space. Therefore, we can determine the waiting time to get the output vector, but we can also put a limit on T according to the physical capability of the machine in use. On 32-bit machines the memory is limited to 2^{32} bytes (i.e. 4GB). On a 64-bit machine this number is virtually increased to 2^{64}, but most machines run out of physical resources (RAM and virtual memory). For instance, in the comparison machine described above we cannot go beyond 4GB even if on a 64-bit system.

The output matrix records the evolution of configuration vectors over time, thus having size $(L \times T)$. Each number in the matrix cell occupies 8-bytes, due to double-precision floating-point format. Consequently the physical memory occupied by such matrix is $L \times T \cdot 8$bytes. In the reference machine, the maximum available physical space is from 6 to 8GB; therefore, we can run the choice process for a limit time of $T = 10^6$ since $L = 686$, i.e. $10^6 \cdot 686 \cdot 8$ bytes $= 5.1$GB.

Finally we want to offer an alternative to test the codes on a software under free license. A popular Matlab-like environment is the high-level interpreted language GNU Octave. We have used the version 3.8.0 of Octave, and the estimation of the parameter on computer architecture are $a_0 \approx 2.7 \cdot 10^{-4}$ and $a_1 \approx 1.4 \cdot 10^{-7}$, with an execution cost of $C = 36.9$sec for $L \sim 7 \cdot 10^2$ and $T \sim 10^5$. This results to be a much slower performance relative to the Matlab execution time.

The efficiency of the algorithm depend on many factors (programming language, CPU, type of data); besides that, its efficiency depends also on the RAM of the calculator and the way data is stored. Appendix B details the efficiency behavior of the codes presented in this work.

6 Conclusion

This paper has spelled out a discrete choice model accounting for different determinants of individual choices. In particular, the model disentangles the effect of externalities from other factors such as the intrinsic characteristics of the object of choice and the idiosyncratic preferences held by agents. Relative to previous works in the same line of inquiry, the present one has taken a step forward in allowing for non-linear externalities among agents.

Together with the theoretical model, also an entire estimation framework has been developed. The model's parameters were estimated numerically through a multistage procedure, while also testing for statistical significance via Monte Carlo simulations. This involved different numerical optimization techniques, at last finding a preferable approach in successive parabolic interpolation.

A detailed information about technical issues of the estimation have been addressed in such a way to have a full command of the entire process of optimization.

A Maximum likelihood

This section discusses the numerical implementation of various Maximum Likelihood (ML) approaches that we have used to carry out the inferential analysis described in Section 4. The main purpose is to show why we have turned to a χ^2-based estimation as an alternative to ML. This choice entails moving from a point (or local) estimation approach, such as ML, to an aggregate approach. More precisely, ML compares single configuration vectors (or their local neighbors), while χ^2-based estimation compares occupancy distributions, in this sense representing an aggregate approach.

The general estimator can be defined with an optimization problem of the form

$$\hat{\boldsymbol{\theta}} = \underset{\theta \in P}{\operatorname{argmin}} \{\Theta(\boldsymbol{\theta})\} \ , \tag{57}$$

where the parameter vector $\boldsymbol{\theta} \in P \subset \mathbb{R}^m$ and the objective function $\Theta(\boldsymbol{\theta}) : P \to \mathbb{R}$. For example, in the χ^2 optimization described in Section 4.1, we have $\boldsymbol{\theta} = c$, $m = 1$, and $\Theta(\boldsymbol{\theta}) = \bar{\mathcal{F}}(\boldsymbol{n}'(c), \boldsymbol{n}_o)$ as a measure of distance between occupancy distributions. Here, instead, we consider the maximum likelihood approach to the solution of problem (57). In particular, we define and discuss the ML computation of the estimator $\hat{\boldsymbol{\theta}}$ through various numerical methods. We do so considering the case of two unknown parameters $\boldsymbol{\theta} = (b, c)$ and a known vector $\boldsymbol{a}$, so that $m = 2$. Given this setting, we first define the various numerical ML techniques we have adopted in Sections A.1–A.1.3, and then we discuss the problems on which they end up stumbling in Section A.2.

A.1 Grid search method

The most basic way to attack problem (57) in terms of likelihood is the grid search approach. The observed data is given by a point $\boldsymbol{n}_o \in \mathbb{R}^L$ in the configuration space:

$$\boldsymbol{n}_o = \begin{bmatrix} n_1 \\ n_2 \\ \vdots \\ n_L \end{bmatrix}_o \ , \tag{58}$$

where L is the number of alternatives available to the agent. Given $\boldsymbol{n}_o$ and $\boldsymbol{a}$, one has to look for the simulated configuration $\boldsymbol{n}'(\hat{b}, \hat{c})$ which is most likely to have produced the observed data. In particular, numerical ML estimation does so by seeking for the values $(\hat{b}, \hat{c})$ that maximize the likelihood function (or minimize its negative).

Let P be a two-dimensional rectangular grid in the space of parameters $G = [b_{min}, b_{max}] \times [c_{min}, c_{max}]$. Each interval is divided to construct an equally spaced grid on the region G with $g_b \times g_c$ grid points, which are spaced according to the boundaries $[i, i + g_b\epsilon_b) \times [j, j + g_c\epsilon_c)$. Here, ϵ_b and ϵ_c are the grid step sizes given by $g_b = (b_{max} - b_{min})/\epsilon_b$ and $g_c = (c_{max} - c_{min})/\epsilon_c$ and $i, j = 0, 1, \ldots, g$. For the sake of simplicity we take $g_b = g_c = g$.

We evaluate the objective function at each point of the grid, so as to find the point which solves problem (57), which in this setting becomes

$$(\hat{b}, \hat{c}) = \underset{(b,c)\in G}{\operatorname{argmin}} \{\Theta(b, c)\} \ , \tag{59}$$

where the objective function Θ is the likelihood function. The sample configurations are generated by simulating the evolution of the configuration state for each couple of values (b, c) on the grid, so as to have for each point T simulated configurations

$$\{\boldsymbol{n}'_{t=1}, \boldsymbol{n}'_{t=2}, \ldots, \boldsymbol{n}'_{t=T}\}_{(b,c)} \ , \tag{60}$$

where it assumed that the process has already reached the equilibrium.

As a first approach, we use a straight point estimation obtained by counting how many times in a simulation the observed configuration $\boldsymbol{n}_o$ is reached for each point in the grid. The objective function Θ in this case is the likelihood function

$$\mathcal{L}(b, c) = -\Theta(\boldsymbol{\theta}) = \frac{1}{T}\sum_{t=1}^{T} \delta_{\boldsymbol{n'}(t;b,c),\boldsymbol{n}_o} \ , \tag{61}$$

which compares a simulated configuration $\boldsymbol{n}'(t; b, c)$ with the observed configuration $\boldsymbol{n}_o$. The optimum $(\hat{b}, \hat{c})$ that solves problem (59) is the couple of parameter values which, if the underlying model is true, make it most likely to observe $\boldsymbol{n}_o$. Notably, a couple of tricks can be used to improve numerical performance.

A.1.1 Decimation procedure

The first trick aims at reducing time of convergence through a decimation procedure that avoids calculations for those points which are unlikely to solve problem (57). Basically, we put a threshold in the likelihood estimation procedure and split the time T of every single simulation in a fixed number of equal time intervals (ΔT) $\{T_1, T_2, \ldots, T_D = T\}$. For each interval, we neglect all the points which have not reached a given threshold in the ML value. The aim is to keep only those points that are good candidates to be a maximum.

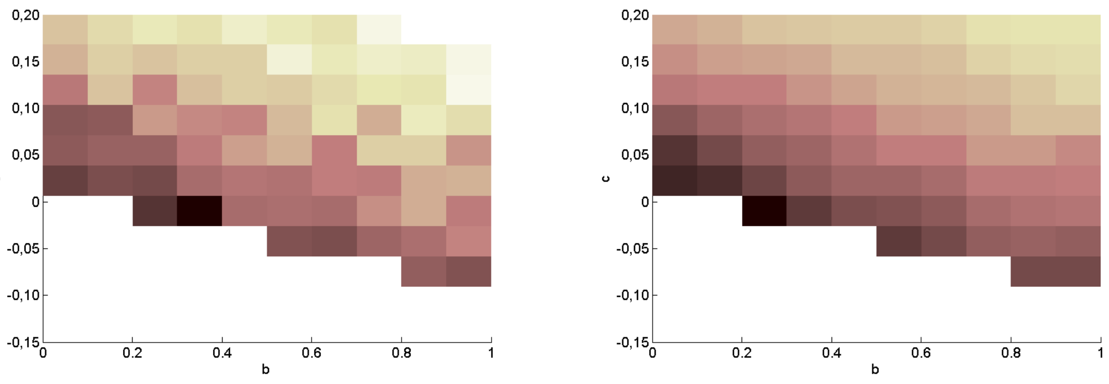
As an example we define the threshold as a cut factor $r < 1$. It means that for each time interval we neglect the r ratio of pixels with lowest likelihood. The total cost of the decimation procedure is:

$$C_{kNN} = C(L, \Delta T) \sum_{k=1}^{D} N r^k = C(L, \Delta T) \cdot N \frac{1 - r^D}{1 - r}; , \tag{62}$$

where $T = D \cdot \Delta T$. On the other hand, the straight point estimation has a cost of $C_{tot} = NC(L, T)$. Using the property of linearity we have $C(L, T) = D \cdot C(L, \Delta T)$. Hence, the computational time cost saved is described by the cost ratio:

$$\frac{C_{kNN}}{C_{tot}} = \frac{1 - r^D}{1 - r} \cdot \frac{1}{D} \ . \tag{63}$$

As an example, splitting the total time in $D = 20$ pieces and with a cut ratio $r = 1/2$ (i.e. we



(a) Straight point estimation, equation (61). (b) Convolution mask.

Figure 12: Example of numerical maximum likelihood estimation.

Note: The example in the figure has the following parameter values: $L = 3$, $a_1 = a_2 = a_3 = 1$, and a grid of 100 points ($g = 10$). The observed configuration is simulated with parameters ($b = 0.5, c = 0$) and $\boldsymbol{n}_o = [1, 5, 3]$. The sample configuration vectors are run for $T = 10^5$ time streps in order to collect a sufficiently large sample to compute statistics. Darker pixels indicate and higher likelihood value. Empty pixels represent the values of c lying outside the domain defined in Section 3.

cut half of the sample at every time step), we have a cost ratio of 1/10.

A.1.2 Convolution smoothing

Another trick consists in stabilizing the counting of nearby pixels by considering neighboring points in the counting procedure of the likelihood estimation. In order to get a better convergence to the estimation of the likelihood we can weight the value of each bin with the values of its neighbors. In practice, we apply a convolution mask on each point (i, j) in two-dimensional space:

$$O(i,j) = \sum_{k=1}^{m} \sum_{l=1}^{n} I(i + k - 1, j + l - 1)K(i,j) \,, \tag{64}$$

where $I(k, l)$ is the ML value in each point of the two-dimensional space, and $K(i, j)$ is the weight of each point of the mask. To carry out this procedure, we adopt a convolution mask with dimension 3×3 in which each element has weight 1. At this purpose we have used the build-in Matlab function conv2.m.

A.1.3 k-nearest neighbors

An alternative nonparametric method to carry out MLE is referred to as the k-nearest neighbors (k-NN, see Silverman, 1998). This method tries to adapt the amount of smoothing to the local density of data, and k indicates the degree of smoothing. The idea is to base estimation on a fixed number of k observations, which are closest to the reference point according to some measure of distance.

In particular, at each time step we calculate the euclidean distance between the simulated configuration vectors $\{\boldsymbol{n}'_1, \ldots, \boldsymbol{n}'_k, \ldots, \boldsymbol{n}'_T\}_{(b,c)}$ and the observed vector $\boldsymbol{n}_o$. Specifically, distance is measured element by element (that is, location by location in the application in

Giulio Bottazzi and Vanni (2014)) as

$$D_k = ||\boldsymbol{n}_o - \boldsymbol{n}_k||. \tag{65}$$

The order statistics for the distances are then D_k are $0 \leq D_{(1)}, D_{(2)}, \leq, \ldots, D_{(T)}$, defining $R_k = D_{(k)}$. Hence, the observations corresponding to these order statistics are the "nearest neighbors" of $\boldsymbol{n}_o$. The first nearest neighbors is the closest observation to $\boldsymbol{n}_o$, the second nearest neighbor is the observation second closest, and so on. The observations ranked by distance from $\boldsymbol{n}_o$ are then $\{\boldsymbol{n}'_{(1)}, \ldots, \boldsymbol{n}'_{(k)}, \ldots, \boldsymbol{n}'_{(T)}\}$. The kth nearest neighbor of $\boldsymbol{n}_o$ is $\boldsymbol{n}'_{(k)}$.

The k-NN likelihood estimate is defined as

$$\mathcal{L} = \frac{k}{T c_L R_k^L}, \tag{66}$$

where $c_L = \pi^{L/2}/\Gamma((L+2)/2)$ is the volume of the L-dimensional unit sphere, and T is the number of samples. The estimator is inversely proportional to the distance R_k. If R_k is small, this means that there are many observations near $\boldsymbol{n}_o$, so L must be large . While if R_k is large this means that there are not many observations near $\boldsymbol{n}_o$, so L must be small. The choice of the smoothing parameter follows Silverman (1998), thus being $k \approx T^{4/(L+4)}$.

The k-NN method is one of many smoothing techniques, among which also kernel density estimation (KDE) is especially popular. In short, KDE is based on a local choice of the bandwidth (the smoothing parameter) and a particular choice of a window (kernel) function. Clearly, both methods have their own advantages and disadvantages; yet, in our particular case, there are precise reasons to pend for k-NN. First, the main advantage of this estimator in our case is that smoothing varies according to the number of observations in a region, thus delivering a smoother estimation with an higher adaptive behavior. This characteristic is crucial in our setting, and it is even more important in multivariate distributions. That is also the case when the bias of the estimator becomes of the same order of the one from the kernel method, limiting one of the disadvantages of the k-NN technique. Another disadvantage is that the estimated function do not integrate to one over the entire domain. In our case this problem is less relevant since we typically only care about the point-wise behavior of the estimated function.

A.2 Issues with numerical maximum likelihood estimation

The computation of numerical ML through the basic search grid method entails a high computational cost, especially for high values of L. That is precisely why decimation, convolution smoothing, and k-nearest neighbors have been used, so as to reduce the computational cost. This attempt has succeeded, as the computational cost was indeed sensibly reduced. Despite these improvements, however, numerical ML estimation remains problematic. In particular, two limitations arise when using ML estimation. The two limitations have to do with ambiguity in counting and with computational cost.

To clarify these points, it is convenient to expose a simple example with few alternatives ($L = 3$) and a small number of agents ($N = 9$). For the sake of simplicity let us also assume that $a_i = a_j \; \forall i, j$. We need to look for the likelihood that a given configuration $\boldsymbol{n_o}$ is detected by counting how many times such configuration is found during a simulation of our model for some values (b, c).

It is straightforward to notice that a simulated configuration $\boldsymbol{n}_t$ is considered to be different from $\boldsymbol{n}_o$ even when it appears in an order probabilistically equivalent to $\boldsymbol{n}_o$ but with a different redistribution. For instance, suppose that $\boldsymbol{n}_o = [3, 4, 2]$, then

$$\boldsymbol{n}_t = \begin{bmatrix} 3 \\ 2 \\ 4 \end{bmatrix} \neq \begin{bmatrix} 3 \\ 4 \\ 2 \end{bmatrix}.$$

In this case, $\boldsymbol{n}_t$ is not counted as being equal to $\boldsymbol{n}_o$. This reduces the ability to discriminate the real likelihood that an observed configuration has been replicated with a particular set of parameters. Therefore, to count a sufficient number of instances in which $\boldsymbol{n}_t = \boldsymbol{n}_o$, one has to obtain an especially large sample.

The second reason why we discarded numerical ML estimation lies in its high computational cost. Although the k-nearest neighbors approach reduces this problem, it cannot avoid intrinsic confusion on what neighborhood is about. Therefore, even when we move from a point ML estimation to a local estimation via the k-nearest neighbors method, the computing of reliable measures of ML is still poor for $L \gg 10$, like in the application in Giulio Bottazzi and Vanni (2014). Of course this is a numerical constraint deriving from the high computational cost.

The combination of ambiguity and computational cost is the reason why me moved from a point (or local) estimation method such as ML to an aggregate method such as the χ^2-distances between occupancies.

B Program languages and performances

The storing of multidimensional arrays in linear memory can be carried out either by row- or column-major order. Row-major order is used in C/C++, Mathematica, Python, SAS, and others; column-major order is used in Fortran, OpenGL, MATLAB, GNU Octave, R and others. In row-major storage, a multidimensional array in linear memory is organized so that the columns are listed in sequence one after the other. As a consequence, all the codes of the paper follow the column-major order. In fact, using a row-major order would have made the program much slower.

In order to estimate the cost performance in the case of row-major order, we run our fit on equation (54), obtaining parameters $a_0 \approx 2.1 \cdot 10^{-5}$sec, $a_1 \approx 1.4 \cdot 10^{-7}$sec and a consequent time cost of $C \approx 11.9$sec. That is almost twice the time cost in the column-major order implementation of the algorithm. The difference between row- and column-order cost is larger in Matlab than in Octave, because Matlab (from version 6.5 onward) features the Just-In-Time (JIT) accelerator. This improves the speed of M-functions, particularly with loops. Furthermore, the JIT accelerator favors the column-major order implementation over the row one. Specifically, turning off the accelerator approximately doubles the execution time obtained with the onset of the accelerator (the default condition in Matlab).[2] Another reason why Matlab codes have less computational costs is the use of many built-in functions that have multi-threaded implementations, which speed up function calls. Besides that, Matlab, internally uses the Intel Math Kernel Library (Intel MKL) for vector and matrix operations. This gives Matlab a significant advantage over Octave.[3]

In terms of memory cost, even if the performance remains the same, Octave distributions lack of a 64bit version.[4] This reduces the maximum amount of virtual memory a single process can address. Therefore, any official version of Octave has a lower process limit, in terms of memory space, relative to a 64-bit version of Matlab.

for what regards the optimization estimations, the use of parallel computing increases the performance of the calculation and speeds up the procedure (both the grid and the successive parabolic interpolation technique). The use of parallel computing distributes independent simulations to run them in parallel on multiple MATLAB sessions, also known as workers. Distributing the simulations significantly reduces the optimization time because the time required to simulate the model dominates the total optimization time. If we imagine to use the grid method, the optimization is performed S times for every seed. With the use of parallel computing, the software distributes the simulations required for constraint and objective computations. Each workers is devoted to a single computation of the minimum, so that parallel computing reduces the cost of a factor equal to the number of processors in the machine (excluding the time overheads associated with configuring the system for parallel computing and loading the MATLAB workers).

C Procedure to get persistent states of a markovian chain

The discussion of the domain definition discussed in section 3 maintains the same state space as the set

$$S_{N,L} = \left\{ (n_1, \ldots, n_L) \,\middle|\, n_{\geq} 0, \sum_{l=1}^{L} n_l = N \right\} . \tag{67}$$

Each vector of this set can be reached with positive probability and the number of elements of the state space is

$$\dim S_{N,L} = \binom{N+L-1}{N} ,$$

that is the maximum dimension of the Markov chain. The sequence of individual choices is structured so that, at each time step, one agent is selected at random to revise his choice. In general, also other non-random rules could be adopted to select who is called to operate a revision, thus affecting the dynamics of the model. In the present case, however, the aim is to

[2] Use the command `feature accel off`.

[3] Use the commands `version -lapack` and `version -blas` in Matlab to check the version of MKL that Matlab is using.

[4] On Unix-like systems it is possible to compile Octave with 64-bit indexing as shown on gnu.org instructions.

keep the structure of selection as agnostic as possible precisely by attributing to all agents an equal probability to be selected for choice revision. Under this premise, the evolution of the system from configuration $\boldsymbol{n}$ at time t to configuration $\boldsymbol{n}'$ at $t+1$ is defined in terms of the transition probability $P\{\boldsymbol{n}'_{t+1}|\boldsymbol{n}_t\}$. Since the agent that is called to revise his current choice m is selected at random, it follows that $\Pr\{B\} = n_m/N$. This probability must be then multiplied by the probability p_l to select alternative l conditional to the fact that the agent is no longer among those opting for m, that is without considering self-interaction.

$$P\{\boldsymbol{n}'|\boldsymbol{n}\} = \frac{n_m}{N} \frac{a_l + b(n_{l,t} - \delta_{l,m}) + c(n_{l,t} - \delta_{l,m})^2}{\sum_{l=1}^{L} \{a_l + b(n_l - \delta_{l,m}) + c(n_l - \delta_{l,m})^2\}} \tag{68}$$

where the Kronecker term $\delta_{l,m}$ is 1 if $l = m$ and 0 otherwise. As proved in Giulio Bottazzi and Vanni (2014) the Markov chain associated to the non linear model is irreducible, it is also ergodic.

However, it is also possible to consider values of c such that $c < c_{min} < 0$. To see why, let us first define the numerator W_l and the denominator D starting from the probability p_l to select the new alternative. According to equation (68), such probability is

$$p_l = \frac{a_l + bn_{l,t} + cn_{l,t}^2}{A + bN + c\sum_{l=1}^{L} n_l^2} = \frac{W_l}{D} = \frac{g_l}{\sum_{j=1}^{L} g_j}\ , \tag{69}$$

where we consider only the case $m \neq l$ for the sake of simplicity, since we aim merely at estimating the order of magnitude of W_l/D.

We will find a computational procedure such that even starting for a given negative c and an initial configuration $\mathbf{n}_0$ such that the utility function g_l is negative for some l, is still possible to end up to an irreducible markov chain after an adjustment finite time of steps.

The extreme limit for c is a value for which is not possible to have D positive for any combination of the configuration vector $(n_1, \ldots, n_L)$. Using Cauchy's formula:

$$L\sum_{l=1}^{L} n_l^2 - \left(\sum_{l=1}^{L} n_l\right)^2 = \tfrac{1}{2}\sum_{l=1}^{L}\sum_{m=1}^{L}(n_l - n_m)^2,$$

it is possible to recover that:

$$\frac{1}{L}N^2 \leq \sum_{l=1}^{L} n_l^2 \leq N^2 \tag{70}$$

so the minimum c that allows to have a positive denominator in (69) is

$$\underline{c} > -\frac{A + bN}{N^2/L}\ . \tag{71}$$

For any other value of $c > \underline{c}$ it is possible to reduce the set of persistent states and reach the equilibrium after a transient.

Let us first consider the case in which $D > 0\ \forall l$ in equation (69). In this instance, equation (70) implies $c > c_p = -(A + bN)/N^2$, which is close to the extreme condition considered in

Section 3. As long as $D > 0\ \forall l$, equation (69) returns zero if

$$n_l^* > \frac{c\sqrt{\frac{b^2-4ac}{c^2}} - b}{2c}. , \tag{72}$$

which makes W_l negative. This choice forces to decrease the number of agents in the alternatives with occupancy number $n_l > n_l^*$[5]

Let us now consider the case $D \leq 0$, but with

$$-\frac{A+bN}{N^2/L} < c < -\frac{A+bN}{N^2} .$$

If $W_l < 0$, we follow the same prescription as before setting $p_l = 0$. If $W_l > 0$, instead, we set $p_l = |p_l|$, so as to force the occupancy configuration to become more distributed among alternatives. This allows to meet $D > 0$.

In all these cases, the system starts from initial state $\boldsymbol{n}_{in}$ with $c > \underline{c}$. Then, after a finite number of steps in which the states are not persistent, the dynamics is restricted to a state space with persistent states. Hence, the transition probability (68) comes to describe again an irreducible Markov chain with lower cardinality of the state space. This allows to substitute the more restrictive constraint (6) with condition in (71)

[5]Note that for all this value one should expect that $\sum_{l=1}^{L} n_l^* \geq N$ in order to have a state space with at least unitary dimension (only one element).

Acknowledgement

We thank professor Enrico Scalas for his precious feedbacks. The present work has been supported by the Italian Ministry of University and Research as a part of the PRIN 2009 program (grant protocol number 2009H8WPX5).

Code 1: Stochastic evolution of the model.

```
 function n=BGVtool_choice_evolution(a,b0,c,n0,T)
% Non-linear choice model, time evolution.
%   input : N- numero di firms
%                    a=[a1, a2, ...] - pseudo vector (row)
%                    b0= scalar value equal for every choice
%                    c=scalar value equal for every choice
%                    n0= initial configuration to start the process
%                    T = total time of the evolution (simulation time)
%   output :
%                     n=  Matrix_ each colomn (configuration vector)
%
%   F.Vanni 2014.

N = sum(n0);
L=length(a);
b=b0.*ones(1,L);
% Initialization:
a=a(:)'; % forced to be a row vector
n0=n0(:); % forced to be a colomn vector
n=zeros(L,T);
n(:,1)= repmat(n0,[1 1]);
t=1;
% time evolution :
for t=2:T
     % Ehrenfest term - Random Probability for Revision of the Choice
          ra = ceil(N * rand(1,1)); % random number between 1 and N firms.
          [k,~]=find(ra>[0; cumsum(n(:,t-1))] );% look for the place for removal .
               n it's a column vector.
          m=k(end);%row index for the individual to be removed (row=location)
          n(:,t)=n(:,t-1);
          n(m,t)=n(m,t-1)-1; % leaving the old choice

     % Brillouin term - Probability of the new alternative to choose
          pD=sum(a)+b*n(:,t-1)-b(m) +sum(c.*(n(:,t-1)'- (m==(1:length(b)))).^2);
          pN= a+ b.*n(:,t-1)'- b.*(m==(1:length(b))) + c.*(n(:,t-1)'- (m==(1:
               length(b)))).^2;
          p=pN./pD;
              %   if sum(pN<=0)~=0, error('bug in the evolution process'),end % if
                   there is some g negative
          r=rand(1,1);
          [kk,~]=find(r>[0; cumsum(p')] ); % now p' is a coloumn vector
          in=kk(end);
          n(in,t)=n(in,t)+1; % new choice
end
```

Code 2: Random Number Generator with Polya distribution

```
function r=bgv_rndpolya(N,alpha,M)
% N=total length of the generator vector of random numbers (Multivariate
% generator). if N=1 it is a univariate distribution.
% the vector of the parameters alpha=a_l/b;
if nargin<3,M=1;end
alpha=alpha(:)'; % alpha must be a row vector
alpha = repmat(alpha,M,1); % replilcated alpha over m-rows
G=randg(alpha); % Gamma random numbers with unit scale
%prob=G./sum(G); % normalization
D = bsxfun(@times, G, 1./sum(G,2)); % Dirichlet random vaiables
sum(G,2)==0;
r=mnrnd(N,D,M); % Multinomial random numbers of the probabilitiy
r=r'; % it gives the column of the random vector from Polya distribution (
    Dirichlet-multinomial)
```

Code 3: Expected Occupancy Distribution

```
function [f,moments,f_C]= expected_occupancy(a,ni,e_bin)
% Checked from the paper Sectoral and Geographical specificities in the
% spatial structure of economics activities- BottazziDosi,Fagiolo,Secchi
% f= ocupancy frequency
% f_C= occupancy Class frequency
%  a is the vector of externalities
% ni, is the input configuration vetor (typically the observed data nO)
L=length(a);
N=sum(ni);
b=1;
nl=0:N;
tic
f=zeros(N+1,1);
 reverseStr = '';
% for a complete distribution for a given location:
for n_l=1:N+1,
     percentDone = 100 * (n_l) / (N+1);
     msg = sprintf('>> >> >>  percentage done : %4.1f  << << <<', percentDone);
     fprintf([reverseStr, msg]);
     reverseStr = repmat(sprintf('\b'), 1, length(msg));
f(n_l)=sum(marginal_prob_polya((1:L),a,b,ni,n_l-1)); % prob. of having 0 till N
%figure,bar(p)
end
nn_l=repmat(nl,L,1);
moments=0;
 t_toc= toc;
 msg=sprintf('\n work completed in %4.2f minutes\n',round(100*t_toc)/(100*60));
 fprintf(msg);
  Sx=size(ni);
if nargin<3 f_C=f;
else
for k=1:length(e_bin)-1, f_C(k)=sum( f(e_bin(k)+1 : e_bin(k+1)));end,
f_C=f_C'./Sx(1);
end
```

Code 4: Creation of the uniform binning

```
function [e_bin,h,CF,a]=crea_unibin(x)
a=(0:max(x));
i=1;
nB=5;% number of bins
CF=zeros(1,length(a));
while i<=length(a),
    CF(i)=sum(x<=a(i));
    i=i+1;
end
percent=1+nB;
e_vec=ceil(linspace(min(CF),length(x),percent)); % scrolling vector
e=1;
e_bin=zeros(1,length(e_vec));
    while e<=length(e_vec)
        E=sum( CF <= e_vec(e));
        e_bin(e)=E;
        e=e+1;
    end
e_bin(1)=0; % since [0;min(CF)] is empty
h=zeros(length(e_bin)-1,1); % it is the frequency equidistributed inside e_bins
for k=1:length(e_bin)-1
    h(k)=sum(x>=e_bin(k) & x<e_bin(k+1))/length(x) ;
end
```

Code 5: Re-binning procedure

```
function h=insM_ubi(x,bin_edges)
% inserisci nel binning bin_edges creato in precedenza con crea_ubi.m.
% e crea l'istogramma con quel binning
% x- matrix- I expected the evolution is along the row (each time in a different
    column)

e_bin=bin_edges; % if you already gives the binnning with the bins' edges
 Sx=size(x); % I expected the evolution is along the row (each time in a
    different column)
% if Sx(1)<=Sx(2); error('wrong vector try to transpose it'),end
 h=zeros(length(e_bin)-1,Sx(2)); % it is the frequency equidistributed inside
    e_bins
 %h=zeros(length(e_bin)-1,Sx(2))';
 for k=1:length(e_bin)-1
    %h(k)=length( find( x>=e_bin(k) & x<e_bin(k+1) ) )/length(x) ;
    h(k,:)=sum(   x>=e_bin(k)  &  x<e_bin(k+1) ,1 )./Sx(1) ; % faster
 end
 sum(h);
```

Code 6: Distance Measures and Objective Function

```
 function [dX2,dH]=bgv_distribdist(a,b,c,n,O_bin,tmin,T,f_C)
 % generalization (matrix) of distance of vector over the time interval T-tmin
 %       n=is the configuration references as starting point
 %         O_bin = is 't the binning
 %         tmin = time necessary for reach equilibrium
 %          T- evolution over which to calculate the distances
tmax=T;
if length(n(1,:))==1, disp('configuration␣vector')
ne=BGVtool_choice_evolution(a,b,c,n,tmax); % evoluzione a partire dall'ultima
    configurazione
ne=ne(:,(tmin:tmax));
      if nargin<8 , fq1=insMubi(n,O_bin)
      else fq1=f_C;
      end
freq2=insMubi(ne,O_bin);% se ne dimensionale freq2 ha per colonna per ogni T
sf=size(freq2);
freq1=repmat(fq1,1, sf(2)); % crete a matrix same size of freq2
       % distance measure in terms of chi square test of Pearson
       di=((freq2)-(freq1)).^2./(freq1) ;
       di(isnan(di) | isinf(di)) = 0; %set to 0 non finite occupancy frequency
       dX2=sum(di   ,1);
        %Hellinger  distance
        dH=1/sqrt(2).*sqrt( sum( (sqrt(freq1) - sqrt(freq2)).^2 )    );
num_bins=size(freq1);
elseif  length(n(1,:))==T,      disp('configuration␣matrix␣in␣evolution')
 ne=BGVtool_choice_evolution(a,b,c,n(:,tmin),tmax); % evoluzione a partire dall'
    ultima configurazione
ne=ne(:,(tmin:tmax));
fq1=insMubi(n(:,tmin:tmax),O_bin) ;
freq2=insMubi(ne,O_bin);% se ne dimensionale freq2 ha per colonna per ogni T
sf=size(freq2);
freq1=fq1;
       % distance measure in terms of chi square test of Pearson
       di=((freq2)-(freq1)).^2./(freq1) ;
       di(isnan(di) | isinf(di)) = 0; %set to 0 non finite occupancy frequency
       dX2=sum(di   ,1);
         %Hellinger  distance
        dH=1/sqrt(2).*sqrt( sum( (sqrt(freq1) - sqrt(freq2)).^2 )    );
num_bins=size(freq1);
elseif length(n(1,:))~=T,error('input␣configuration␣wrong␣in␣length'),
end
```

References

G. Bottazzi and U.M. Gragnolati. Cities and clusters: economy-wide and sector specific effects in corporate location. *Regional Studies*, 2012. forthcoming.

G. Bottazzi and A. Secchi. Repeated choices under dynamic externalities. LEM Working Paper Series, September 2007. URL http://www.lem.sssup.it/WPLem/files/2007-08.pdf.

Giulio Bottazzi, Giovanni Dosi, Giorgio Fagiolo, and Angelo Secchi. Sectoral and geographical specificities in the spatial structure of economic activities. *Structural Change and Economic Dynamics*, 19(3):189–202, 2008.

R. Brent. *Algorithms for minimization without derivatives.* Prentice-Hall Inc., 1973.

Luc Devroye. *Non-Uniform Random Variate Generation.* Springer, 1986.

U. Garibaldi and E. Scalas. *Finitary probabilistic methods in econophysics.* Cambridge University Press Cambridge, 2010.

James E. Gentle. *Random Number Generation and Monte Carlo Methods (Statistics and Computing).* Springer, 2004.

Ugo Gragnolati Giulio Bottazzi and Fabio Vanni. A numerical estimation method for discrete choice models with non-linear externalities,. *LEM Working Paper Series*, 2014.

K.W. Ng, G.L. Tian, and M.L. Tang. *Dirichlet and related distributions: Theory, methods and applications*, volume 888. John Wiley & Sons, 2011.

B.W. Silverman. *Density estimation for statistics and data analysis*, volume 26. Chapman and Hall/CRC, 1998.