INFORMATICA, 2022, Vol. 0, No. 0, 1–20

© 2022 *Vilnius University*
DOI: https://doi.org/10.15388/22-INFOR479

Randentropy: A Software to Measure Inequality in Random Systems

Guglielmo D'AMICO[1,*], Stefania SCOCCHERA[2], Loriano STORCHI[3,*]

[1] *Department of Economics, University G. D'Annunzio Chieti-Pescara, viale Pindaro, 42-65127 Pescara, Italy*
[2] *Banco BPM S.p.A., Credit Risk Models, Verona, Italy*
[3] *Department of Pharmacy, University G. D'Annunzio Chieti-Pescara, via dei Vestini 31, 66100 Chieti, Italy*
e-mail: g.damico@unich.it, loriano@storchi.org

Received: March 2021; accepted: March 2022

Abstract. The software Randentropy is designed to estimate inequality in a random system where several individuals interact moving among many communities and producing dependent random quantities of an attribute. The overall inequality is assessed by computing the Random Theil's Entropy. Firstly, the software estimates a piecewise homogeneous Markov chain by identifying the change-points and the relative transition probability matrices. Secondly, it estimates the multivariate distribution function of the attribute using a copula function approach and finally, through a Monte Carlo algorithm, evaluates the expected value of the Random Theil's Entropy. Possible applications are discussed as related to the fields of finance and human mobility.

Key words: random entropy, Markov reward model, copula, change-point.

1. Introduction

The issue of measuring inequality in a system found extensive treatment in the literature. One interesting approach is based on entropic measures. Starting from the pioneering work by Shannon (1948) on the mathematical theory of communication, the concept of entropy has found a rapid development and diffusion in many scientific communities. Notable examples are statistics (see, e.g. Kullback and Leibler, 1951), statistical mechanics (see, e.g. Jaynes, 1957), economy (see, e.g. Theil, 1967) and ecology (see, e.g. Phillips *et al.*, 2006), just to name a few.

Recent efforts have been dedicated mainly to introduce new entropies as the cumulative residual entropy (see, Rao *et al.*, 2004) or the cumulative past entropy (see, Di Crescenzo and Longobardi, 2009). In the meantime, and mainly motivated by economic problems, the notion of random entropy has emerged in terms of a normalization of a random process. The random entropy shares the same functional form as the classical entropy but is related to a random process (D'Amico and Di Biase, 2010). This more general entropy was called

*Corresponding authors.

by the author Dynamic Theil Entropy. Nevertheless, we refer to it as Random Entropy, to avoid any possible misunderstanding with other dynamic entropies which are expressed as deterministic functions as in Di Crescenzo and Longobardi (2002), Asadi and Zohrevand (2007) and Calì *et al.* (2020).

The Random Entropy allows to quantify uncertainty in a random system evolving in time and encompasses recent approaches and measures introduced in Curiel and Bishop (2016). In this paper, we consider the general model considered in a previous work (D'Amico *et al.*, 2019) and we present a software that permits the calculation of the inequality in a general system composed by a number of interacting individuals. Any individual moves among several communities in time and according to its membership, and depending on that of the other individuals, produces an attribute. The dynamic of individuals among the communities is described according to a piecewise homogeneous Markov chain which requires the identification of an unknown number of change-points (i.e. where the Markov chain changes its dynamic). Conditional on the occupancy of the communities, the individuals produce an attribute in quantities expressed by a multivariate probability distribution where the dependence structure is managed by a copula function. Finally, using a Monte Carlo algorithm, we show how to compute the moments of the Random Entropy.

The main innovation brought by this research is the building of the software **Randentropy**. It contemplates different aspects that were only partially considered in other research papers. Indeed, different studies deal with software and packages related to multi-state models of Markovian type. For example, in Ferguson *et al.* (2012) the authors consider a package for computing marginal and conditional occupation probabilities for Markov and non-Markov multi-state models, including the censoring problem and the use of covariates. In Jackson *et al.* (2011), multi-state models for panel data observed continuously and generally based on the Markov assumption have been instead considered. The possibility to obtain a time-varying model is considered using piecewise-constant time-dependent covariates. Contrarily to these studies, our software gives different transition probability matrices according to the change-point detection methodology presented in Polansky (2007), which is based only on observations of the Markov process and not on additional covariates. Moreover, once the piecewise homogeneous Markov chain is identified, the software provides sequences of dependent random vectors denoting the ownership of an attribute by the individuals of the system. Thus, the system becomes a multivariate Markov reward process on which the Random Entropy is evaluated. To our knowledge, our software is the only one that computes the Random Entropy and does it in a very general framework that encompasses recent contributions presenting diversity measurement based on (deterministic) entropy where the migration of individuals among the communities is not allowed, see Marcon and Hérault (2015a). Of potential interest is also the use of the software **Randentropy** to problems approached with the traditional concept of entropy, see e.g. Behrendt *et al.* (2019) and Saad and Ruai (2019).

The subsequent sections of this paper present the general mathematical model, relevant scenarios of application and the software main characteristics, both the CLI (Command Line Interface) and GUI (Graphical User Interface) are described.

2. Theory

The main function driving the development of the software we are presenting here (i.e. **Randentropy**) refers to the computation of a measure of inequality on the distribution of a given attribute among a set of N individuals. The quantity of this attribute depends on a discriminatory criterion, according to whom the individual belongs to a given group. Accordingly to the nomenclature mainly derived within the ecology community, but preserving its general validity also in other domains, we denote the set of individuals as a meta-community that is partitioned in several interacting groups called communities. This description is the same adopted in Marcon and Hérault (2015b).

Let denote the meta-community by $\mathcal{C}$ and the number of its members by N. Each individual $c \in \mathcal{C}$ belongs, at any time $t \in \mathbb{N}$, to one of D different communities that form the meta-community $\mathcal{C}$. The variable $x^c(t)$ with values in $E = \{1, 2, \ldots, D\}$ denotes the community to which the individual c belongs to at time t. Every time the individual is a member of a given community, it owns a quantity of the personal attribute denoted by $s^c(t)$. The considered system is stochastic, in the sense that each individual passes through different communities randomly in the course of time and, as a consequence, the personal attributes evolve over time randomly. In this way, the proposed approach is more general as compared to that proposed by Marcon and Hérault (2015b), where the possibility for members to migrate from a community to another is not permitted.

The sequence of the visited communities by any individual $c \in \mathcal{C}$, that is $\{x^c(t)\}_{t\in\mathbb{N}}$, is assumed to be a realization of a stochastic processes $\mathbf{X}^c := (X^c(t))_{t\in\mathbb{N}}$. Thus, the sequences of individual's attribute, that is $\{s^c(t)\}_{t\in\mathbb{N}}$, evolve randomly, too. We will denote, from now on, the stochastic process describing the evolution of individuals' attribute as $\mathbf{S}^c := (S^c(t))_{t\in\mathbb{N}}$. The processes $\mathbf{X}^c$ and $\mathbf{S}^c$ evolve jointly, meaning that: the evolution of the process $\mathbf{S}^c$ is driven by the stochastic process $\mathbf{X}^c$, which controls it. A precise description of this mechanism follows.

Firstly, we assume an independence assumption between the dynamics of the individuals. Thus, the community process for every individual will be denoted simply by $\mathbf{X} = \mathbf{X}(t)$, and the reference to specific individual $c \in \mathcal{C}$ is dropped.

Moreover, we assume that $\mathbf{X} = \mathbf{X}(t)$ is distributed according to a piecewise homogeneous Markov chain (PHMC). The process $\mathbf{X}$ is a PHMC taking values in the finite set E, if a positive number of change-points k, a sequence $\tau_0 = 0 < \cdots < \tau_k$ of increasing times and a sequence ${}^{(0)}\mathbf{P}, \ldots, {}^{(k)}\mathbf{P}$ of stochastic matrices (such that for any $l \in \mathbb{N}, l \leqslant k$) exist, it ensues that: for any $t \in \{\tau_l, \ldots, \tau_{l+1} - 1\}$ and any $i, j \in \mathrm{E}$ the following Markov property holds:

$$\begin{aligned}&\mathbb{P}\big(X(t+1) = j \big| X(t) = i, X\big(0 : (t-1)\big) = i_{0:(t-1)}\big)\\ &\quad = \mathbb{P}\big(X(t+1) = j \big| X(t) = i\big) = {}^{(l)}p_{ij}.\end{aligned}$$

The symbols $i_{0:(t-1)} = (i_0, \ldots, i_{t-1}) \in E^t$, $X(0 : (t-1)) = (X(0), \ldots, X(t-1))$ and $\{\tau_l, \ldots, \tau_{l+1} - 1\}$ represents the time interval, enclosed between the lth and the $l+1$th change-point where the dynamics at community-level are fixed and described by the transition probability matrix ${}^{(l)}\mathbf{P} = \{{}^{(l)}p_{ij}\}_{i,j\in E}$.

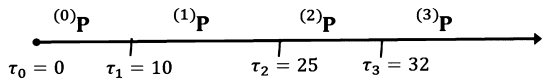


Fig. 1. Example of three change-points.

Intuitively, the term piecewise refers to the existence of some points in time where the dynamic changes consistently. These times are called change-points. They break up the timeline into several sub-periods within whom the Markov process is homogeneous.

However, for the sake of clarity of presentation, consider the example illustrated in Fig. 1 where three change-points are considered at times $\tau_1 = 10$, $\tau_2 = 25$, $\tau_3 = 32$. For every time $t \in \{\tau_0, \ldots, \tau_1 - 1\} = \{0, \ldots, 9\}$ the dynamic of the process is given by the transition probability matrix ${}^{(0)}\mathbf{P}$, thus it results that $\forall t \in \{0, \ldots, 9\}$:

$$\mathbb{P}\big(X(t+1) = j \,\big|\, X(t) = i, X\big(0 : (t-1)\big) = i_{0:(t-1)}\big) = {}^{(0)}p_{ij}.$$

At any time point t during the interval $\{\tau_1, \ldots, \tau_2 - 1\} = \{10, \ldots, 24\}$ it results that

$$\mathbb{P}\big(X(t+1) = j \,\big|\, X(t) = i, X\big(0 : (t-1)\big) = i_{0:(t-1)}\big) = {}^{(1)}p_{ij}.$$

A similar argument applies to the dynamic during the intervals $\{\tau_2, \ldots, \tau_3 - 1\} = \{10, \ldots, 24\}$ and $\{\tau_3, \ldots\} = \{32, \ldots\}$ where the dynamics are given by the matrices ${}^{(2)}\mathbf{P}$ and ${}^{(3)}\mathbf{P}$, respectively.

Next step concerns the specification of the processes describing the personal attributes, i.e. $\mathbf{S}^c$. We consider a meta-community where the personal attributes of the individuals can be considered to be dependent among each others.

This strategy is pursued first by assuming that the marginal distributions of the attributes of the individuals allocated in the same community at a given time share the same probability distribution function. Formally, let F_x be the conditional distribution of attribute $S^c(t)$ knowing the community $X^c(t) = x$ of the individual $c \in \mathcal{C}$, then

$$F_x := \mathcal{D}\big(S^c(t) \,\big|\, X^c(t) = x\big), \quad \text{for any } t \in \mathbb{N},$$

where, for a given random variable A, the symbol $\mathcal{D}(A)$ denotes its probability distribution.

Before presenting our second main assumption we need to present the concept of copula which will be a key issue in the model and software.

An N-dimensional copula C is any function $C : [0,1]^N \to [0,1]$, grounded and N-increasing whose marginals satisfy

$$C_i(u) = C(1, \ldots, 1, u, 1, \ldots, 1) = u), \quad \forall u \in [0,1].$$

From the above definition of the copula, it is understandable that if we consider a set of univariate cumulative distribution functions $F_1, F_2, \ldots, F_N$, the function

$C(F_1, F_2, \ldots, F_N)$ is a multivariate distribution function with marginal distributions F_i, $i = 1, \ldots, N$.

Additionally, a dependence structure is introduced through the application of a copula function. This is formally done advancing the second main assumption stating that: the conditional joint distribution of $(S^1(t), \ldots, S^N(t))$ knowing $(X^1(t) = x^1, \ldots, X^N(t) = x^N)$ is given by

$$\mathcal{D}\big(S^1(t), \ldots, S^N(t) \big| X^1(t) = x^1, \ldots, X^N(t) = x^N\big) = C_\theta(F_{x^1}, \ldots, F_{x^N}),$$

where C_θ is the copula, with dependence parameter θ. According to the considered copula function, θ may also be a vector of parameters.

A notable example of copula function is the Normal (or Gaussian) copula. Let $\mathbf{R}$ be a correlation matrix and denote by $\Phi_{\mathbf{R}}$ the standardized multivariate normal distribution with correlation matrix $\mathbf{R}$. The Gaussian copula is defined according to:

$$C(u_1, \ldots, u_N; \mathbf{R}) = \Phi_{\mathbf{R}}\big(\Phi^{-1}(u_1), \ldots, \Phi^{-1}(u_N)\big),$$

where the vector $(u_1, \ldots, u_N)$ belongs to the unit cube $[0, 1]^N$ and $\Phi^{-1}(\cdot)$ is the inverse of the standard normal cumulative distribution function.

In general, the corresponding density of the copula is

$$c(u_1, \ldots, u_N) = \frac{\partial C(u_1, \ldots, u_N)}{\partial u_1 \cdots \partial u_N}$$

and in the Gaussian case it assumes the well known form

$$c(u_1, \ldots, u_N) = \frac{\exp(-\frac{1}{2} K^T \cdot (\mathbf{R} - I) \cdot K)}{\sqrt{\det(\mathbf{R})}},$$

where $K_i = \Phi^{-1}(u_i)$, see e.g. Durante and Sempi (2016).

Here, the parameters are represented by the correlation matrix $\mathbf{R}$.

As we are interested in measuring the inequality of the distribution of attributes in the meta-community, we need to introduce a measure of inequality. In particular, the measure of inequality we consider allows the user to face with stochastic processes. The measure is based on the Theil entropy (see Theil, 1967), closely related to the Shannon entropy (see Shannon, 1948). Given a probability distribution

$$\mathbf{p} = \big(p^1, \ldots, p^N\big), \quad p^i \geqslant 0, \quad \sum_{i=1}^{N} p^i = 1,$$

the Theil index, $T(\mathbf{p})$ of $\mathbf{p}$, is defined as the Kullback–Leibler (KL) divergence $\mathbb{K}(\mathbf{p}|\mathbf{u})$ between $\mathbf{p}$ and the uniform distribution $\mathbf{u}$, or equivalently, as the difference between $\log(N)$ and the Shannon entropy $\xi(\mathbf{p})$. Precisely,

$$T(\mathbf{p}) := \mathbb{K}(\mathbf{p}|\mathbf{u}) := \sum_{i=1}^{N} p^i \log\big(N \cdot p^i\big) = \log(N) - \xi(\mathbf{p}), \tag{1}$$

where $\xi(\mathbf{p}) = -\sum_{i=1}^{N} p^i \log p^i$. The usual convention that if $p^i = 0$ for some i, the value of the corresponding expression $0\log(0)$ is set to be 0, is considered.

The definition of Theil index has been extended for stochastic processes by D'Amico and Di Biase (2010) and successively applied and further investigated in D'Amico *et al.* (2012) and in D'Amico *et al.* (2014) for an additive decomposition of this index. The random extension of the Theil index is, indeed, introduced.

Let $sh^c(t)$ be the share of the attribute held by individual $c \in \mathcal{C}$ at time $t \in \mathbb{N}$. It is defined as the proportion of its own attribute $S^c(t)$ relative to the sum of the attribute over all individuals, i.e.

$$sh^c(t) = \frac{S^c(t)}{\sum_{d\in\mathcal{C}} S^d(t)}.$$

The vector of shares of attributes at time t, $sh(t) := (sh^c(t))_{c\in\mathcal{C}}$ defines a probability distribution on the set of countries $\mathcal{C}$. Note that $\mathbf{sh} := (sh(t)_{t\in\mathbb{N}})$ is a stochastic process that depends on the stochastic processes $\mathbf{S}^c$, controlled by $\mathbf{X}^c$.

We denote the Random Entropy in the meta-community by the stochastic process $DT(\mathbf{sh}(t))$, defined according to the following equation

$$DT(\mathbf{sh}(t)) = \sum_{c\in\mathcal{C}} sh^c(t)\log\big(N \cdot sh^c(t)\big), \quad t \in \mathbb{N}. \tag{2}$$

In this case also, if $sh^c(t) = 0$ for some c and t, the value of the corresponding expression $0\log(N \cdot 0)$ is set to be 0.

An explicit formula for the expected value of $DT(\mathbf{sh}(t))$ has been provided in D'Amico *et al.* (2019). Nevertheless, that formula can only be effectively implemented for small sized meta-communities and number of communities. In the contrary case, a Monte Carlo simulation approach can be successfully implemented. The proposed algorithm simulates repeatedly the trajectories of all individuals according to the underlying Markov model, providing the sequence of communities to which each individual belongs in time. Moreover, the personal attributes are simulated by using the copula function with marginal distribution for each individual dependent on the community of membership. The expected value of the Random Entropy can be estimated by averaging, for each time, over all simulated attributes in the meta-community.

The whole computational procedure is made of several steps, thus, to simplify the readability of the reported pseudocode (see Algorithm 1), we are omitting some of the preliminary tasks, such as: the identification of the number K and dislocation in time of the change points $\{\tau_k\}_{k=1}^{K}$; the corresponding estimation of the transition probability matrices ${}^{(l)}\mathbf{P} = \{{}^{(l)}p_{ij}\}_{i,j\in E}$; the cdf's $\{F_x, x \in E\}$ of attribute depending on the community x; and the identifiability of the copula function C_θ. Obviously, the software **Randentropy** is designed to solve all the aforementioned tasks, including the implementation of the Monte Carlo algorithm which represents the very last step of the computation.

For easiness of notation we adopt the following vectorial notation along the Algorithm 1:

Algorithm 1 Monte Carlo Simulation of the Random Entropy

for $c = 1 : N$
  set $X(0, c) = i_c$;
end for
set $k = 1$;
3. set $h = \tau_{k-1}$ while $h < (\tau_k \wedge M)$
  for $c = 1 : N$
    sample the random variable $X \sim {}^{(k)}p_{X(h-1,c),\cdot}$
    set $X(h, c) = X(\omega)$;
  end for
  sample $(v_1, v_2, \ldots, v_N)$ from N independent Uniform $U(0, 1)$;
  set $u_1 = v_1$ and $S(h, 1) = F^{-1}_{X(h,1)}(u_1)$;
  for $b = 2 : N$
    set $C_\theta(v_b|(u_1, \ldots, u_{b-1})) = \frac{\partial^{b-1}_{(u_1,\ldots,u_{b-1})} C_\theta(u_1,\ldots,u_{b-1},v_b,1,\ldots,1)}{\partial^{b-1}_{(u_1,\ldots,u_{b-1})} C_\theta(u_1,\ldots,u_{b-1},1,1,\ldots,1)}$;
    set $u_b = C^{-1}_\theta(v_b|(u_1, \ldots, u_{b-1}))$;
    set $S(h, b) = F^{-1}_{X(h,b)}(u_b)$;
  end for
  for $b = 1 : N$
    set $s(h, b) = \frac{S(h,b)}{\sum_{c=1}^{N} S(h.c)}$;
  end for
  set $DT(h) = \sum_{c=1}^{N} s(h, c) \cdot \log(N \cdot s(h, c))$;
  set $k = k + 1$ and continue to 3.
end while

- $X(t, c) = X^c(t)$ denotes the community to which the individual c belongs to at time t. Thus, $X(\cdot\,, \cdot)$ is a matrix whose values are element of E. Its i-th row $X(i, \cdot)$ provides the meta-community configuration at time i, that is, the allocation of the individuals at that time among the communities. Instead, the j-th column of the matrix $(X(\cdot, j))$ gives the trajectory of the individual j in time, that is, the sequence of communities it visited in time;
- $s(t, c) = sh^c(t)$ denotes the attribute held by individual c at time h. Thus, $s(\cdot\,, \cdot)$ is a matrix whose values are non-negative real numbers. Its i-th row $s(i, \cdot)$ provides the share of the attribute owned by the individuals of the meta-community at time i; it represents a probability distribution. The j-th column of the matrix $s(\cdot\,, j)$ shows instead the evolution in time of the share of the attribute owned by the individual j;
- $DT(t) = DT(\mathbf{sh}(t))$ denotes the value of the Random Entropy at time t in the meta-community. Specifically, it gives the Theil's entropy computed on the probability distribution $s(t, \cdot)$ which represents a realization of the Random Entropy in a given simulation;
- M denotes the horizon time of the simulation.

The Algorithm 1 generates a vector of observations $(u_1, \ldots, u_N)$ from random variables having Uniform $U(0, 1)$ marginals and sharing the N-dimensional Copula C_θ. To do this, first a vector $(v_1, \ldots, v_N)$ is generated from N independent uniform distributions over [0, 1] and then their conditional distributions are assessed using the copula, in fact the quantity

$$C_\theta\big(v_b\big|(u_1, \ldots, u_{b-1})\big) = \frac{\partial^{b-1}_{(u_1,\ldots,u_{b-1})} C_\theta(u_1, \ldots, u_{b-1}, v_b, 1, \ldots, 1)}{\partial^{b-1}_{(u_1,\ldots,u_{b-1})} C_\theta(u_1, \ldots, u_{b-1}, 1, 1, \ldots, 1)},$$

gives the conditional cumulative distribution function of a uniform random variable U_b given the values of the previous $b - 1$ variables, i.e.

$$\mathbb{P}(U_b \leqslant v_b | U_1 = u_1, \ldots, U_{b-1} = u_{b-1}).$$

The computation of $u_b = C_\theta^{-1}(v_b|(u_1, \ldots, u_{b-1}))$ produces the number $u_b \in [0, 1]$ that is dependent on $(u_1, \ldots, u_{b-1})$. The value u_b is then used to generate the attribute of individual b through the inverse of the cumulative distribution function of the specific community to which the individual belongs to at time h, i.e.

$$S(h, b) = F^{-1}_{X(h,b)}(u_b).$$

In this way, the resulting individual attributes at any time h show a dependence to each other which is due to the copula function.

The result of Algorithm 1 is a sequence of values $\{DT(h)\}$, $h = 1, \ldots, M$. Now, if we execute the cited algorithm L times, we can denote by $\{DT^{(l)}(h)\}$, $h = 1, \ldots, M$ the result of the simulation at the l-th repetition. Then, we are able to provide an estimation of the expected value of the Random Entropy by the average value in the L simulation, i.e.

$$\widehat{DT}(h) = \frac{1}{L} \sum_{l=1}^{L} DT^{(l)}(h), \quad h = 1, \ldots, M.$$

3. Relevant Scenarios of Application

In this section we provide a short description of two possible domains of application of the model. Certainly, a variety of additional situations falls well within the described theoretical setting.

3.1. *Financial Inequality in an Economic Area*

This application was originally considered by D'Amico *et al.* (2018a) and (2018b) and successively in a more comprehensive way in D'Amico *et al.* (2019). In this framework we have a meta-community that coincides with a given set of countries all belonging to

a given Economic Area. A possible case is represented by the European Economic Area. Practically, every country receives a note about its financial creditworthiness, which is expressed in terms of a sovereign credit rating, see e.g. Trueck and Rachev (2009) and D'Amico *et al.* (2017). Credit ratings are measured in an ordinal scale and assigned by the rating agencies. Moody's, Standard & Poor's and Fitch are three major among others.

Each rating class can be seen as a community, in which the countries are allocated at every time. According to its own riskiness (expressed by rating class), each country pays interest rates on its debt. When the interest rates are compared to a benchmark they define the so-called credit spreads. Thus, credit spreads can be seen as personal attributes held by each country in time.

Empirical analysis has shown that credit spreads of European countries are positively correlated, with the exception of Denmark, Sweden and the United Kingdom. To model this complex correlation structure a copula function can be used according to our framework. Once the credit spreads are obtained, it is possible to compute the vector of attributes at time t, $sh(t) := (sh^c(t))_{c\in\mathcal{C}}$. Finally, the computation of the expected value of $DT(\mathbf{sh}(t))$ gives an effective tool for forecasting the financial inequality in an economic area and its evolution in time.

3.2. *Human Mobility and Environmental Implications*

Another area in which the $DT(\mathbf{sh}(t))$ might be useful is related to the analysis of human mobility data and specific attributes of interest, see e.g. Song *et al.* (2006) and Krumme *et al.* (2013). Evidently, it is possible to use Markov chains as a tool to measure patterns of movements of individuals in a given area. Substantially, the global area, in which the totality of individuals (the meta-community) lives, is partitioned into different locations (communities) and the probability of the next visited location is assumed to depend only on the current location and not on the previous ones. As members of a given location, individuals possess a personal attribute that can be of different nature.

For example, it would be possible to consider pollution as a variable depending on the specific location, and to measure by the index $DT(\mathbf{sh}(t))$ the inequality of the distribution of pollution in the global area and how it may evolve in time. Another possible choice, for the personal attribute, can be the level of expenditures, in such a case the Random Entropy could be used for assessing the inequality of expenditures in the area. The latter approach can represent an indeed useful tool to optimize the displacement policies of new markets and stores.

4. Computational Details and Applications

The software we are presenting here has been engineered so that the main computational kernel is included in a single python module named **randentropymod** (Storchi, 2020). The cited module contains two classes: **randentropykernel** and **changepoint**. The two classes are devoted to the Markov reward approach computation, and to the change-point estimation, respectively. The full software bundle is then composed by two Command

Line Interfaces (CLIs): **randentropy.py** and **randentropy_qt.py**, and a single Graphical User Interface (GUI) based on PyQt5 Summerfield (2007) (i.e. the Python binding of the cross-platform GUI toolkit).

While the two mentioned CLIs have been specifically developed to perform separately the Markov reward computation (i.e. **randentropy.py**) and the change-point estimation (i.e. **changepoint.py**), the GUI has a wider ability. Indeed, the GUI may be used to perform both the change-point estimation as well as the Markov reward computation, and clearly also to easily visualize and explore the obtained results.

The full software suite has been developed within the Linux OS environment. However, once the needed packages are downloaded and installed, it should work, without restrictions, also under Mac OS and Windows thanks to the intrinsic portability nature of the Python programming language. The Python packages, in addition to the aforementioned PyQT5, strictly needed to run the code are: `Numpy` (see Dubois *et al.*, 1996) and `Scipy` (Jones *et al.*, 2001) used to engineered the numerical tasks, `matplotlib` for the plots and data visualization (see Hunter, 2007).

4.1. *The Randentropykernel Class and Related CLI*

As already stated, the **randentropykernel** class is devoted to the computation of the Random Entropy which is based on the Markov model with dependent rewards as described in Section 2. The class is made of several methods as the one to specify the community matrix (i.e. **set_community**) and the attributes matrix (i.e. **set_attributes**), which correspond to the matrices $X(\cdot\,,\cdot)$ and $s(\cdot\,,\cdot)$ used in the algorithm, respectively. There are clearly various methods to tune the computation behaviour such as: set the number of Monte Carlo simulation steps (i.e. **set_num_of_mc_iterations**), or the simulated time period **set_simulated_time**. Finally, the user has the ability to enable or disable the copula function via the **set_usecopula** method, and clearly to perform the main computation calling the **run_computation** method. The software makes use of a Gaussian (or Normal) Copula which is probably the most frequently used copula in the applications. Nevertheless, Algorithm 1 is general and holds for any copula function, and since the software is open source, any researcher can adapt it to consider a different copula. Once the computation is completed, the user can retrieve all the results: the first and the second-order moments of the Random Entropy using **get_entropy** and **get_entropy_sigma**, respectively.

The **randentropy.py** is the CLI that is naturally bonded to the mentioned class. As can be seen from Fig. 2, the user has the possibility to specify two input matrices (i.e. to specify both their locations and names): the first one representing the community matrix, while the second is the Attributes one. The mentioned matrices may be stored both on a MatLab file or on a CSV style one.

Evidently, the CLI options reported in Fig. 2 reflect the cited **randentropykernel** capabilities. Then, `-s` allows for the bin width specification, needed to estimate the probability distribution of the attribute given the community membership. Secondly, `-t` enables the user to specify the simulated period, and `-n` refers to the number of Monte Carlo iterations. Optionally, the `-i` flag allows the user to run the simulation after computing the stationary distribution.

```
usage: randentropy.py [-h] -m RMATFILENAME -b IMATFILENAME [-s STEP] [-t TPREV]
                      -n MAXRUN [-M NAMEOFMATRIX] [-B NAMEOFBPMATRIX] [-v] [-i]
                      [-S] [-c]

optional arguments:
  -h, --help            show this help message and exit
  -m RMATFILENAME, --rmat-filename RMATFILENAME
                        Observed transition matrix filename
  -b IMATFILENAME, --imat-filename IMATFILENAME
                        Rewards matrix filename
  -s STEP, --step STEP  Bin width
  -t TPREV, --time-prev TPREV
                        Simulated period
  -n MAXRUN, --max-run MAXRUN
                        Monte carlo iterations
  -M NAMEOFMATRIX, --name-of-matrix NAMEOFMATRIX
                        Name of the observed transition matrix (default=ratings)
  -B NAMEOFBPMATRIX, --name-of-bpmatrix NAMEOFBPMATRIX
                        Name of the rewards matrix (default=interest_rates)
  -v, --verbose         Increase output verbosity
  -i, --time-inf        Run the simulation using stationary distribution
  -S, --seed            Use a seed for the random generator
  -c, --use-copula      Use the copula based Markov reward
```

Fig. 2. CLI for the Markov reward approach.

It is finally somehow interesting to report here that: in case one wants to perform the simulation using the stationary distribution π of the Markov chain $\mathbf{X} = \mathbf{X}(t)$ we need to solve a linear matrix equation $ax = b$. To solve the given equation one can compute the value of x that minimizes the Euclidean 2-norm $\|b - ax\|^2$. This has been done by applying a specific function within Numpy libraries (see Dubois *et al.*, 1996).

4.2. *The Changepoint Class and Related CLI*

As already stated within the **randentropymod** module there is also the **changepoint** class. The cited class, and thus the related CLI, is devoted to detect the position of k change-points, where $k = 1, 2, 3$. In particular, the code finds the positions of the change-points by maximizing the likelihood function of the observed trajectories of the members within their communities. At the same time, the Λ test is carried out in order to assess statistically significant differences among the transition probability matrices found. Additional details on this statistical test are available in Polansky (2007) and D'Amico *et al.* (2019).

The most relevant methods within the class are needed to specify the transition matrix (i.e. **set_community**) and the number of change-points to be detected (i.e. **set_num_of_cps**). Once the initial settings have been specified, the main computation starts using the **compute_cps** method. Finally, the calculated x change-points can be retrieved using the **get_cp1_found**, **get_cp2_found** and **get_cp3_found,** respectively, for the first, second and third change-point.

Once again the CLI options, reported in Fig. 3, as expected, reflect the class capabilities. Thus, to run the code, the input transition matrix has to be specified, in terms of a Matlab or a CSV filename, as well as the matrix name within the file (options `-m` and `-M`, respectively). The number of change-points to be considered has to be defined as well (i.e. using the `-c` option), otherwise the code will run assuming a single change-point. Optionally, an output filename, where all the results are written, can be specified using the `-o/-output-file` option.

```
usage: changepoint.py [-h] -m RMATFILENAME [-M NAMEOFMATRIX] [-c NUMOFCP]
                      [-o OUTF] [--iterations] [--cp1-start CP1START]
                      [--cp1-stop CP1STOP] [--cp2-start CP2START]
                      [--cp2-stop CP2STOP] [--cp3-start CP3START]
                      [--cp3-stop CP3STOP] [--delta-cp DELTACP]
                      [--perform-test PERFORMTEST]

optional arguments:
  -h, --help            show this help message and exit
  -m RMATFILENAME, --rmat-filename RMATFILENAME
                        Observed transition matrix filename
  -M NAMEOFMATRIX, --name-of-matrix NAMEOFMATRIX
                        Name of the observed transition matrix (default:
                        ratings)
  -c NUMOFCP, --numof-cp NUMOFCP
                        Number of change points 1, 2, 3 (default: 1)
  -o OUTF, --output-file OUTF
                        Dumps all values (default: change.txt)
  --iterations          Use iteration number instead of progressbar
  --cp1-start CP1START  CP 1 start from (default: 1)
  --cp1-stop CP1STOP    CP 1 stop (default: -1 i.e. will stop at maximum time)
  --cp2-start CP2START  CP 2 start from (default: 1)
  --cp2-stop CP2STOP    CP 2 stop (default: -1 i.e. will stop at maximum time)
  --cp3-start CP3START  CP 3 start from (default: 1)
  --cp3-stop CP3STOP    CP 3 stop (default: -1 i.e. will stop at maximum time)
  --delta-cp DELTACP    Delta time between CPs (default: 1 no delta)if delta <=
                        0 will use cp2 and cp3 start and stop values
  --perform-test PERFORMTEST
                        Perfom Lambda test for the specified
                        cp1:cp2;cp3;num_of_run (default: "-1;0" i.e. no test is
                        performed) will use also cp2 and cp3 if --numof-cp is
                        equal to 2 or 3
```

Fig. 3. CLI for change-point detection algorithm.

Finally, we introduced some methods, and clearly the relative CLI options that can be used also to distribute the computational burden among several processes, thus CPUs. Indeed, while working with a huge amount of data it can be convenient to specify a range of time within which the algorithm is carried out, or to use a specific time distance between two change-points. Thus, the user has the ability to define a range of time for the first change-point (the same applies for the others) via the `set_cp1_start_stop` method. Similarly, using the `set_delta_cp` method, one can specify the delta time to be considered among the change-points.

4.3. *Graphical User Interface*

All the previously illustrated functionalities have been integrated also on a GUI (Graphical User Interface). The GUI has been implemented using PyQT5, a comprehensive set of Python bindings for Qt v5 (PyQT, 2012). While we implemented two different CLIs, to fully cover the various aspects implemented within the `randentropymod`, the GUI is unique and can be access via the `randentropy_qt.py` file (Storchi, 2020).

The computation starts after choosing an input file, it can be both a Matlab, as well as a CSV, containing two matrices. The first matrix has to contain the data of the variable which is supposed to evolve according to a Homogeneous Markov Chain (HMC) (e.g. in the financial application the variable consists on the sovereign credit ratings, see Section 3.1). As a matter of fact, the first matrix is expected to be named "ratings" by default (see Fig. 4). The second matrix has to refer to the reward process describing the attribute which is driven by the HMC. In the case of the financial application, as illustrated in Section 3.1,

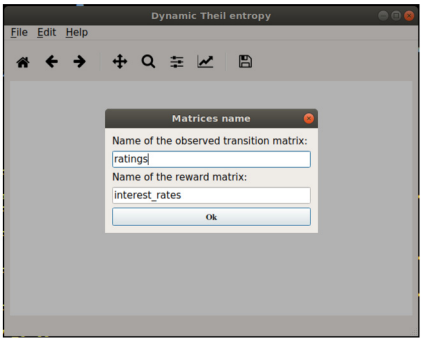


Fig. 4. Dialog to specify the input matrices.

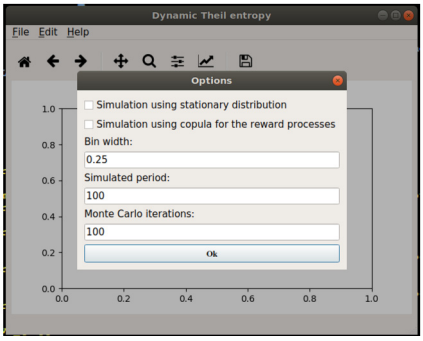


Fig. 5. Dialog to specify the the input parameters related to the Monte Carlo simulation.

this is the credit spread. As the code directly computes the credit spread starting from the interest rates, the second matrix directly collects the interest rate data. Indeed, by default, this matrix within the file is expected to be named "interest_rates" (see Fig. 4).

Once the two matrices have been specified, the user may start the computation: `Edit -> Run`. The use is prompted with a dialog window, reported in Fig. 5, where there is the ability to specify: the bin width to estimate the empirical distributions (one for each ordered variable of the first matrix), the simulated period and the number of Monte Carlo iterations.

Alternatively, the user can flag "Simulation using stationary distribution" to compute the asymptotic values of the Random Theil's Entropy. After clicking the button `OK`, the program will start the computation, and when it finishes, it returns the plot of the Dynamic inequality (Fig. 6) that the user has the ability to interact with and to save as a graphical file (i.e. PNG, PDF, PS, and more).

Subsequently, by clicking on `Edit ->` Plot CS distributions the user can plot the histograms of the empirical distributions of the attribute. Moreover, by clicking on `Edit -> View Transition matrix` the transition probability matrix, estimated on the sequences of visited communities, is shown (Fig. 7).

Finally, with `Edit -> RunChangePoint` one can run the change-point detection algorithm. As described for the CLI, the code runs after the specification of: the number of

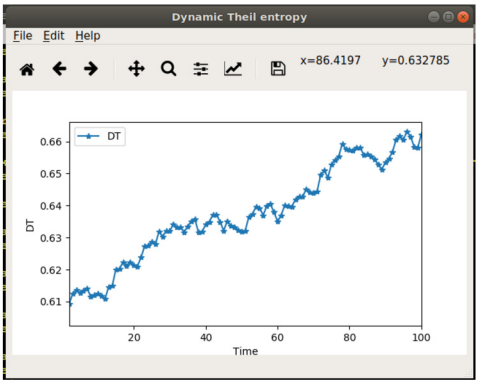
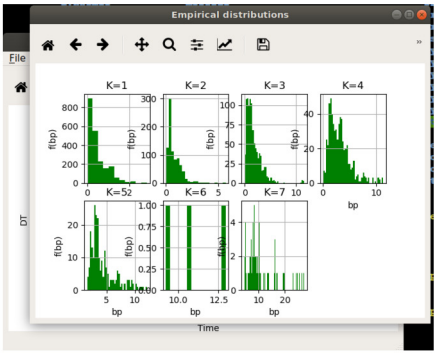


Fig. 6. Output: dynamic inequality.



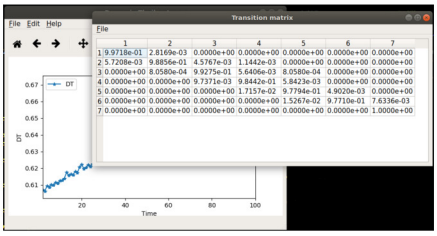
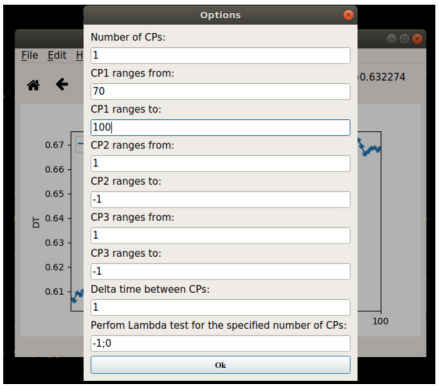


Fig. 7. Histogram of the CS empirical distribution/transition probability matrix.



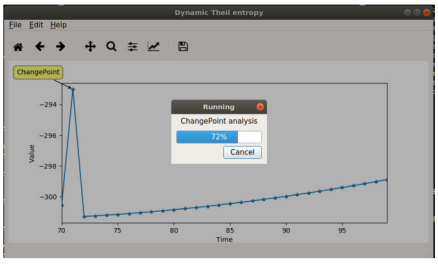


Fig. 8. Options required for change-point detection.

change-points to be detected and the corresponding Λ test (see Fig. 8); the range of time where the algorithm is carried out and, eventually, the distance between two subsequent change-points.

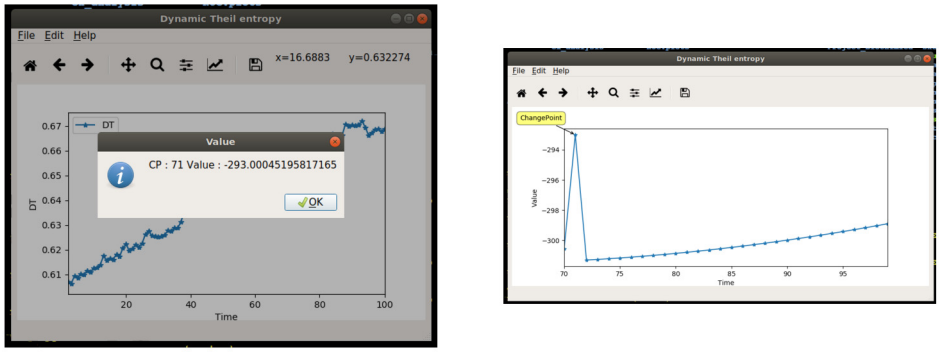


Fig. 9. Output of the change-point detection algorithm.

In the case reported in Fig. 8, a single change-point is detected within a range of time spreading between $t = 70$ and $t = 100$.

After confirming the chosen options, the computation starts and the GUI returns the plot of the likelihood function estimated on the community data (see in Fig. 9), together with the value of the maximum likelihood function, and the corresponding position of the calculated change-point. Evidently, also in this case, the resulting plot can be saved into a standard graphical file format.

4.4. *Testing Financial Inequality in an Economic Area*

Finally, we will show how the described CLIs and GUI can be used to predict the financial inequality in the European Economic Area according to the theoretical model proposed in D'Amico *et al.* (2018a, 2018b). In this specific case, the meta-community coincides with all the countries within the European Community. Thus, each rating class, as assigned by rating agencies, can be seen as a community, in which the countries are allocated at every time step. Clearly, as also already stated in the previous section, the credit spread represents the personal attributes held by each country.

The results we are here reporting have been obtained using the monthly rating, attributed by the Standard & Poor's agency, to the 26 European countries (UK and Cyprus have been excluded in the current meta-community sample) from January 1998 to December 2016 (see, D'Amico *et al.*, 2018a, for extra details on the data-set we are here considering).

To detect the position of a change-point, within the considered horizon time, we compute the maximum value of the likelihood function considered as a function of the position of the change point. Finally, we fix the change point as the value that maximizes the likelihood function. In the proposed software one can use both the **changepoint.py** CLI as well as the GUI:

$$python3 changepoint.py - m./files/sep_m onthly.mat - c1.$$

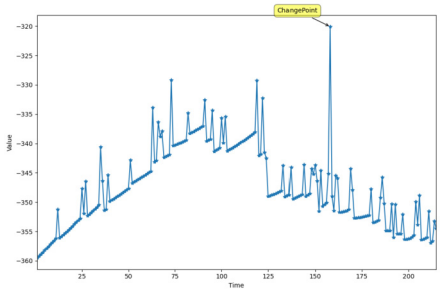


Fig. 10. GUI results for the change-point detection, see text for details.

The result is reported in Fig. 10, where the likelihood function is computed depending on the position of the change point (measured on the X-axis). The software detects a change-point at time 158 (the maximum value of the likelihood function). The value 158 corresponds to a change point detected in January 2012. Indeed, at the beginning of 2012 the value of the total credit spread in Europe had a peak of about 10.000 basis points (bp) and this growth was driven by the rise of the securities yield of Greece (2.924 bp), Ireland (1.245 bp) and Portugal (1.385 bp), see D'Amico *et al.* (2018b) for more detail about the evolution of financial variables. Similarly, for the interested reader, financial examples with multiple change-points can be found in D'Amico *et al.* (2019)

It is relevant to notice that the software (i.e. the **changepoint.py** CLI) also provides an indication related to the choice of the best model as it computes the Bayesian information criterion (BIC) to balance the improvement in the goodness of fit test obtained by increasing the number of the parameters obtained by an increase in the number of change points. Precisely, the BIC is evaluated according to the relation

$$BIC(k) = D \cdot (D-1) \cdot (k+1) \log(n) - 2 \sum_{r=0}^{k} L(\tau_r, \tau_{r+1}).$$

k is the number of change points, n is the size of the sample, D is the cardinality of the state space of the Markov chain, thus, $D \cdot (D-1) \cdot (k+1)$ is the total number of parameters. The quantity $\sum_{r=0}^{k} L(\tau_r, \tau_{r+1})$ is the likelihood function conditional on the estimated change points. The best model can be selected by minimizing the value of the $BIC(k)$ with respect to k. The application of the BIC to our financial data provides interesting practical results that can be summarized as follows. First, we set $k = 1$ and we obtain a value of the change point equal to 158 months. The corresponding log-likelihood function assumes the maximum value of -320.06 and a BIC equal to 1129.10. Second, we set $k = 2$ to understand if two change points better describe our data. In this case, the optimal change points are identified at times 73 and 119. The corresponding log-likelihood function assumes the maximum value of -311.12 which shows an increasing ability of the model with two change points to fit the data but the BIC value increases to 1623.29. Thus, the model with a single change point was found as the most suitable having least BIC value.

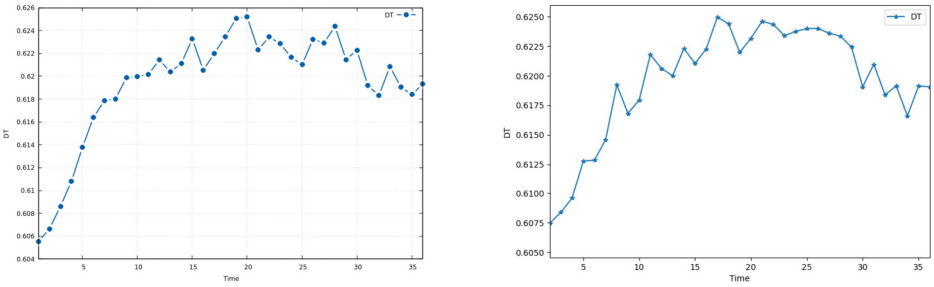


Fig. 11. Random Entropy. Results obtained using the CLI are reported on the left panel, while the ones obtained using the GUI have been reported on the right panel.

Equivalently, a user can forecast the financial inequality in an economic area and its evolution in time via the **randentropy.py** (or the GUI):

$$python3 randentropy.py - m./files/sep_monthly.mat - b./files/sep_monthly.mat - s0.25 - t36 - n1000 - v,$$

where the forecast period has been set to 36 months, using 1000 Monte Carlo simulations. The final result, reported in Fig. 11, shows a similar trend for both the GUI and CLI, with some clear differences related to the implicit randomness of the Monte Carlo procedure (clearly the user can easily avoid this difference selecting a fixed random seed using the – seed option, or equivalently via the `set_use_a_seed` method within the `randentropykernel` class). The results entail a sharp increase in short-term financial inequality, as measured in term of credit spread, which is expected to persist in the first 10 months of the forecast. Then, the rise is expected to be less pronounced until the reaching of its maximum value around month 20. Immediately afterwards, a slight decrease is expected to be observed.

As a final remark, it is somehow important to underline that, evidently, a user has the capability of building its own code, to perform the same or similar computations just described, accessing directly the functionalities implemented within the **randentropymod** Python 3.x module.

5. Conclusions and Perspectives

The **Randentropy** software allows estimating the inequality in a stochastic system according to the framework based on Random Entropy as developed in D'Amico *et al.* (2019). The methodology is able to consider dependent behaviours of the individuals and time-varying dynamics, which may be of interest in several applied domains. Possible developments of the research include the possibility to consider semi-Markov models, as done in the **SemiMarkov** R Package developed by Król and Saint-Pierre (2015), to which a reward scheme based on a copula function should be attached, followed by the evaluation of the Random Entropy according to our software.

Random Entropy evaluation, in the presented general framework, is a new and challenging subject of research and is not available in any software; this renders our investigation an "unicum" in the literature of inequality assessment in stochastic systems.

References

Asadi, M., Zohrevand, Y. (2007). On the dynamic cumulative residual entropy. *Journal of Statistical Planning and Inference*, 137(6), 1931–1941.

Behrendt, S., Dimpfl, T., Peter, F.J., Zimmermann, D.J. (2019). RTransferEntropy—quantifying information flow between different time series using effective transfer entropy. *SoftwareX*, 10, 100265.

Calì, C., Longobardi, M., Navarro, J. (2020). Properties for generalized cumulative past measures of information. *Probability in the Engineering and Informational Sciences*, 34(1), 92–111.

Curiel, R.P., Bishop, S. (2016). A measure of the concentration of rare events. *Scientific Reports*, 6, 32369.

D'Amico, G., Di Biase, G. (2010). Generalized concentration/inequality indices of economic systems evolving in time. *Wseas Transactions on Mathematics*, 9(2), 140–149.

D'Amico, G., Di Biase, G., Manca, R. (2012). Income inequality dynamic measurement of Markov models: application to some European countries. *Economic Modelling*, 29(5), 1598–1602.

D'Amico, G., Di Biase, G., Manca, R. (2014). Decomposition of the population dynamic Theil's entropy and its application to four european countries. *Hitotsubashi Journal of Economics*, 55(2), 229–239.

D'Amico, G., Di Biase, G., Janssen, J., Manca, R. (2017). *Semi-Markov Migration Models for Credit Risk*. John Wiley & Sons.

D'Amico, G., Scocchera, S., Storchi, L. (2018a). Financial risk distribution in European Union. *Physica A: Statistical Mechanics and its Applications*, 505, 252–267.

D'Amico, G., Regnault, P., Scocchera, S., Storchi, L. (2018b). A continuous-time inequality measure applied to financial risk: the case of the European Union. *International Journal of Financial Studies*, 6(3), 62.

D'Amico, G., Petroni, F., Regnault, P., Scocchera, S., Storchi, L. (2019). A Copula-based Markov reward approach to the credit spread in the European Union. *Applied Mathematical Finance*, 26(4), 359–386.

Di Crescenzo, A., Longobardi, M. (2002). Entropy-based measure of uncertainty in past lifetime distributions. *Journal of Applied probability*, 39, 434–440.

Di Crescenzo, A., Longobardi, M. (2009). On cumulative entropies. *Journal of Statistical Planning and Inference*, 139(12), 4072–4087.

Dubois, P.F., Hinsen, K., Hugunin, J. (1996). Numerical python. *Computers in Physics*, 10(3), 262–267.

Durante, F., Sempi, C. (2016). *Principles of Copula Theory*, Vol. 474. CRC Press, Boca Raton, FL.

Ferguson, N., Datta, S., Brock, G. (2012). msSurv: an R package for nonparametric estimation of multistate models. *Journal of Statistical Software*, 50(14), 1–24.

Hunter, J.D. (2007). Matplotlib: a 2D graphics environment. *Computing in Science & Engineering*, 9(3), 90–95.

Jackson, C.H., (2011). Multi-state models for panel data: the msm package for R. *Journal of Statistical Software*, 38(8), 1–29.

Jaynes, E.T. (1957). Information theory and statistical mechanics. *Physical Review*, 106(4), 620.

Jones, E., Oliphant, T., Peterson, P. (2001). SciPy: Open source scientific tools for Python. [Online; accessed 2019-02-05]. http://www.scipy.org.

Król, A., Saint-Pierre, P. (2015). SemiMarkov: an R package for parametric estimation in multi-state semi-Markov models. *Journal of Statistical Software*, 66(6).

Krumme, C., Llorente, A., Cebrian, M., Moro, E. (2013). The predictability of consumer visitation patterns. *Scientific Reports*, 3, 1645.

Kullback, S., Leibler, R.A. (1951). On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1), 79–86.

Marcon, E., Hérault, B. (2015a). entropart: an R package to measure and partition diversity. *Journal of Statistical Software*, 67(1), 1–26.

Marcon, E., Hérault, B. (2015b). entropart: an R package to measure and partition diversity. *Journal of Statistical Software*, 67(1), 1–26.

Phillips, S.J., Anderson, R.P., Schapire, R.E. (2006). Maximum entropy modeling of species geographic distributions. *Ecological Modelling*, 190(3-4), 231–259.

Polansky, A.M. (2007). Detecting change-points in Markov chains. *Computational Statistics & Data Analysis*, 51(12), 6013–6026.

PyQT (2012). PyQt Reference Guide. http://www.riverbankcomputing.com/static/Docs/PyQt4/html/index.html.

Rao, M., Chen, Y., Vemuri, B.C., Wang, F. (2004). Cumulative residual entropy: a new measure of information. *IEEE Transactions on Information Theory*, 50(6), 1220–1228.

Saad, T., Ruai, G. (2019). PyMaxEnt: a Python software for maximum entropy moment reconstruction. *SoftwareX*, 10, 100353.

Shannon, C.E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27(3), 379–423.

Song, L., Kotz, D., Jain, R., He, X. (2006). Evaluating next-cell predictors with extensive Wi-Fi mobility data. *IEEE Transactions on Mobile Computing*, 5(12), 1633–1649.

Storchi, L. (2020). MarkovTheil code. GitHub.

Summerfield, M. (2007). *Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming (paperback)*. Pearson Education.

Theil, H. (1967). *Economics and information theory*. Technical report.

Trueck, S., Rachev, S.T. (2009). *Rating Based Modeling of Credit Risk: Theory and Application of Migration Matrices*. Academic Press.

G. D'Amico is a full professor of mathematical methods in economics, finance and insurance at the Department of Economics of the "G. D'Annunzio" University of Chieti-Pescara. He received his PhD in mathematics for applications in economics, finance and insurance from the University "La Sapienza" of Rome in May 2005. His research interests include the theory of stochastic processes and their applications in finance, insurance, economics, reliability and wind energy. He is interested also in nonparametric statistical inference for stochastic processes. His research has appeared in several refereed journals such as *European Journal of Operational Research*, *Applied Mathematical Finance*, *Scandinavian Actuarial Journal*, *Applied Mathematical Modelling*, *IMA Journal of Management Mathematics*, *Journal of the Operational Research Society*, *Reliability Engineering and System Safety*, *Stochastics, Insurance: Mathematics and Economics*. He has published a book with John Wiley and Sons.

S. Scocchera works in the Credit Risk Model office at Banco BPM SPA, dealing with projects concerning the Credit Portfolio Model, the inclusion of the climate risk (ESG) within risk parameters and satellite models. She received her PhD in accounting, management and finance with specialization in mathematical finance from the "G. D'Annunzio" University of Chieti-Pescara in May 2019.

L. Storchi is an associate professor and after more than 15 years of research activity he has acquired wide competences in several programming languages, numerical methods and data modelling. His multidisciplinary background is reflected both in the list of his scientific interests, as well as in the diversity of his publications.