Towards a Generic Theoretical Framework for Pattern-Based LUCC Modeling

François-Rémi Mazy, Pierre-Yves Longaretti

Highlights

An Accurate and Powerful Calibration-Estimation Method Based on Kernel Density Estimation

François-Rémi Mazy,Pierre-Yves Longaretti

- Accurate calibration-estimation methods for LUCC models are needed to ensure their reliability.
- A calibration-estimation method based on Kernel Density Estimation has been developed (Bayes-eKDE). Our method is substantially more accurate and efficient compared to existing ones.
- The method is implemented in our own software CLUMPY (Comprehensive Land Use Model in Python).
- Comparison and validation of calibration-estimation methods must be performed in explanatory variable space, not from map comparison.

Towards a Generic Theoretical Framework for Pattern-Based LUCC Modeling

An Accurate and Powerful Calibration-Estimation Method Based on Kernel Density Estimation

François-Rémi Mazy[a,*], Pierre-Yves Longaretti[a,b]

[a] *Université Grenoble Alpes, CNRS, Inria, Grenoble INP, LJK, 38000 Grenoble, France*
[b] *Université Grenoble Alpes, CNRS-INSU, IPAG, CS 40700, 38052 Grenoble, France*

ARTICLE INFO

Keywords:
Land Use Change, Land Cover Change, LUCC, Model Development, Model Evaluation, Model Accuracy, Calibration, Probability, Transition, Kernel Density Estimation.

ABSTRACT

Several modeling strategies have been proposed to study Land Use and Land Cover changes (LUCC). However, substantial discrepancies have been noted between different models for the same problem, questioning their overall reliability and reproducibility. To address this challenge, we elaborate a generic, formally correct, theoretical framework for pattern-based LUCC modeling, which is implemented in our own software, CLUMPY (Comprehensive Land Use [and cover] Modeling in PYthon).

The present work focuses on calibration. We devise a kernel density calibration-estimation method (Bayes-eKDE) that is shown on synthetic artificial data to be both accurate and algorithmically efficient. We also introduce a generic evaluation method that allows us to compare the calibration efficiency of existing models. The gain in precision and computational time of our calibration method is precisely quantified in this way.

1. Introduction

Land Use and Land Cover Change (LUCC) models[1] are now widely used in environmental science in a variety of contexts to analyze a large array of issues (Agarwal et al., 2000; Verburg et al., 2006; Bielecka, 2020). In the recent years, a number of studies have undertaken a comparison of model results across a wide range of modeling environment types. These studies have displayed substantial discrepancies in the results, due to differences in input data, modeling strategies or scenario assumptions (Mas et al., 2014, 2022; Prestele et al., 2016; Alexander et al., 2017; García-Álvarez et al., 2022). These findings question the robustness and soundness of the conclusions drawn from such models (Verburg et al., 2019).

We focus here on pattern-based LUCC models (see, e.g., Camacho Olmedo et al. 2018 for a convenient typology and description of various LUCC modeling framework types). This is a widespread approach, implemented in a number of existing software, based on maps in raster format. Among the most common such software, one can mention the CLUE family (Verburg et al., 1999), the latest member being CLUMondo (van Asselen and Verburg, 2013; van Vliet et al., 2015), Dinamica EGO (Soares-Filho et al., 2002), or Idrisi LCM[2] (Eastman et al., 1995). Despite the numerous modeling environments of this type that have been developed over the years, none of their constituent steps and methodological choices are clearly formalized

*Corresponding author
francois-remi.mazy@inria.fr, +3 368-017-2217 (F. Mazy); pierre-yves.longaretti@inria.fr (P. Longaretti)
ORCID(s): 0000-0001-8405-0141 (F. Mazy); 0000-0002-4940-0756 (P. Longaretti)

[1] In this paper, *modeling environment* refers to any of the available software that relies on software-specific but application-generic modeling choices, allowing users to set up an implementation (instantiation) of the modeling environment in specific case studies. The term *model* refers either to the *modeling environment* (for the software) or *modeling framework* (for the concepts used) thus defined, or to an actual implementation (instantiation) of the software for a particular problem and context. *Theory* refers more specifically to the mathematical and conceptual apparatus that have been used in a specific modeling framework and implemented in its related modeling environment, but other types of theories are useful or even necessary in LUCC modeling, most prominently from social and environmental sciences.

[2] For the purposes of this study, we make use of Idrisi Selva, a somewhat old version, but for which we do have a licence. LCM is now part of TerrSet, the new software of Clarks Lab.

and exhaustively described in the literature. Furthermore, the calibration and validation methods adopted differ widely from one model to another (van Vliet et al., 2016; Noszczyk, 2019) making it difficult to perform systematic formal and algorithmic comparisons between models. Moreover, some methods and results of other fields could be more extensively used, such as the one described in this work, which is borrowed from the machine learning toolbox.

For these pattern-based LUCC models, substantial differences of behavior between the various modeling environments mentioned above are observed even for the same set of data, explanatory variables selection and scenario. Such differences necessarily come for the most part from two major types of choices: the quantification of the correlation between predictors (explanatory variables) and actual patterns of change (e.g., through logistic regression, weights of evidence, neural networks...) performed in model calibration on the one hand, and the implementation of allocation performed in future projection of LUCC on the other hand. This is true even in controlled settings using artificial synthetic data (as in the analysis of Mas et al. 2014).

The calibration process, responsible for the first type of discrepancies, is the object of the present work. The second type is related to the allocation process and will be addressed in a dedicated paper. The main object of the series of papers initiated here is to analyze errors in modeling choices or in algorithm implementation (or under-optimal choices and implementations) which underlie most of the discrepancies in behavior of the various models. As such, we attempt to provide a firm theoretical basis for the mathematical and algorithmic foundations of pattern-based LUCC modeling. Such an approach should eventually replace the intuitive and often imprecise rules that have been used in the elaboration of existing modeling environments. Mazy and Longaretti (2022a) present an overview of the whole analysis, with an emphasis on the calibration process, and can be used as in introduction to our work.

This article is organized as follows. We start with a methodological discussion explaining in more detail the rationale, relevance and strategy of our approach. Indeed, the work done in the present paper is part of a more global line of research whose objectives and methods are most likely difficult to grasp for most LUCC software users and possible for a number of LUCC software developers as well. We therefore clarify these points, as they are not transparent in the technical work presented in the rest of the paper, and may lead to a misunderstanding of this work relevance (section 2). The following section is devoted to a description of the generic architecture of pattern-based LUCC modeling environments (section 3). This section also formulates the problem of transition probability estimation and introduces our notations. We then present our method of transition probability estimation (section 4). The implementation of our method called Bayes-eKDE is detailed next (section 5). Section 6 describes the strategy we have adopted to evaluate and compare the performances of our method along with various existing modeling software. Section 7 presents our controlled setting based in synthetic data and assumed transition probability distributions; actual comparisons with other modeling environment calibration methods are discussed there as well. The last section summarizes the main results and concludes the paper (section 8).

2. Methodological choices and research strategy relevance

Although the intended audience of this paper is LUCC software designers and developers, and more generally anyone interested in the formal foundations of quantitative LUCC modeling, the primary target of this section is LUCC software users. These potential readers are not expected to be interested in the technical details of our work, but may still be interested in understanding its nature and purposes. This may also be true of some software developers. Some of the points made here will by necessity be familiar to some readers.

The starting point of our questioning is the following. If a pattern-based LUCC software fails to some extent to produce expected results (e.g., observed spatial patterns of LUC changes), is this because of a defect in the software conception, of lack of quality of the data used, of incorrect user choices in the problem set-up (such as an incorrect choice of explanatory variables), of intrinsic characteristics of the problem, or a combination of these?

To address this question, it is important to keep in mind that the common validation methods of LUCC software based on various measures of spatial accuracy in the prediction of LUC state changes must be used with caution. Even for perfect data, perfectly known explanatory variables and a perfect software from a

calibration and allocation point of view (which is never the case in practice), spatial accuracy can only be expected if the transition probabilities are very inhomogeneous and the fractional amounts of LUCC state change important, i.e., if the probabilistic nature of the problem is not too far from being deterministic. For an extreme but unrealistic context where probabilities of LUC state change are completely spatially homogeneous, actual state changes are completely random. In reality, an actual case study will in effect occupy some middle ground between total randomness and exact predictability. Improving the level of predictability of LUC changes is in fact a major goal of land-use theoretical analyses (such analyses are rooted, e.g., in social and environmental sciences). But this level of predictability is what ultimately limits the level of spatial prediction accuracy of LUCC modeling.

Because of this, in the end, some "reasonable" spatial accuracy will be obtained if: i/ the chosen LUCC modeling framework implemented in a given software is correct; ii/ the problem is well enough understood and the underlying causes of change are known to a sufficient level, so that explanatory variables can be chosen wisely to capture effectively these underlying causes[3]; iii/ the quality of the data is not a serious issue. A practical consequence of this is that one must be able to judge a modeling framework correctness independently of the spatial accuracy of LUC patterns change "prediction", of explanatory variable relevance and of data quality. This is precisely one of the main objectives of our line of research.

To make progress on this issue, we break the problem up into several parts, in order to analyze if existing software are correctly designed. To this effect, a common practice in other research fields is first to examine critically the mathematical apparatus that was used to set up the modeling framework implemented in the software, to check whether this algorithmic implementation itself conforms to its intended purpose, and finally to check if the software performs as expected in problems where the other sources of concern (here, data quality and user potential set-up errors) are eliminated. In principle these conditions should be satisfied by all software. In practice, we found that none of the software we scrutinized does, to varying degrees. Furthermore, this procedure must be applied to the various pieces making up a LUCC software, because each is rather complex in itself and can be examined largely independently from the others. These pieces usually consist in an explanatory variable selection module, a calibration module (to extract relevant information from the data on these explanatory variables and their relation to past observed patterns of LUC changes), and an allocation module (to produce actual LUC changes based on this extracted information), although some software (e.g., the CLUE family) have a simpler structure for various reasons.

The present paper deals with calibration issues only. The way we address this presents at least two potentially paradoxical features from a LUCC software user point of view: we focus only on pixel LUC state changes and we focus on explanatory variable space and not on physical space. The first is paradoxical because LUC changes almost exclusively occur in bunches of contiguous pixels (patches), and not pixel by pixel, independently of each other; the second is paradoxical precisely because some level of spatial location accuracy is an important concern for a LUCC software user. Let us address these two points in turn.

In relation to the first point, note that other software's calibration procedures also focus on various forms of pixel statistical properties. But, by design, these pixel probability distributions carry no information on patch transition as they carry no information on pixel correlations. How then can this calibration be at all relevant to actual case studies? The answer is related to assumptions that are rarely if ever mentioned by software developers, possibly because the necessity of these assumptions has not been recognized.

Let us discuss this on the example of Dinamica EGO, whose modeling framework is possibly the most explicit and sophisticated and about this point among existing pattern-based LUCC modeling environments. In Dinamica EGO, patches are produced from two basic "bricks": a seed pixel (i.e., a pixel around which a patch undergoing a change of LUC state is formed), and patch characteristics. Dinamica EGO implicitly assumes that, to a very high level of approximation, seed pixels are statistically independent of each other[4], a property which is obviously not true for all pixels (there is a high chance that neighboring pixels are in the same LUC state or change state to the same LUC state, for example). Similarly, patch characteristics are assumed to be highly statistically independent of seed pixel ones[5]. And finally, individual seed pixel

[3]Explanatory variables need not be truly explanatory in nature, but only efficiently correlated to actual patterns of change. However and quite clearly, the closer to actual causal processes they can be (intrinsically or through proxies), the better the resulting correlation.

[4]This point has never been proved, but can be checked on a case-by-case basis.

[5]Same comment.

probability distributions are assumed to be the same (or nearly the same) as all individual pixels probability distributions for any given LUC state change[6]. These assumptions are also made in our own approach to pattern-based LUCC modeling. We will discuss elsewhere why they are reasonably well-satisfied in LUCC case studies. For the time being, we point out that the conjunction of these assumptions justifies, for Dinamica EGO and for our own software (CLUMPY), that their respective calibration procedures focus on characterizing the probability distributions of pixels, while ignoring pixel-to-pixel correlations, i.e., ignoring the fact that transitions occur in patches (patch formation is addressed in the allocation module). For other software, the assumptions are different, and in fact incorrect to varying extents (this point is shown in our article on allocation, Mazy and Longaretti 2022b), but these software nevertheless focus on calibrating some types of pixel probability distribution on such a basis.

As for the second paradox, note that the probability distributions to be characterized, which depend on the modeling environment, are always probability distributions in explanatory space. This follows because explanatory variables are chosen to identify which pixels are more likely to undergo a transition. In explanatory space, and because pixels correlations are ignored in the probability distributions of interest as discussed in the previous paragraph[7], the fact that pixels are contiguous or not in physical space is immaterial. Contiguity in patch transition is recovered when forming actual patches, while the spatial location of the patches is controlled by the pixel probability distribution that have been calibrated, through the spatial dependence of the chosen explanatory variables. Observed patterns of change are then recovered from these features, inasmuch as explanatory variables are accurate predictors of these patterns, as pointed out earlier.

We now return to the point made above, i.e., that the relevance and accuracy of calibration procedures (the focus of the present paper) must be evaluated independently of the spatial accuracy of the prediction of patterns of change, of explanatory variable relevance and of data quality. This is achieved through a procedure whose relevance may also not be immediately apparent to a LUCC software user, i.e., by setting up problems in explanatory variable space in which we know beforehand what the answer should be, and testing how our method (and others) perform to recover this information, which is the only known method to achieve this purpose. In practice, we completely specify the relevant pixel probability distributions, allocate changes on the basis of this information in a (demonstrably) statistically correct way, and estimate how accurately these probability distributions are recovered by a number of calibration procedures (including ours) through specific criteria that we have elaborated to this effect (the procedure is detailed in section 6.1). The probability distributions chosen for this exercise are not designed to be related to clear and understandable features in physical space, but to test calibration procedures in a rather demanding way. More complete or realistic tests can always be elaborated, but this evaluation of calibration accuracy should be sufficient for its intended purpose, in our experience.

3. Problem Formulation

We define here in a precise and general way the problem of modeling spatially explicit LUCC. Thus, we introduce a generic architecture of such a model (section 3.1) and notations (section 3.2). We also frame the specific problem of transition probability estimation (section 3.3) that is addressed in this paper as well as the properties of the different quantities involved (section 3.4). This probability distribution estimation is non-parametric. This is mandatory for elaborating a precise calibration procedure.

3.1. Summary of Pattern-Based LUCC Model Architecture

Generally speaking, pattern-based LUCC modeling environments display similar architectures in order to produce the two major types of maps required in LUCC analyses and projections: transition probability maps and allocation maps during simulations of future LUC states. This architecture is organized around a number of modules. In the present work, we adopt a structure in three modules: calibration, estimation

[6]Same comment. Note also that, as pointed out above, the seed pixel correlations are totally different from the correlation properties of the total pixel population. However this information is not encoded in individual pixel probability distributions, which is precisely what allows the seed pixels and total pixel individual probability distributions to be identical to a high degree of precision.

[7]Other types of probability distributions do carry information on correlations between contiguous pixels probability of change, but the three necessary properties stated above for our calibration framework to be relevant provide a simpler way to tackle patch transitions from a mathematical and algorithmic point of view than characterizing correlations in a statistical way.

and allocation (Fig. 1), whose purpose is described below. In existing modeling environments, calibration and estimation constitute a single module, although dynamic explanatory variable maps are sometimes recomputed at each time step (e.g., in Dinamica EGO), implying the existence of a (re)estimation procedure. Calibration and estimation are formally distinguished here because of the specific characteristics of our procedure and a precise definition of these two processes will be given in section 3.2. If some explanatory variable maps evolve and need to be updated, this is also performed at the estimation stage.

All modeling environments usually rely on land use and cover (LUC) raster maps of the study area at some initial dates[8] t_0 and t_1, usually at least a few years apart, where the area of interest is fully spatially characterized in terms of LUC states. The spatial resolution of these maps is quite variable (a few meters to kilometers). The LUC typology of these initial maps is defined by the user from available data, and adapted to the user's specific needs on a case-by-case basis. This typology collects all LUC states observed at t_0 and t_1 as well as the ones of interest a simulation of future changes (which may or may not be all present at these initial dates).

Several maps of characteristic quantities, called explanatory variables, are also used. Explanatory variables are identified by the modeler prior to the elaboration of a LUCC model. These quantities are selected on the basis of their correlations with observed changes of LUC states between t_0 and t_1. Some of these explanatory variable maps may be readily available, e.g., the elevation above sea level or slope, or they can be relative to specific characteristics of the LUC maps, e.g., the distance to urban areas, in which case they may evolve along with changes of LUC states. The issue of explanatory variable selection is not addressed in this work; they are assumed to be an input of the method. A new selection method based on the joint analysis of variables relevance and redundancy is proposed in a companion paper (Longaretti and Mazy, 2022).

These three elements (two initial LUC maps and a set of explanatory variable maps) constitute the inputs of the model calibration-estimation process, which aims at producing the probability distributions of explanatory variables, conditional to specific initial (at t_0) and final (at t_1) LUC states. These distributions are not specified in physical space, but in explanatory variable space. Spatial probability distribution maps follow because every pixel in a map is characterized by specific values of the various explanatory variables.

The allocation process explicitly assigns new LUC states at various future dates for all pixels in the study area. This allocation process is initialized from a LUC map at time t_s, usually equal to t_1 at the start of a simulation[9]. The explanatory variable maps are also required at time t_s as well as an overall transition matrix that defines the total areas that are expected to change LUC state during the simulation time steps dt, for all possible state changes; such overall changes are usually specified through scenarios. This produces an allocated map at time $t_s + dt$, which serves as a new LUC map when the allocation process is iterated in a simulation. Time steps dt are usually of a few years.

We distinguish two types of modules: the first one for calibration and estimation and the second one for allocation. The first two are grouped together because they are conceptually related, and constitute the object of the present work. The allocation process itself will be examined in a forthcoming paper (Mazy and Longaretti, 2022b).

In some case studies, it is useful to divide the study area into a complete set of non-overlapping spatial regions. In general, the LUC architecture described above is then applied independently in each region. However, such an approach can lead to some spatial discontinuities of transition probabilities at the interfaces between regions. Correcting this potential problem is possible but this question is not addressed in this work.

3.2. Definitions and Notations

Our objective requires some mathematical rigor. To this effect, we need to introduce a number of definitions and notations for the quantities that generically appear in pattern-based LUCC modeling. Some of these have already been used in other modeling environments, in particular Dinamica EGO, but we aim here at a higher level of precision and rigor, and consequently, our set of notations is substantially more extensive than usual. These notations are summarized in Table 1.

[8] Some modeling environments, such as CLUMondo, rely on a single map, but must introduce further assumptions to characterize LUC changes from this single map. This strategy is much less precise but unavoidable when data are scarce.

[9] Unless an additional data map at t_2 has been used for validation, in which case the simulation often starts at t_2.

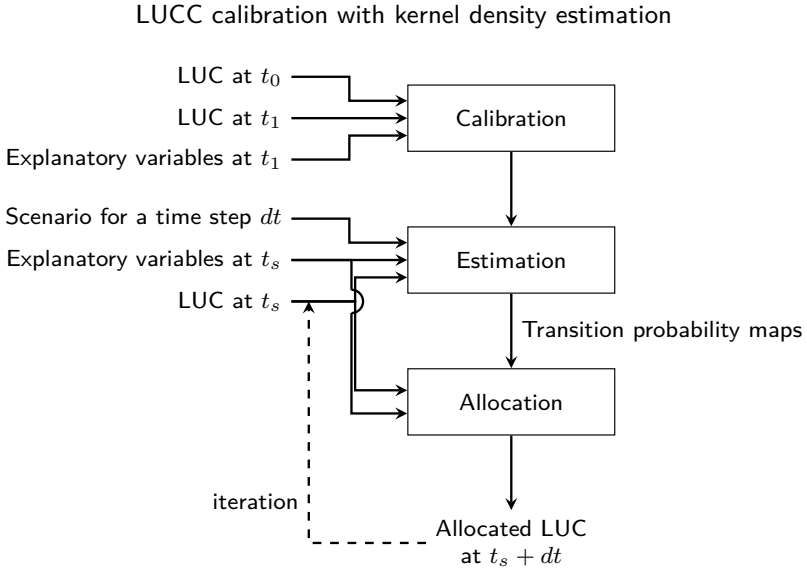


Figure 1: General architecture of our spatially explicit LUC modeling strategy.

In practice, our first task is to evaluate the probability of change to all possible LUC states, for each pixel[10]. Without loss of generality, we can single out a specific initial state for this objective and focus on pixels characterized by this initial state. The process of transition probability characterization must then be repeated for all possible initial states.

Land use and cover (LUC) states Each LUC state is represented by a positive natural number (the null value is used for unknown state or pixels outside the study area, a common convention in GIS environments). The notation u is used for a pixel state at t_0 (resp. t_s) and v for this pixel state at t_1.

Raster quantities As stated in section 3.1, we use two maps at some initial dates t_0 and t_1 for calibration purposes. Also, because the maps at t_0 (initial calibration map) and t_s (map at some simulation step s) are different, the pixels in both maps for a given initial state u have no reason to be identical. Because of this we introduce specific notations for these relevant sets of pixels associated to the same initial state u at different times.

More precisely, let n_{I_v} be the number of pixels undergoing a change from state u to state v during the period $t_0 \to t_1$, and let I_v be the set of indices of these pixels[11] $\{1, \cdots, n_{I_v}\}$. Each pixel is then identified by its index i. We use a superscript i to associate any quantity of interest to a particular pixel i. For example, the spatial coordinates of pixel $i \in I_v$ are denoted $\boldsymbol{x}^i = (x_0^i, x_1^i)$.

The second set of pixels associated with the same initial state u refers to $t = t_s$. Let n_J be the number of pixels in state u and J be the set of all pixel indices $\{1, \cdots, n_J\}$. Each pixel is identified by its index j which can be used as a superscript.

Explanatory variables Explanatory (or ancillary, or predictor or independent[12]) variables are quantities defined on pixels, and considered relevant for the statistical prediction of LUC changes, due to their correlation with past observed changes. Also, some explanatory variable are dynamic, and their maps need to be updated at each time step in the course of a simulation. For example the shortest distance of pixels to the limits of urban areas might be used as an explanatory variable in a simulation of urban sprawl and such a distance will evolve in the course of such a simulation.

[10] Although actual LUCC occur in patches of contiguous pixels, characterizing single pixel transition probabilities is a required first step. The precise connection between pixel and patch transitions will be examined in another paper in this series.

[11] This implies the existence of an ordering for this subset of all pixels. This ordering can be chosen arbitrarily. For simplicity and by slight abuse of language, pixels are identified to their index in what follows.

[12] The term "independent variable" is sometimes used, in opposition to dependent variables (e.g., probability densities) which are functions of independent variables. This does not imply that explanatory variables are statistically independent.

Let d be the number of explanatory variables. For future use, we define the explanatory variable space either as a point space ($\mathcal{D}$) or as a vector space of dimension d ($\mathbb{R}^d$); by construction, $\mathcal{D} \subset \mathbb{R}^d$. The explanatory variables associated to individual pixels define either specific points or vectors in this space.

More precisely, an explanatory variable can have an infinite, finite or semi-finite range of values. When the range is limited in this way, the explanatory variables are viewed as points, or as subsets of the vector space, but do not constitute a sub-vector space. For example, the distance to an urban area is semi-finite because all possible values are ≥ 0. Let $\mathcal{D}_k \subset \mathbb{R}$ be the kth explanatory variable's domain. In the example just mentioned, it would be equal to $[0, +\infty[$. By construction, $\mathcal{D} = \mathcal{D}_1 \times \cdots \times \mathcal{D}_d$.

Calibration and estimation The notions of calibration and estimation have been introduced in section 3.1 but not defined. We therefore start by providing a generic definition of these two terms, and will illustrate their specific meaning for the present procedure at various points in this work. *Calibration* is a process collecting relevant calibration explanatory variable values and their associated pixels; the meaning of *relevant* in this sentence will be made clear section 3.3. Various preliminary transformations of these explanatory variables that are necessary for estimation are also performed in the calibration module (in particular, the whitening transformation that will be described in section 4.3). Note that because some explanatory variables are dynamic (i.e., evolve during the course of a LUCC simulation), the relevant explanatory variable values at the calibration times t_0 and t_1 are the ones stored for the estimation process.

The final objective of the calibration-estimation procedure is to produce maps of relevant probability distributions, in particular maps of various conditional probability distributions of explanatory variables that will be introduced in section 3.3. *Estimation* is the process providing these maps from the calibration data. Because some explanatory variables are dynamic, as just pointed out, these maps must be updated at each simulation time step. Such maps can also in principle be produced at the calibration step, but are actually not needed then. This is why in Fig. 1 estimation is performed at each simulation time step.

Two different notations are used for explanatory variables: they are designated by $\boldsymbol{y}$ when the corresponding pixels belong to the calibration set $\boldsymbol{Y}$ (pixels in LUC state belonging to J) or $\boldsymbol{Y}_v$ (pixels in LUC belonging to I_v), and by $\boldsymbol{z}$ when the corresponding pixels belong to the estimation set $\boldsymbol{Z}$.

More precisely, $\boldsymbol{y}$ (resp. $\boldsymbol{z}$) is a column vector collecting the d values y_k (resp. z_k) ($k \in \{1, \ldots, d\}$). When used in probability distributions, $\boldsymbol{y}$ (resp. $\boldsymbol{z}$) stands instead for the d-tuple collecting all y_ks (resp. z_ks) instead of the column vector, and constitutes in this view a point in explanatory space. To alleviate notations, we do not use different symbols when referring to vectors or d-tuples or points; the difference of meaning should be obvious from the context.

In practice, the set of calibration pixels associated to the calibration explanatory variable values is either the set[13] I_v or the set J. For any such pixel the associated vector of explanatory variables is denoted $\boldsymbol{y}^i$ while y_k^i refers to the kth explanatory variable value for this pixel. A map of the kth explanatory variable in the calibration phase is obtained for the whole collection of y_k^i values from the related pixel spatial locations $\boldsymbol{x}^i$. Finally, we define $\boldsymbol{Y}$ (resp. $\boldsymbol{Y}_v$) as the n_J-tuple (resp. n_{I_v}-tuple) of d-tuples $\boldsymbol{y}$. Such a collection accounts for multiple appearances of a given $\boldsymbol{y}$ value (i.e., a given d-tuple $\boldsymbol{y}$ may appear a number of times in $\boldsymbol{Y}$).

In a similar way, the set of pixels associated to the estimation explanatory variable values is always the set J. For a given pixel $j \in J$, $\boldsymbol{z}^j$, z_k^j and the n_J-tuple $\boldsymbol{Z}$ of $\boldsymbol{z}$ values taken by pixels in J are defined in the same way as in the previous paragraph. Note that by construction, $\boldsymbol{Z} = \boldsymbol{Y}$: for the relevant explanatory variables, the estimation and calibration explanatory variable values are identical.

It is customary in the LUCC modeling literature to bin continuous variables so that continuous and discrete variables can be placed on the same footing. However, although calibration is apparently simpler with binned

[13] This implies that some quantities are calibrated (i.e., selected and transformed) and estimated at t_s, while some other are calibrated at t_0 and t_1 and estimated at t_s. The rationale for this will become apparent in section 3.3.

element	set	definition
u, v	V	LUC state at t_0 and t_1
i	I_v (size n_{I_v})	index of pixels of initial state u at t_0 and final state v at t_1
j	J (size n_J)	index of pixels of state u at t_s
y_k, y_k^i	$\mathcal{D}_k$	kth calibration explanatory variable value taken by pixels in I_v
$\boldsymbol{y}$, $\boldsymbol{y}^i$	$\mathcal{D}$	d-tuple (or column vector or point) of calibration explanatory variables for pixels in I_v
$\boldsymbol{Y}$	not defined	n_J-tuples of values of $\boldsymbol{y}$ taken by pixels in J
$\boldsymbol{Y}_v$	not defined	n_{I_v}-tuples of values of $\boldsymbol{y}$ taken by pixels in I_v
z_k, z_k^i, z_k^j	$\mathcal{D}_k$	kth estimation explanatory variable value for pixels in J
$\boldsymbol{z}$, $\boldsymbol{z}^i$, $\boldsymbol{z}^j$	$\mathcal{D}$	d-tuple (or column vector or point) of estimation explanatory variables for pixels in J
$\boldsymbol{Z}$ $(=\boldsymbol{Y})$	not defined	n_J-tuples of values of $\boldsymbol{z}$ taken by pixels in J

Table 1
Summary of notations and definitions. All quantities pertain to the same given (and arbitrarily chosen) initial state u.

variables, binning poses its own problems, and we avoid this altogether in the present work. As will be shown in the section 7, this also leads to a more powerful and accurate calibration strategy (Bayes rule assisted by efficient Kernel Density Estimation or Bayes-eKDE in short).

In this work we mostly ignore the possible existence of discrete explanatory variables. This follows because discrete variables do not pose any particular calibration problems (the standard pixel counting method is well-defined to estimate all distributions with respect to such categorical data). Appendix C shows how a mix of discrete and continuous variables is dealt with in the framework of our Bayes-eKDE method; in effect, the problem is reduced to a finite number of purely continuous calibrations. For this reason, the remainder of the paper focuses on continuous calibration while ignoring discrete variables, to alleviate notations. Our software (CLUMPY) is set up to deal with both discrete and continuous variables along the lines described in this Appendix.

3.3. Transition Probability Estimation and Bayes' Rule

The transition probability from u to v during a time step $dt = t_{s+1} - t_s$ for a pixel characterized by the d-tuple of explanatory variables $\boldsymbol{z}$ is noted $P(v|u, \boldsymbol{z})$. This notation indicates that this probability is conditional to the knowledge of u and $\boldsymbol{z}$. Bayes' rule allows us to evaluate this quantity from three other probabilities or probability densities[14]:

$$P(v|u, \boldsymbol{z}) = P^*(v|u)\frac{\rho(\boldsymbol{z}|u, v)}{\rho(\boldsymbol{z}|u)}, \tag{1}$$

where $P^*(v|u)$ is the targeted global change probability (the fraction of pixels undergoing such a change). It can be extracted at the start of the simulation from the calibration maps at t_0 and t_1, or, more commonly, it can be specified by a scenario of LUC evolution that one wishes to implement in a LUCC simulation. Whatever method is used, it is assumed in the present analysis that $P^*(v|u)$ is given, and the * superscript is used to highlight the fact that this quantity represents an input by the user, not a quantity to be evaluated in the calibration-estimation process.

Because we make use of continuous variables, Bayes' rule involves probability densities, and not only probabilities: $\rho(\boldsymbol{z}|u)$ is the probability density of $\boldsymbol{z}$ for pixels of initial state u, and $\rho(\boldsymbol{z}|u, v)$ is the probability density of $\boldsymbol{z}$ for pixels undergoing a state change from u to v. Probabilities themselves are non zero only on a given interval of $\boldsymbol{z}$. For example, $P(\boldsymbol{z}_1 < \boldsymbol{z} < \boldsymbol{z}_2|u) = \int_{\boldsymbol{z}_1}^{\boldsymbol{z}_2} \rho(\boldsymbol{z}|u)d\boldsymbol{z}$, where the convention $\boldsymbol{z}_1 < \boldsymbol{z}_2$ has been used when $z_{k,1} < z_{k,2}$ for all explanatory variables k. When this "interval" in $\boldsymbol{z}$ is small, the probabilities usually become proportional to the corresponding small volume in $\boldsymbol{z}$ space. We conventionally use the

[14]Strictly speaking, $\rho(\boldsymbol{z}|u, v)$ and $\rho(\boldsymbol{z}|u)$ are defined at some specific time t, while $P(v|u, \boldsymbol{z})$ and $P^*(v|u)$ are defined for this time and a time interval dt. Consequently, the probability densities ρ should also conditionally depend on t and the probabilities P on t and dt in this expression. Furthermore, the calibration explanatory variable values of $\rho(\boldsymbol{z}|u)$ depend on both t_0 and t_1. These extra dependencies are dropped to alleviate notations.

letter ρ for probability densities and the uppercase letter P for probabilities. A more formal definition of probability densities is provided in Appendix B.

The use of Bayes' rule is motivated by the fact that the two probability densities $\rho(\boldsymbol{z}|u)$ and $\rho(\boldsymbol{z}|u,v)$ on the right-hand side of Eq. (1) are much easier to estimate than the probability on the left-hand side, a feature already noted by the designers of the Dinamica EGO modeling environment. It is therefore not surprising that we make use here of the same starting point, Eq. (1), although the calibration method outlined below differs substantially from the one used in this software.

A major aim of the LUC calibration-estimation process is — or should be — to estimate these two probability densities. Both $\rho(\boldsymbol{z}|u)$ and $\rho(\boldsymbol{z}|u,v)$ are estimated for all pixels with LUC state equal to u at t_s (pixels in J)[15]. J is also the set of calibration pixels for $\rho(\boldsymbol{z}|u)$, while I_v is the set of calibration pixels of $\rho(\boldsymbol{z}|u,v)$. This defines the relevant sets of pixels involved in the definition of the calibration and estimation processes (section 3.2). Consequently, in principle, $\rho(\boldsymbol{z}|u,v) = \rho(\boldsymbol{z}|u,v,\boldsymbol{Y}_v)$ is also conditionally dependent on the calibration pixels of the set $\boldsymbol{Y}_v$. This dependence is dropped to alleviate our notations.

This difference between the two probability density has a simple but unavoidable motivation. The probability density $\rho(\boldsymbol{z}|u)$ is defined at each t_s, independently of previous LUC state changes. As some explanatory variables are dynamic, this probability density changes at each time step t_s, as it characterizes a pixel. Both features make the use of J the only possible choice of pixel set for both calibration and estimation explanatory variable values. The probability density $\rho(\boldsymbol{z}|u,v)$ on the other hand is again defined at t_s but some assumption has to be made in order to specify it for allocation [which is the purpose of $P(v|u,\boldsymbol{z})$]. The assumption is that, *a priori*, the structure of the underlying causes producing a change is the same between t_s and t_{s+1} (allocation step) and t_0 and t_1 (calibration data). In other words, this is an *a priori* probability density for allocation, in the Bayesian sense. One may also impose some *a priori* changes to this *a priori* assumption, if, e.g., they are required by the scenario that is implemented in the simulations of interest. But this does not affect the Bayesian *a priori* meaning of this probability density. This being said, even if the underlying causal structure is assumed to be time-independent, the probability density itself must be updated for each pixel, because some explanatory variables are dynamic (recall that P and ρ are attached to pixels).

The estimation of such probability densities has been extensively studied in the machine learning literature, in particular through the use of density estimators (hereafter DE). The resulting transition probability estimation procedure is outlined in Fig. 2. The n_J-tuple $\boldsymbol{Y} = \boldsymbol{Z}$ is used as input for the density evaluation of $\rho(\boldsymbol{z}|u)$, denoted $\tilde{\rho}(\boldsymbol{z}|u)$, while the n_{I_v}-tuples $\boldsymbol{Y}_v$ along with $\boldsymbol{Z}$ are used as input in the density evaluation of $\rho(\boldsymbol{z}|u,v)$, denoted $\tilde{\rho}(\boldsymbol{z}|u,v)$. This involves two more detailed sub-processes (fit and eval). The fit procedure tells the estimator what the observed data are, while the eval procedure estimates the probability density from these data and from an appropriate density estimator technique. Both procedures are described in section 5.2, drawing on the material of sections 4 and 5. Finally, Bayes' rule is used to compute the estimate $\tilde{P}(v|u,\boldsymbol{z})$ of $P(v|u,\boldsymbol{z})$, from the previous two probability density estimates. This last step requires the use of a correction algorithm (Bayes' adjustment) to cope with the approximations involved in these estimators (this is detailed in section 5.3).

3.4. Probability Distribution Properties

The probability density estimates should verify the following closure relations:

$$\int_{\boldsymbol{z}\in\mathbb{R}^d} \tilde{\rho}(\boldsymbol{z}|u)d\boldsymbol{z} = 1, \tag{2}$$

$$\int_{\boldsymbol{z}\in\mathbb{R}^d} \tilde{\rho}(\boldsymbol{z}|u,v)d\boldsymbol{z} = 1, \tag{3}$$

as the exact (unknown) probability densities ρ do satisfy these very same relations.

Furthermore, although allocation algorithms generally use a method of allocation in patches and not pixel by pixel, the transition probabilities are defined so that the proportion of pixels changing state must

[15] This is further explained and commented upon in section 4.2 for $\rho(\boldsymbol{z}|u,v)$.

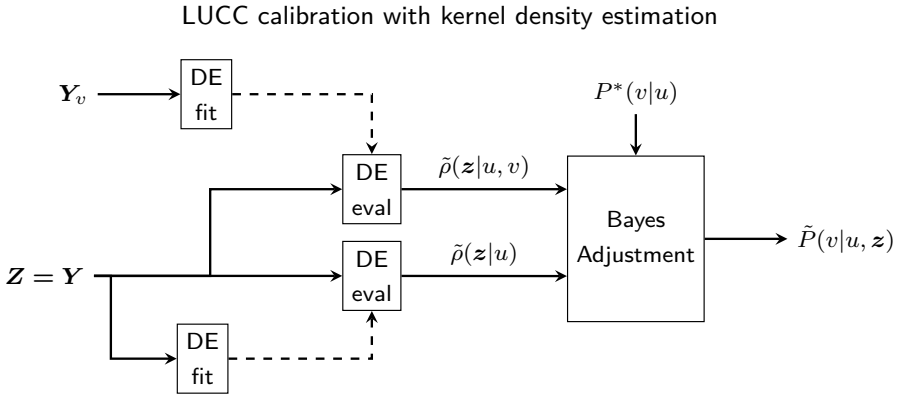


Figure 2: Transition probability estimation architecture. The density estimation processes fit and eval are detailed in Fig. 7. The Bayes adjustment algorithm is detailed in section 5.3.

be equal on average to the expected level defined by $P^*(v|u)$, i.e.,

$$\int_{\boldsymbol{z}\in\mathbb{R}^d} \tilde{P}(v|u,\boldsymbol{z}) \times \tilde{\rho}(\boldsymbol{z}|u)d\boldsymbol{z} = \int_{\boldsymbol{z}\in\mathbb{R}^d} \tilde{\rho}(v,\boldsymbol{z}|u)d\boldsymbol{z} = P^*(v|u), \tag{4}$$

These relations are exact for the exact probability distributions. It is therefore useful to enforce them as well on the estimations of these distributions. This is achieved through the Bayes adjustment procedure of section 5.3.

4. Transition Probability Estimation Method

We now flesh out the process generically described in section 3.3. We start with the probability density estimation procedure (noted DE in figure 2). Density estimation is widely addressed problem in the machine learning literature, and the most common approach, kernel density estimation, or KDE, is described below and adopted here. The presentation is self-contained, and requires no prior knowledge of machine learning techniques.

4.1. Kernel Density Estimation Short Summary

Kernel density estimation is one of the most important non parametric methods used in machine learning to estimate a density distribution. A kernel is a function of finite support or finite support integral. This function is usually monotonically decreasing away from its center, and symmetric with respect to this center; this center itself is a free parameter of the kernel function. The extension of the kernel function is characterized by a parameter called "bandwidth" in the machine learning literature and we stick to this designation in the remainder of this work. The kernel function integral is normalized to unity. For probabilistic problems, kernel density estimation makes use of observed statistical realizations of the underlying probability distributions. It has been widely studied for both univariate and multivariate distributions (Wand, 1992).

Kernel density estimation proceeds through a kind of moving mask average, applied in the space of the explanatory variables of the problem; this provides an estimate of the actual probability distribution at each point (i.e., individual explanatory variable values) in this space from the location of the calibration points themselves, from an estimate of the local density of surrounding calibration points. In practice, this is achieved by attaching a kernel function of well-defined shape and extent (bandwidth) to each calibration point in explanatory variable space (i.e., using all observation points as Kernel centers, see Fig. 3) and summing these kernel functions at all estimation points to obtain an estimate of the real density function (Fig. 4). Depending on whether $\rho(\boldsymbol{z}|u,v)$ or $\rho(\boldsymbol{z}|u)$ is estimated, the calibration points belong respectively to $\boldsymbol{Y}_v$ or $\boldsymbol{Z}$.

As advertised above, when the Kernel is symmetric with respect to its center (a condition satisfied by most kernels) this estimation is easily shown to be equivalent to a weighted-average of calibration points in explanatory variable space, the weight being obtained by the value of the kernel attached to the calibration point, the kernel function itself being attached to the point where the estimate is performed. This equivalent view gives some intuitive feel of what a kernel density estimation actually does. Note that this estimation is

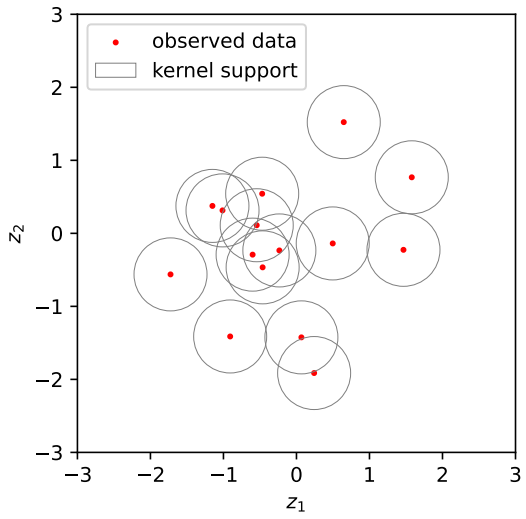


Figure 3: Illustration of the kernel density estimation method in a simplified 2-dimensional case. Red points represent observed data and gray circles highlight the kernel support, *i.e.* the radius of influence of the kernel around each observation.

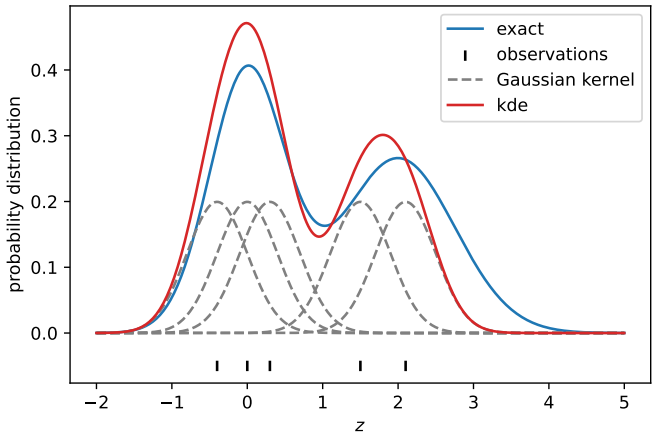


Figure 4: Illustration of the uni-variate kernel density estimation given 6 observations. The underlying exact probability density is equal to $1/2 \times [\mathcal{N}(0, 0.5) + \mathcal{N}(2, 0.75)]$ where $\mathcal{N}(z, \sigma)$ is a unidimensional gaussian distribution of mean z and dispersion σ.

performed everywhere in explanatory variable space. This multivariate interpolation based on an estimate of the local density of calibration points is the key feature allowing us to avoid the usual assumption of independence of explanatory variables, without loss of precision.

These two ways of interpreting the kernel density estimation method are equivalent and have immediate consequences. First, as for all estimation methods, the more calibration data one has, the more precise the estimation will be. Second, the bandwidth is a key parameter which directly controls the quality of the estimation process. Indeed, choosing too narrow a bandwidth is a source of over-interpretation (over-fitting) and the estimate will be over-dominated by the particular observations that are available, resulting in noisy estimations. On the other hand, choosing too large a bandwidth leads to over-smoothing and to a loss of information with respect to the data. Determining an adequate bandwidth is therefore a crucial and widely discussed issue in the literature. This is addressed in section 4.5.

Multivariate estimations have been discussed by Wand (1992) in a very general setting. We use here a simplified form of these authors' approach, where the kernel is in fact a product of uniform kernels (Scott,

2015). More precisely, we propose to estimate $\rho(\boldsymbol{z}|u)$ through:

$$\tilde{\rho}(\boldsymbol{z}|u) = \frac{1}{n_J h^d} \sum_{j \in J} K\left(\left\| \frac{\boldsymbol{z} - \boldsymbol{y}^j}{h} \right\|_p \right), \tag{5}$$

where $\|(\boldsymbol{z} - \boldsymbol{y}^j)/h\|_p$ is the p-order distance in explanatory variable space, K the kernel function whose integral is equal to one and h is the bandwidth, i.e., the kernel characteristic extension parameter. A commonly chosen distance is the usual second order (Euclidean) one, defined by:

$$\|(\boldsymbol{z} - \boldsymbol{y}^j)/h\|_2 = \left[\sum_{k=1}^{d} (z_k - y_k^j)^2 / h^2 \right]^{1/2}, \tag{6}$$

Similarly:

$$\tilde{\rho}(\boldsymbol{z}|u, v) = \frac{1}{n_{I_v} h^d} \sum_{i \in I_v} K\left(\left\| \frac{\boldsymbol{z} - \boldsymbol{y}^i}{h} \right\|_p \right). \tag{7}$$

The Gaussian kernel is the most common one in the literature. It is defined as :

$$K_{\text{gaussian}}(r) = \frac{1}{(2\pi)^{d/2}} \exp\left(-\frac{r^2}{2} \right), \tag{8}$$

where the second-order distance has been used in Eq. (5). However, as will be seen in section 4.4, it may be necessary to use a simpler kernel with computationally less intensive partial integrals. We therefore propose to use alternatively a box kernel defined as:

$$K_{\text{box}}(r) = \frac{1}{2^d} \begin{cases} 1 & \text{if } |r| \leq 1 \\ 0 & \text{else} \end{cases} \tag{9}$$

for an infinite-order distance in Eq. (5), *i.e.* $\|(\boldsymbol{z} - \boldsymbol{y}^i)/h\|_\infty = \max_{k=1}^{d} (z_k - y_k^i)/h$.

4.2. Comments

Eqs. (5) and (7) are the most important relations in our calibration-estimation procedure. As already pointed out in sections 3.2 and 3.3, the estimation explanatory values $\boldsymbol{z}$ are associated to pixels in J, i.e., to all pixels in state u at t_s. For $\rho(\boldsymbol{z}|u)$, calibration and estimation explanatory values are identical, while for $\rho(\boldsymbol{z}|u, v)$, the calibration values come from the calibration data (explanatory variable values associated to pixels in I_v).

More precisely, the meaning of these two equations is the following. Both are defined at any point $\boldsymbol{z}$ in explanatory variable space. Eq. (5) estimate is based on the local density of calibration points in state u at t_s, i.e., points corresponding to pixels in ensemble, the location of which is specified by $\boldsymbol{y}^i$ ($i \in J$): the higher the density of calibration points in the vicinity of $\boldsymbol{z}$, the larger the probability density, the vicinity being defined by the kernel bandwidth. This gives directly the estimate of the probability density $\tilde{\rho}(\boldsymbol{z}|u)$ at t_s. In practice this estimate is needed at all points $\boldsymbol{z}^i$ ($i \in J$), i.e., the estimation points are the same as the calibration points, a feature already pointed out in our definitions (section 3.2).

Eq. (7) is more subtle. Its first, immediate meaning is that it gives the estimate of the probability $\tilde{\rho}(\boldsymbol{z}|u, v)d\boldsymbol{z}$ that a given $\boldsymbol{z}$ within $d\boldsymbol{z}$ at t_0 leads to the $u \to v$ transition in the time span $t_1 - t_0$. This estimates results from the density in explanatory space $\mathcal{D}$ of calibration pixels in state u who have effectively transited from u to v between t_0 and t_1 (pixels in I_v). But this estimate applies also to all other points in state u at t_0, on the basis of this same local density of calibration points in $\mathcal{D}$; as such, because these other points did not actually change state, the estimate has a counterfactual flavor: the other points could have undergone a transition, with this probability for $\boldsymbol{z}$ within $d\boldsymbol{z}$ to have been selected.

The second meaning follows from the *a priori* assumption stated in section 3.3: this probability density also characterizes the probability that a given $\boldsymbol{z}$ within $d\boldsymbol{z}$ at t_s leads to the $u \to v$ transition in the time

span[16] $t_{s+1} - t_s = t_1 - t_0$. Consequently, it can in particular be applied to all $\boldsymbol{z}$ values associated to pixels in J at t_s. Note finally that in this second meaning, the calibration pixels originally in I_v at t_0 may no longer have the same explanatory variable values at t_s or even may no longer be in state u, which may make Eq. (7) paradoxical at first sight. However, this is of no consequence, as the *a priori* assumption does not bear on pixels themselves, but on the relation their $\boldsymbol{z}$ value bears to the explanatory variable values of the calibration set, in explanatory space $\mathcal{D}$ (and not in physical space), i.e., to the set $\{\boldsymbol{y}^i \in \mathcal{D} | i \in I_v\}$: this set exists in explanatory variable space $\mathcal{D}$ independently of time and pixels.

In patten-based LUCC modeling, time steps are usually small enough so that the fraction of pixels undergoing a specific $u \to v$ transition is small (e.g., $\sim 10^{-3}$). Consequently, the available data for calibration is rather sparse, which implies that these data tend to be undersampled and the resulting probability estimates to be noisy as soon as more than a couple of explanatory variables are selected (a very common situation). In particular, in relation to undersampling, some combinations of explanatory variable values may not have been observed for a specific transition even though very similar combinations may have. These unobserved combinations have no reason to be *a priori* assigned a null transition probability.

A common way to circumvent undersampling issues is to assume that explanatory variables are independent (as done, e.g., in Dinamica EGO), so that the needed probability distributions can be computed from the product of individual probability distributions: $\rho(\boldsymbol{z}|u) = \prod_k \rho(z_k|u)$ and $\rho(\boldsymbol{z}|u,v) = \prod_k \rho(z_k|u,v)$. The individual (marginal) probability distributions are unavoidably much less noisy than the multidimensional ones, so that this does indeed reduce the overall noise level in the estimates. However, this independence assumption has undesirable consequences on the accuracy of the estimates, even when the level of correlation between explanatory variables is low. Namely, this includes in the probability distribution combinations of individual z_k values which would otherwise actually have zero probability of transition, and this reduces the probability distribution of combinations which can actually occur (in order to satisfy the cumulative probability distribution closure relation). These deviations may be substantial even if the level of correlation between explanatory variables is low; e.g., even a 20% correlation as measured by Pearson's correlation matrix (a rather common level) may lead to substantial (factor several) differences in the probability distribution magnitude for some practically important combinations. This counter-intuitive feature can be checked on simple examples. This is the major reason why we have avoided the assumption of statistical independence of explanatory variables.

Kernel density estimations altogether avoid these issues. Undersampled explanatory variable combinations probabilities are corrected for by the interpolation provided by the estimator, and the noise of multidimensional distributions can be reduced in an optimal or near-optimal way by an appropriate choice of the Kernel bandwidth (see section 4.5). In particular, our KDE method provides an interpolation in explanatory variable space that provides a natural level of transition probability for unobserved but legitimate combinations, as explained in relation to our discussion of Eq. (7) at the beginning of this section. These features are exemplified in section 7.

Finally, Bayes rule is used in full in this method (hence its name Bayes-eKDE[17]). In particular, probability densities and the state transition probability are not defined up to some kind of normalization factor here, contrarily to what is done, e.g., in Dinamica EGO. The main reason for this stems from our choice of calibration-estimation method, but also from the fact that in order to identify biases systematically in existing software, one cannot enforce the overall amount of state change per time step [defined by $P^*(v|u)$ exactly] for each realization of the allocation, but only statistically over a number of realizations of the same allocation step. For this purpose the exact probability density must be used. This will be detailed in our forthcoming paper on allocation.

4.3. Whitening Transformation

We have chosen a uniform bandwidth in all dimensions. However, the different explanatory variables, considered individually, would in general require different bandwidths, as the distribution of observed point will depend on the explanatory variable under consideration. In order for the choice of a unique bandwidth to be meaningful, one first needs to normalize the data, so that, in each direction in explanatory space, the

[16] Other time spans can be envisioned, but this requires to rescale the probability density. Such a rescaling is dealt with, e.g., in Dinamica EGO, and will be addressed in our forthcoming paper on allocation.

[17] eKDE refers to efficient KDE, see section 5.1.

data mean is zero and the covariance is the unity matrix. Such a transformation of the data is called a whitening transformation (WT).

This procedure is a generalization of the standardizing of a random variable $\boldsymbol{x} = (x_1, x_2 \ldots, x_d)^T$ (the superscript T refers to the transposed vector), which is generically carried out by defining $\boldsymbol{X} = \boldsymbol{V}^{-1/2}\boldsymbol{x}$ where $\boldsymbol{V}$ is the diagonal matrix of variances σ_i^2 of x_i: this results in a unit variance for X_i but does not remove the correlations between X_i and X_j. The whitening transform performs both objectives. More precisely, defining the covariance matrix $\boldsymbol{C}$ of $\boldsymbol{x}$, the whitening transformation $\boldsymbol{W}$ of $\boldsymbol{x}$ transforms $\boldsymbol{x}$ into $\boldsymbol{X} = \boldsymbol{W}\boldsymbol{x}$, with covariance matrix $\boldsymbol{W}\boldsymbol{C}\boldsymbol{W}^T = \boldsymbol{I}$ where $\boldsymbol{I}$ is the identity matrix. From $\boldsymbol{W}\boldsymbol{C}\boldsymbol{W}^T = \boldsymbol{I}$, one immediately obtains $\boldsymbol{W}(\boldsymbol{C}\boldsymbol{W}^T\boldsymbol{W}) = \boldsymbol{W}$, hence

$$\boldsymbol{W}^T\boldsymbol{W} = \boldsymbol{C}^{-1}, \tag{10}$$

which is the most important relation defining the whitening transformation. Unfortunately, there is an infinite number of matrices satisfying this constraint, and the problem needs to be further specified from relevant considerations. This is the object of the present section.

To this effect, we rely on the diagonalization of the covariance matrix, which is discussed first. In practice, we need to normalize (in the meaning just defined) data corresponding to both $\rho(\boldsymbol{z}|u)$ and $\rho(\boldsymbol{z}|u, v)$. The corresponding number of pixels are n_I and n_{I_v}, respectively. In the next subsection, both cases are treated simultaneously by introducing $n = n_I$ or n_{I_v}.

4.3.1. Covariance matrix estimate and its diagonalization

Let us first recall the definition of the $d \times d$ covariance matrix $\boldsymbol{C}$:

$$C_{k,k'} \equiv \mathrm{cov}(z_k, z_{k'}) \equiv E\Big((z_k - \mu_k))\,(z_{k'} - \mu_{k'})\Big), \tag{11}$$

where $E(\cdot)$ stands for the expectation value (ensemble mean) over all pixels in initial state u (i.e., all pixels in I), and $\mu_l = E(z_l)$ is the sample mean. In vector notation, $\boldsymbol{C} = E([\boldsymbol{z} - \boldsymbol{\mu}][\boldsymbol{z} - \boldsymbol{\mu}]^T)$ where the superscript T refers to the transposition operation. In practice, one chooses to work with centered data, i.e., data with zero mean. To this effect, we define translated vectors

$$\bar{\boldsymbol{z}} = \boldsymbol{z} - \boldsymbol{\mu}, \tag{12}$$

so that $\boldsymbol{C} = E(\bar{\boldsymbol{z}}\bar{\boldsymbol{z}}^T)$.

In general neither the sample mean nor the expectation values are exactly known, and the covariance matrix is evaluated from its estimate $\hat{\boldsymbol{C}}$ on the available data[18]:

$$\hat{C}_{k,k'} = \frac{1}{n-1}\sum_{i=1}^{n}(z_k^i - \hat{\mu}_k^i)(z_{k'}^i - \hat{\mu}_{k'}^i), \tag{13}$$

where $\hat{\mu}_l = \sum_i z_l^i/n$ is the sample estimate of μ_l. The replacement of the expected mean by a sample estimate creates a bias in the evaluation of the correlation coefficients. Indeed, defining $S_{kk'}^2 = \sum_{i=1}^{n}(z_k^i - \hat{\mu}_k^i)(z_{k'}^i - \hat{\mu}_{k'}^i)/n$ and $\sigma_{kk'}^2 = E([z_k - \mu_k][z_{k'} - \mu_{k'}])$, one has $E(S_{kk'}^2) = (n-1)\sigma_{kk'}^2/n$. This justifies the normalization adopted in the estimate of Eq. (13). In vector notation, $\hat{\boldsymbol{C}} = n/(n-1) \times \sum_{i=1}^{n}[\boldsymbol{z}^i - \hat{\boldsymbol{\mu}}^i][\boldsymbol{z}^i - \hat{\boldsymbol{\mu}}^i]^T$. We also introduce estimates of the translated vectors[19], $\hat{\boldsymbol{z}} = \boldsymbol{z} - \hat{\boldsymbol{\mu}}$ so that

$$\hat{\boldsymbol{C}} = \frac{1}{n-1}\sum_{i=1}^{n}\hat{\boldsymbol{z}}^i\hat{\boldsymbol{z}}^{T,i} \tag{14}$$

As this matrix is real and symmetrical, it is diagonalizable with the help of a unitary matrix $\boldsymbol{V}$, i.e.,

$$\hat{\boldsymbol{C}} = \boldsymbol{V}\boldsymbol{L}\boldsymbol{V}^T, \tag{15}$$

[18] Introducing $\hat{\boldsymbol{Z}}$, the matrix of size $n_I \times d$ (number of pixels times number of explanatory variables) which contains the data to be transformed, one can write the estimate of the covariance matrix in more compact form: $\hat{\boldsymbol{C}} = \hat{\boldsymbol{Z}}\hat{\boldsymbol{Z}}^T/(n_I - 1)$.

[19] This estimate has vanishing expectation value as well as vanishing mean over the sample.

where $\boldsymbol{L}$ is the $d \times d$ diagonal matrix of the eigenvalues in decreasing order and $\boldsymbol{V}$ is an eigenvector matrix of size $d \times d$. Thus by definition, $\boldsymbol{L}$ is a diagonal matrix whose diagonal elements are the variances of the corresponding variables $z_k - \hat{\mu}_k$.

Let us finally introduce the inverse of the square root of the estimate of the covariance matrix[20] $\hat{\boldsymbol{C}}$:

$$\hat{\boldsymbol{C}}^{-1/2} = \boldsymbol{V}\boldsymbol{L}^{-1/2}\boldsymbol{V}^T, \tag{16}$$

which is directly related to the whitening transformation defined below.

4.3.2. Whitening Transformation expression

There is an infinite number of possible choices for the whitening matrix $\boldsymbol{W}$ (Kessy et al., 2018). A possible choice is the following[21]:

$$\boldsymbol{W} = \hat{\boldsymbol{C}}^{-1/2}, \tag{17}$$

with associated normalized data vector

$$\hat{\boldsymbol{z}}^* = \boldsymbol{W}\hat{\boldsymbol{z}}. \tag{18}$$

From Eq. (16), $\boldsymbol{W} = \boldsymbol{W}^T$ so that $\boldsymbol{W}^T\boldsymbol{W} = \hat{\boldsymbol{C}}^{-1}$ as required from Eq. (10). Furthermore, from Eq. (18), $\hat{\boldsymbol{z}}^*$ is centered on the sample because $\hat{\boldsymbol{z}}$ is. Finally, the estimate $\hat{\boldsymbol{\Sigma}}$ of covariance matrix of $\hat{\boldsymbol{z}}^*$ is equal to the identity matrix[22]:

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{n-1}\sum_{i=1}^{n} \hat{\boldsymbol{z}}^{*,i}\hat{\boldsymbol{z}}^{*,T,i} = \boldsymbol{W}\hat{\boldsymbol{C}}\boldsymbol{W}^T = \boldsymbol{I}, \tag{19}$$

from Eqs. (15), (16), (17) and (18). Thus, the chosen matrix $\boldsymbol{W}$ achieves our normalization purpose: the data are now centered, have normalized dispersion, and vanishing cross-correlations.

Note that both the original and transformed probability distributions must satisfy the closure relations Eqs. (2) and (3). For example, for $\rho(\boldsymbol{z}|u)$, one has

$$\int_{\boldsymbol{z}\in\mathbb{R}^d} \tilde{\rho}(\boldsymbol{z}|u)d\boldsymbol{z} = \int_{\boldsymbol{z}^*\in\mathbb{R}^d} \tilde{\rho}(\boldsymbol{z}^*|u)d\boldsymbol{z}^* = 1. \tag{20}$$

As the identity of the probabilities holds over any range in random variable space, the change of variable defined by Eq. (18) leads to

$$\rho(\boldsymbol{z}|u) = |\det \boldsymbol{W}|\rho(\boldsymbol{z}^*|u), \tag{21}$$

$$\rho(\boldsymbol{z}|u,v) = |\det \boldsymbol{W}|\rho(\boldsymbol{z}^*|u,v). \tag{22}$$

The whitening transformation is noted WT in Fig. 7 and the whitening transformation scale specified in Eqs. (21) and (22) is noted *WT scale*. In the remainder of this work, the superscript * is dropped; unless otherwise specified, all variables are transformed variables and the kernels defined earlier apply to these transformed variables.

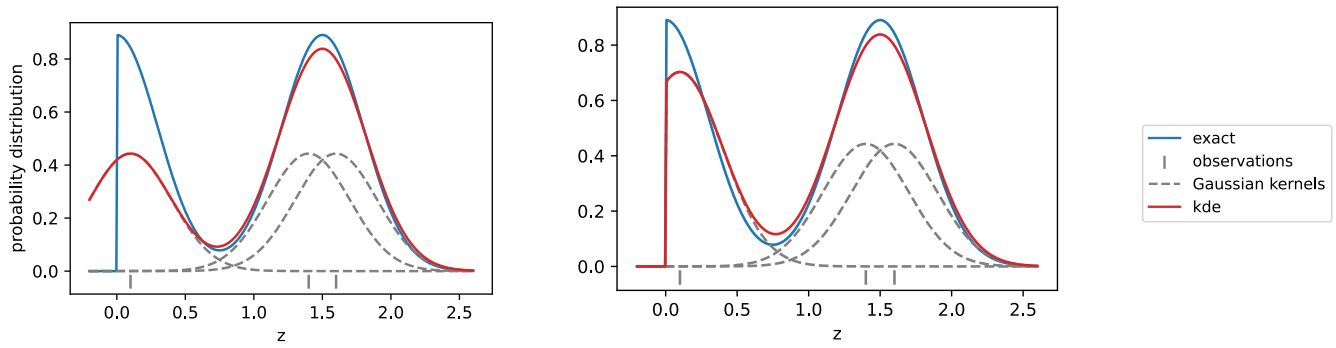
4.4. Boundary Bias Correction

Close to an explanatory variable boundary, a kernel density estimation presents a well-known bias (Diggle, 1985). For bounded variables, the kernel density estimation method will underestimate the density in the vicinity of the variable boundary because the method assimilates the absence of elements beyond the boundary as a probability of occurrence (non-occurrence, in this case) and not as an intrinsic property

[20] One can show that this matrix is unique.

[21] This is the first of the five natural choices discussed by Kessy et al. (2018), known as the ZCA-Mahalanobis whitening transformation procedure (ZCA stands for "zero-phase components analysis"). See Kessy et al. (2018) for more details.

[22] Note that this does not imply that the related random variables $\hat{z}_k^*$ are statistically independent, so that the associated probability distributions, $\rho(\boldsymbol{z}|u)$ or $\rho(\boldsymbol{z}|u,v)$ cannot be assumed to be separable.



(a) without correction (b) with correction

Figure 5: Very simple illustration of the boundary bias correction in a unidimensional case restricted to the half-space $[0, +\infty)$. In the absence of correction, observations near the border end up in an underestimation of the probability density 5a. The boundary bias correction allows us to correct this phenomenon while enforcing the probability closure for the obtained density estimate 5b. Although the correction is not perfect, it results in a substantial gain in precision in the estimation of the probability density distributions.

(Fig. 5a). For this reason, the kernel should be truncated beyond the boundary. In effect this amounts to defining a new kernel that is the truncated version of the original one, and this new kernel must accordingly be correctly normalized.

In a multidimensional case, boundaries form hyperplanes which define a set of closed half-spaces; this applies as well to transformed data as discussed in section 4.3, and the boundary hyperplanes are also transformed into hyperplanes in the transformed space (which however are no longer aligned with the different explanatory variables). The half-spaces intersection is therefore the domain of interest in the transformed explanatory variable space and is noted $\mathcal{D}$. A given kernel at a particular observation point should then be divided by the cumulative distribution function $\int_{\mathcal{D}} K(\cdot)$ in order to account for this kernel redefinition and normalization (Fig. 5b). In other words, the kernels truncated by the boundaries of $\mathcal{D}$ must be renormalized by their effective volume in $\mathcal{D}$, as kernels are necessarily normalized to unity.

Thus, the estimated probability $\tilde{\rho}(\boldsymbol{z}|u,v)$ defined in Eq. (7) can be corrected as follows

$$\tilde{\rho}(\boldsymbol{z}|u,v) = \frac{1}{n_{I_v} h^d} \sum_{i \in I_v} \frac{K\left(\left\| \frac{\boldsymbol{z} - \boldsymbol{y}^i}{h} \right\|_p \right)}{\int_{z \in \mathcal{D}} K\left(\left\| \frac{\boldsymbol{z} - \boldsymbol{y}^i}{h} \right\|_p \right)}. \tag{23}$$

and a similar correction is applied to compute $\tilde{\rho}(\boldsymbol{y}|u)$ from Eq. (5).

However, $\int_{\mathcal{D}} K(\cdot)$ may be difficult to calculate efficiently in an exact or approximate way, depending on the nature of the kernel K. Thus, in the case of the Gaussian kernel [Eq. (8)], the computation of the integral over the domain D requires an approximate computational method, e.g. a Monte-Carlo computation of integrals (due to the multi-dimensional nature of the problem) which is not very efficient numerically if one has to repeat this method for each observed element near the boundaries.

In order to avoid this problem, we have adopted the box kernel based on the infinite norm [Eq. (9)]. With this choice, computing the integral of the kernel on the domain $\mathcal{D}$ amounts to determining the volume of a hypercube limited by several half-spaces. Cho and Kim (2020) proposed an exact and numerically efficient equation for this calculation. We implemented this method in a dedicated python package HYPERCLIP[23] and used it to implement the edge correction discussed in this section in our algorithms.

[23] https://gitlab.inria.fr/fmazy/hyperclip

4.5. Bandwidth Selection

Determining the optimal bandwidth parameter for the estimation of a probability density from a set of observations is a widely studied problem in the literature. Many methods have been proposed. The vast majority of them are based on the minimization of the squared difference between the estimate and the exact probability. Since the exact probability density is unknown, it is necessary to approximate this difference, such as the Plug-In method (Wand and Jones, 1994; Duong and Hazelton, 2003) which injects an approximation obtained by an estimate made with a pilot bandwidth. Also widespread, the cross-validation method (Rudemo, 1982; Sain et al., 1994; Duong and Hazelton, 2005) uses among other things the leave-one-out estimator. Chacón and Duong (2018) and Schindler (2011) summarize these methods.

However, once the squared difference approximation has been performed, it is necessary to vary the bandwidth and determine a minimum. This step is ambiguous in most real applications. Indeed, in our LUCC problem, as soon as the number of explanatory variables increases, most of the time, no minimum stands out, either because of the presence of several local minima, or because of the monotonicity of the function. Very often, such methods lead to an unsuitable choice of bandwidths, which is often similar to undersmoothing.

In order to circumvent these problems, Terrell (1990) proposed to use a "maximum smoothing principle". This method is based only on general information about the data such as the standard deviation, but has the advantage of avoiding naive over-interpretation or over-fitting of the data. The bandwidth proposed by Terrel for a multivariate problem once a whitening transformation has been performed is defined as follow :

$$h_{\text{Terrel}} = \frac{(d+8)^{(d+6)/2}\ \pi^{d/2} \int K^2}{16\ n\ (d+2)\ \Gamma(d/2+4)}, \tag{24}$$

where n is the number of observation data points (pixels), d the number of explanatory variables, Γ is the Gamma function, and $\int K^2 = 2^{-d}$ for the box kernel. In LUCC analyses, avoiding data over-interpretation is often desirable, and the reliability and speed of the calculation makes this a method of choice in this context.

5. Implementation

In the previous section, we proposed a theoretical and statistically correct method to estimate the transition probabilities, notably by introducing the kernel density estimation method. However, at first sight, this method is numerically very intensive. Moreover, in an actual case study, we need to take care of inappropriate numerical values in the output of Bayes' formula (e.g., large values in under-sampled regions, in particular in the tails of the distributions). We therefore propose in this section an efficient method for approximating density estimation by kernels and detail a Bayes fitting algorithm designed to deal with the problem just mentioned. To simplify the presentation of the following algorithms, we focus here on the estimation of $\rho(\boldsymbol{z}|u,v)$. The estimation of $\rho(\boldsymbol{z}|u)$ is done in a similar way, *mutatis mutandis*.

5.1. Efficient Kernel Density Estimation

A density estimation with the kernel method requires a large number of operations in a naive implementation Scott (2015): calling m the number of calibration data points and n the number of estimation data points, the required number of operations is $\mathcal{O}(mn)$. In the present context, the m training pixels are either the n_J calibration data pixels observed in state u, or the n_{I_v} calibration data pixels having undergone a transition $u \to v$, while the n estimation pixels are all pixels of interest (i.e., all n_J pixels when considering all possible transitions from a given initial state u).

This estimation can be approximated and accelerated with the use of fast Fourier transforms (OBrien et al., 2016; Langrené and Warin, 2019) or hashing (Charikar and Siminelakis, 2017; Siminelakis et al., 2019; Backurs et al., 2019). These methods, although attractive, have the disadvantage of generating a plain matrix whose size becomes numerically prohibitive as soon as the number of explanatory variables exceeds 3 which is routinely the case in LUCC modeling (any comparison with these other methods is then impossible because of their incapacity to treat the given problem).

Sub-sampling strategies have been alternatively proposed (Zheng et al., 2013). Various methods have been elaborated to this effect while enforcing a given maximum error level (Phillips, 2013; Phillips and Tai, 2018b,a; Tai, 2022). The simplest such method consists in performing a uniform random sub-sampling, provided that the number of training elements exceeds a certain threshold.

Alternatively, we propose a simple alternative KDE approximation method called efficient KDE (eKDE). The basic principle is the following. We start by binning the set of observed points at fine grain with respect to the chosen bandwidth. Then, instead of considering a kernel on each of the observed points, we place only a single kernel in the center of each bin, weighted by the number of observed points within this bin. By extension, when we want to estimate the probability density at a given point, we consider instead the center of the bin to which it belongs and identify the bins that are close enough to contribute to the estimation as a simple weighted summation.

This simplification considerably alleviates the numerical burden of the KDE estimation; however, in spite of its conceptual simplicity, the actual algorithmic implementation of this procedure involves a number of delicate points. We describe this implementation in two steps, fit and estimation, designated in this way in Fig. 7.

5.1.1. Efficient KDE method: fit

We follow Wells and Ting (2019) who proposed to bin the calibration data points. The idea is to use a sufficiently small binning scale so that the KDE main idea is still relevant. To this effect, let us first define a characteristic scale s such that $\int_{-hs}^{hs} K(r)dr \approx 1$. This is introduced mostly for unbounded kernels. For example, we take $s = 3$ for the Gaussian kernel defined in Eq. (8); for bounded kernels, $s = 1$, e.g., for the box kernel defined in Eq. (9). Next, we define the bin width w_k for each explanatory variable k with the help of an auxiliary odd integer q_k; more precisely $w_k = 2hs/q_k$. Note that once explanatory variables have been normalized as described in section 4.3, a unique q and w can be chosen.

This binning associates to a calibration explanatory variable y_k (or rather, its centered and normalized version) a range Γ_k of possible integer values, each representing a bin identified by a dummy integer γ_k: $1 \leq \gamma_k \leq \Gamma_k$. This d-tuple identifies a bin in explanatory variable space. The set of calibration bins is denoted Γ_y. Finally, we divide the number of observation data points associated to each bin by the total number of observations which gives us the set of weights of the bins noted $\boldsymbol{\nu}$.

Fig. 6 illustrates this fit step with the help of a very simple 2-dimensional example (i.e., we make use of two explanatory variables, z_0 and z_1), using the box kernel. We make use, for simplicity, of only 10 observed points (pixels) and one point for which we wish to estimate the probability density (Fig. 6a) — either $\tilde{\rho}(\boldsymbol{z}|u)$ or $\tilde{\rho}(\boldsymbol{z}|u, v)$. We choose $q = 3$ for both explanatory variables and perform the binning (Fig. 6b) for a chosen q. Finally, we identify the set of nearby non-empty bins Γ_y^* and their weighting $\boldsymbol{\nu}$ (Fig. 6c) in the vicinity of the estimation point just defined. We have used q to define the extent of the vicinity of the current bin, as this defines the maximal extent of our bounded kernel function.

If the domain of interest in explanatory variable space is bounded, the boundary bias correction of section 4.4 must be applied. In practice we correct the weights $\boldsymbol{\nu}$ in the following way. The bias correction is directly applied to bins: we define bin center by $\boldsymbol{y}_0$, and attach the kernel function to this center. From the reasoning of section 4.4, the kernel must be divided by the cumulative distribution function $\int_{\mathcal{D}} K(\cdot)$ in order to renormalize the kernel function truncated by the boundaries of $\mathcal{D}$, as kernels are necessarily normalized to unity. This kernel renormalization can equivalently be applied to the kernel weight, leading to the following expression for the renormalized weight $\boldsymbol{\nu}_r$:

$$\boldsymbol{\nu}_r = \frac{\boldsymbol{\nu}}{\displaystyle\int_{z\in\mathcal{D}} K\left(\left\|\frac{\boldsymbol{z}-\boldsymbol{y}_0}{h}\right\|_p\right)}. \tag{25}$$

5.1.2. Efficient KDE method: estimation

We can now estimate the probability density for any estimation point. The points we want to estimate are binned in the same way as in the fit step and are noted Γ_z^*. Thus, the probability density is not estimated at the exact estimation point location in explanatory space but at the nearest bin center. This is performed

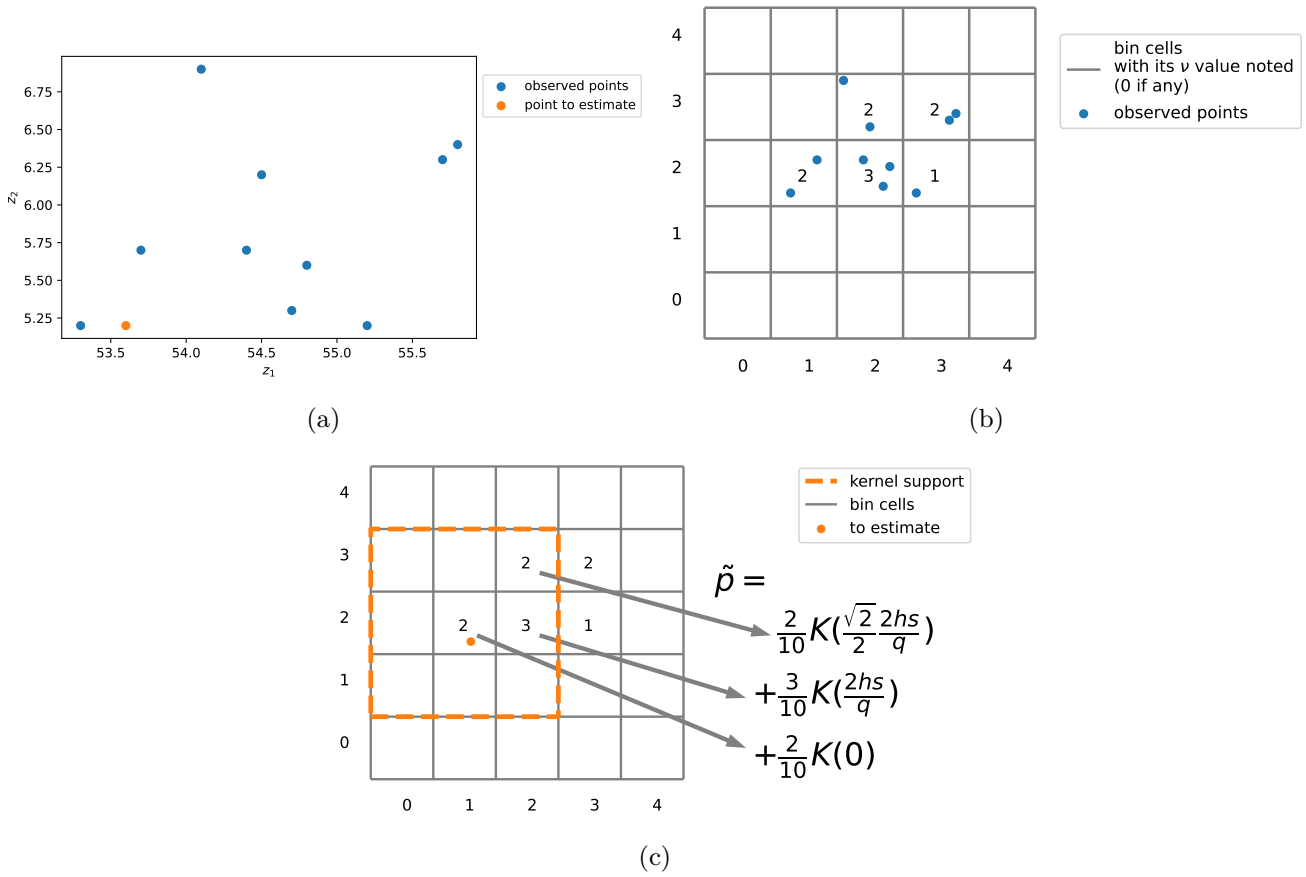


Figure 6: A very simple example to illustrate the fit step of our eKDE method. We have set 10 calibration points in a 2-dimensional explanatory variable space and one estimation point whose probability density we want to estimate (a). The binning and boundary bias correction are shown in (b). The non-empty bins, associated Γ_z^* and corresponding weights $\boldsymbol{\nu}$ are easily identified on the figure: in order to obtain our estimation probability density, we identify the nearby non empty-bins in the dashed-orange delimited square, which contribute to the looked-for estimation (c).

from the set of neighboring bin centers whose kernel's support (bounded by s as explained above) would encompass the bin center where the estimation is made. By construction, this selects q^d bins (since q is an odd integer, there is no ambiguity); this is reduced by selecting the non empty bins in this collection. It is possible to design a numerically efficient algorithm to identify these bins (see our Supplementary Material). For each such bin, the value of the kernel at the estimated point is calculated and multiplied by its weight, defined in the fit step. Finally, we sum these values to obtain the estimated probability density.

This procedure is illustrated on the example presented in the previous section (see Fig. 6). We wish to estimate the probability density at a given point represented in orange in Fig. 6a. The explanatory variable space is first binned in way explained above. Then we identify the relevant non-empty bins (Fig. 6c) among all the bins in a $q = 3$ window around the bin of interest in each direction (including this bin). In our example, there are 4 non-empty bins that will be used in the calculation of the probability density estimate. We compute the center-to-center distances of the bins in the non-binarized space to evaluate the corresponding kernels and sum them with their weights to obtain the estimation (Fig. 6c).

Such an algorithm seems quite simple at first sight. However, when the number of dimensions increases, it is necessary to find an efficient way to explore the calibration points, which together form a kind of sparse matrix, and this operation is not trivial. A too naive exploration is quickly excluded because of the burden in computation time. We describe an efficient algorithm dedicated to this task in Supplementary Material.

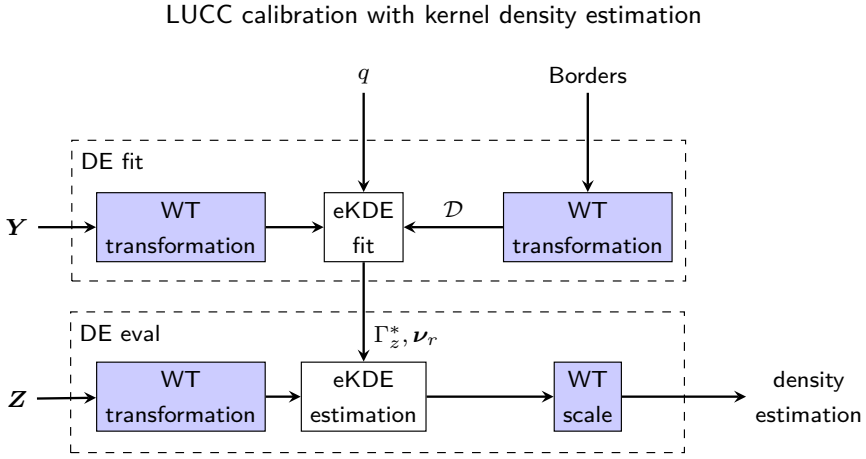


Figure 7: Implementation architecture of the density estimation method. The dashed-line boxes DE fit and DE eval are the ones referenced in Fig. 2. $\boldsymbol{Z}$ and $\boldsymbol{Y}$ refer to the estimation and calibration sets of d-tuples of explanatory variable, respectively. WT is the whitening transformation (section 4.3) and WT scaling refers to the scaling required to relate original and transformed variables probability distributions (section 4.3.2). eKDE is the efficient kernel density estimation method (section 5.1) with fit and estimation functions

5.1.3. Parallelization

In order to speed up the estimation of transition probabilities, one may parallelize some the operations involved. In our model, we chose to parallelize the estimation step (section 5.1.2). Thus, we separate the set of points to be estimated into a certain number of subsets equal to the number of processors that we wish to use. Each processor is assigned to the processing of a single subset and the results are then combined. The case study presented in section 7 exemplifies the significant reduction in computation time achieved by increasing the number of processors (Fig. 12a).

5.2. Density Estimator Architecture

We are now in position to present in a clear and logical fashion the various operations described above and required in our Kernel density estimation method. These are sketched in a sequential manner in Fig. 7 and summarized under the "DE fit" and "DE eval" blocks in Fig. 2. These blocks are also identified with dotted line boxes in Fig. 7. The final objective is to evaluate the probability density of the complete collection of explanatory variables sets of values $\boldsymbol{Z}$ from the calibration set $\boldsymbol{Y}$ ($= \boldsymbol{Y}_v$ or $\boldsymbol{Z}$ as previously discussed).

The whitening transformation is first obtained (WT transformation, section 4.3) and applied to the data. The borders also undergo the same whitening transformation, which defines the transformed domain $\mathcal{D}$ in explanatory variable space; this domain is used in the calculation of the boundary bias correction, sections 4.4 and section 5.1.1. Once the parameter q is chosen the estimator fit procedure can be performed (KDE fit, section 5.1.1). The same whitening transformation is applied to the actual set of interest $\boldsymbol{Z}$. An estimate of the probability density is obtained from the estimation procedure of section section 5.1.2, with a bandwidth selected as proposed in section 4.5. Finally, the whitening scaling required to relate original and transformed variables probability distributions is applied (section 4.3.2).

Our KDE procedure has a single free parameter, q, set by default to $q = 51$ (see Appendix E).

5.3. Bayes Adjustment Process

We now turn to the last process of Fig. 2. The exact (and unknown) transition probability obeys Eq. (4). As this is a known constraint, one may expect that the estimation of the transition probability will be more accurate if it also required to satisfy this constraint. This requirement translates into the following sum over pixels[24]:

[24]In principle we should make this adjustment directly from the integral of the closure relation Eq. (4). Doing this on a pixel sum is nearly equivalent due to the short proof on expectation values that follows, and much simpler and more efficient from an algorithmic point of view.

$$\frac{1}{n_I}\sum_{i=1}^{n_I}\tilde{P}(v|\boldsymbol{z^i},u)=P^*(v|u). \tag{26}$$

This relation can be justified in the following way. Let us introduce the degeneracy factor g_z which counts all pixels having a same $\boldsymbol{z}$ value within $d\boldsymbol{z}$ for a given initial state u. This number is not exactly known, but its expectation value is[25]: $E(g_z)=\rho(\boldsymbol{z}|u)d\boldsymbol{z}/n_I$, by definition (this expectation value is exact in the limit of an infinite number of pixels). This allows us to transform the sum over pixels in an integral over $\boldsymbol{z}$. Therefore[26]:

$$E\left(\frac{1}{n_J}\sum_{i=1}^{n_J}P(v|\boldsymbol{z^i},u)\right)=\int d\boldsymbol{z}P(v|\boldsymbol{z},u)\rho(\boldsymbol{z}|u)=P^*(v|u). \tag{27}$$

where the last equality follows from the closure relation Eq. (4). Enforcing this relation on the estimation of the transition probability itself and not only on its expectation value leads to Eq. (26).

This property has very little chance to be exactly verified in actual data because of the approximate nature of the probability density estimation method. Moreover, imposing a too large global transition probability $P^*(v|u)$ can lead to[27] $\sum_{v\neq u}\tilde{P}(v|u,\boldsymbol{z}^i)>1$ for some pixels i. Thus, in general, the left-hand side of Eq. (26) is not equal to the right-hand side. In order to correct for this difference, we propose a simple algorithm which modifies the transition probability estimate by the ratio of the two, in order to guarantee the required equality (algorithm 1). Note that if the requested probabilities $P^*(v|u)$ are particularly large, a saturation phenomenon of the highest transition probabilities occurs (the corresponding pixels will eventually have all changed state). This Bayes adjustment process has been included in Fig. 2. The corresponding algorithm is outlined below (Algorithm 1).

6. Evaluation Method

Evaluating the results of a LUCC modeling strategy is not a common practice in the literature, which raises questions on the robustness of the obtained results (van Vliet et al., 2016). As an attempt to cope with this issue, it has been proposed to compare different models on a controlled land-use change problem (Mas et al., 2014), from data created by the authors; however, as the transition probability was not specified at the onset, it was only possible to compare the models between them without being able to evaluate their objective quality. This objective quality can only be assessed by evaluating the ability of any modeling strategy to estimate the transition probability $P(v|u,\boldsymbol{z})$, although this quantity is not known in actual case studies. To this effect, we design a completely controlled setting where all relevant transition probabilities are *a priori* known. These are constituted by all quantities on the right-hand side of Bayes rule, Eq. (1), i.e., $P^*(v|u)$, $\rho(\boldsymbol{z}|u)$, and $\rho(\boldsymbol{z}|u,v)$, from which the left-hand side of Bayes rule is completely specified as well. This allows us to evaluate the precision of our procedure along with existing ones in the reconstruction of these quantities from pseudo calibration data constructed from given probability distributions.

Although actual changes occur in patches of contiguous pixels, the calibration-estimation process evaluated here focuses on individual pixel probability distributions; patches can then be produced on this basis. Because we work in explanatory variable space, and in line with the explanations given in section 2, we ignore here the organization of pixels in physical space. Further arguments can be given to legitimate this

[25]This can be defined in an ensemble average meaning, and could be evaluated if one could draw an infinite number of realizations of the observed data, under the same transition probability. This never occurs in reality (there is a unique history), but can be asymptotically performed numerically, and can be used as a basis for this thought experiment.

[26]In fact, we make use here of successive forms of expectation values: first for the estimate itself, then for the degeneracy factor. A more formal proof will be given in our forthcoming paper on allocation.

[27]In principle, and for various reasons that will be discussed in our forthcoming allocation paper, spatially explicit pattern-based LUCC models are meaningful only if the relative number of state changes per time step is "small", in which case the possibility just discussed should not arise. But one may not *a priori* exclude practical applications in which this is not true, at least for some transitions, even though these should be avoided for self-consistency and precision.

Algorithm 1: Bayes Adjustment (BA). It is an outline algorithm without reference to a given programming language.

Input:
for all i, $\tilde{\rho}(\boldsymbol{z}_i|u)$,
for all v, and for all i, $\tilde{\rho}(\boldsymbol{z}_i|u,v)$,
for all v, $P^*(v|u)$,
n_c^* (a natural number to stop the algorithm in case of non convergence)

for all i, $\rho(v|u,\boldsymbol{z}_i) := \dfrac{\rho(\boldsymbol{z}_i|u,v)}{\rho(\boldsymbol{z}_i|u)}P^*(v|u)$
for all i, $P(v|u,\boldsymbol{z}_i) := \dfrac{P(v|u,\boldsymbol{z}_i)}{\frac{1}{m}\sum_{i=1}^{m}P(v|u,\boldsymbol{z}_i)}P^*(v|u)$
for all i, $s_i := \sum_{v\neq u}P(v|u,\boldsymbol{z_i})$
$n_c \leftarrow 0$
while it exists i such as $s_i > 1$ and $n_c < n_c^*$ **do**
  for all i such as $s_i > 1$ and for all $v \neq u$, $P(v|u,\boldsymbol{z}_i) := \dfrac{P(v|u,\boldsymbol{z}_i)}{s_i}$
  for all i and for all $v \neq u$, $P(v|u,\boldsymbol{z}_i) := \dfrac{P(v|u,\boldsymbol{z}_i)}{\frac{1}{m}\sum_{i=1}^{m}P(v|u,\boldsymbol{z}_i)}P^*(v|u)$
  for all i, $s_i := \sum_{v\neq u}P(v|u,\boldsymbol{z_i})$
  $n_c := n_c + 1$
for all i, $P(v=u|u,\boldsymbol{z}_i) = 1 - \sum_{v\neq u}P(v|u,\boldsymbol{z}_i)$
return for all i, $P(v|u,\boldsymbol{z}_i)$

procedure. First, the labelling process of all pixels is arbitrary, and the spatial continuity may be disorganized in this labelling process without problem as long as the pixel coordinates are known, and as long as pixels correlations are ignored (this is a feature of the probability distributions involved in Bayes rules). Second, we consider only continuous probability distributions in explanatory variable space (as categorical explanatory variables pose no specific calibration problem). In such a case, if explanatory variables are continuous functions of planar coordinates, the probability distribution represented in physical space will also be continuous. Therefore, one can focus on reconstructing this probability distribution in explanatory variable space, without loss of generality; the associated probability distribution in physical space is recovered from the explanatory variables dependence on spatial location. Finally, ensuring on this basis a correct reconstruction of the pixel probability distributions is sufficient to ensure that patch construction is a well-defined process (see again the explanation given in section 2 on this point, which will be more formally proved elsewhere).

Also, allocating final states to pixels individually can be performed by a very simple algorithm. This allows us to focus on the efficiency and precision of the calibration-estimation procedure itself. We will examine the efficiency of patch allocation in our forthcoming paper on allocation.

One last word. Calibration is performed on two given maps, produced at times t_0 and t_1. The pixel transition probability $P(v|\boldsymbol{z},u)$ that is produced from these data depends implicitly on the time step $t_1 - t_0$. It is possible to produce a transition probability on a different time-step, with the help of additional assumptions, usually that the distribution of patch sizes produced during the time-step is independent of the time-step but that their total number is proportional to the time-step[28].

[28]Both assumptions implicitly assume that time-steps are "small enough" (how small requires to be defined). More sophisticated assumptions can be used if this is not the case. We will not discuss these points here, but will return to this time-step issue in our allocation paper.

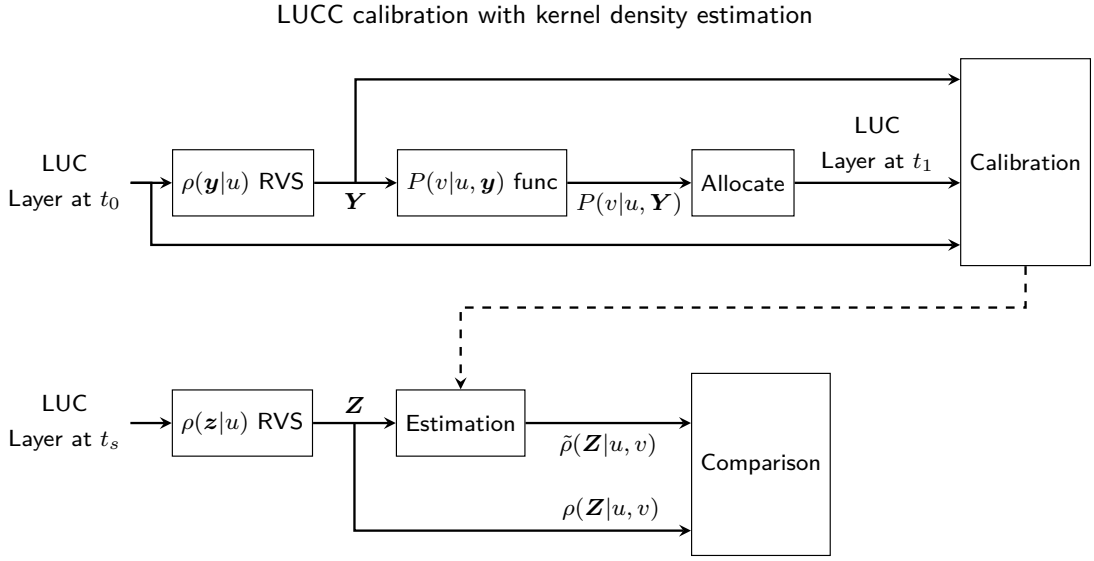


Figure 8: Comparison method of estimation models for controlled settings with artificial data. The Random Variable Sampling (RVS) process affects to each pixel in state u an explanatory variable d-tuple according the chosen probability distribution $\rho(\boldsymbol{z}|u)$.

6.1. Process

The outline of the comparison process of the different calibration methods is shown on Fig. 8 and is described below. We start with the calibration part of the process. We first produce a LUC map at time t_0, restricted to initial state u (as our calibration-estimation procedure treats initial states independently of one another). At the start of the process, such maps are "bland", i.e., they contain only u pixels with no associated explanatory variable values. As we work almost exclusively in explanatory variable space, where the spatial distribution of pixels is unessential, the only important parameter associated to such a map is in fact the number n of pixels in state u.

Next, we exactly specify the function $\rho(\boldsymbol{z}|u)$. This function must obey the closure condition Eq. (2). We then randomly affect to each pixel in state u an explanatory variable d-tuple $\boldsymbol{y}$ according to this chosen probability density[29]; the collection of d-tuples $\boldsymbol{Y}$ will be used as calibration points in explanatory variable space. The distribution $\rho(\boldsymbol{y}|u,v)$ is also exactly specified and obeys the closure condition Eq. (3). Finally, $P^*(v|u)$ is chosen in compliance with the small change hypothesis. We compute the function $P(v|u,\boldsymbol{y})$ (considered as a function of $\boldsymbol{y}$) from the inputs $\rho(\boldsymbol{y}|u)$, $\rho(\boldsymbol{y}|u,v)$ and $P^*(v|u)$ with the help of Bayes rule Eq. (1), and finally allocate to each pixel a final LUC state v according to the probability distribution $P(v|u,\boldsymbol{y})$ (we use a standard algorithm to this effect, described in appendix D). We thus obtain a LUC map at time t_1. The LUC maps at time t_0 and t_1 and the explanatory variables $\boldsymbol{Y}$ complete the elaboration of our calibration data.

We now turn to the comparison part of the comparison process, which essentially consists in the reconstruction of estimates of the inputs $\rho(\boldsymbol{z}|u)$ and $\rho(\boldsymbol{z}|u,v)$ from these data, in order to compare it to the actual input we used at the onset of the whole process. The comparison focuses on the distribution $\rho(\boldsymbol{z}|u,v)$ and its estimate, $\tilde{\rho}(\boldsymbol{z}|u,v)$, as this is the most important quantity in the whole procedure (it is the quantity most sensitive to possible undersampling, and the most important input in Bayes rule, as it characterizes the dependence of the transition probability on explanatory variables). In practice, we choose a new LUC map at time t_s (which again, for simplicity, contains the same number n of pixels in state u) and affect to each pixel in state u an explanatory variable d-tuple $\boldsymbol{z}$ according to $\rho(\boldsymbol{z}|u)$; the resulting collection $\boldsymbol{Z}$. Next, from our calibration data and efficient KDE estimation procedure, we produce the estimated probability density $\tilde{\rho}(\boldsymbol{z}|u,v)$ on the estimation points. As we do know the actual probability density on this estimation set $\rho(\boldsymbol{z}|u,v)$, we are in position to compare the actual and estimated probability densities. The comparison criteria we use are described in the next subsection.

[29]We use only normal distributions or sums of normal distributions, and this random affectation relies on existing dedicated Python algorithms.

6.2. Comparison Criteria

The quantification of the estimation error can be performed in various ways. Here, we will only use two of them: the mean absolute error, and the one-dimensional cuts. The mean absolute error magnitude ε is given by

$$\varepsilon = \frac{1}{n\overline{\rho(\boldsymbol{z}|u,v)}} \sum_{i=1}^{n} \left|\rho(\boldsymbol{z}_i|u,v) - \tilde{\rho}(\boldsymbol{z}_i|u,v)\right|, \tag{28}$$

where the sum runs on the n pixels in initial state u at t_s, and associated explanatory variable values. This allows us to measure the absolute difference between the exact prediction and the estimate. The absolute error ε is normalized by dividing by the mean value $\overline{\rho(\boldsymbol{z}|u,v)}$ to ensure that the error obtained for different numbers of explanatory variables in section 7 are comparable.

One-dimensional cuts allow us to compare the exact probability and the estimated probability by plotting the estimation on a one-dimensional line in the explanatory variable space.

These two methods of comparison may seem insufficient at first sight. Still, as will become apparent right below when applied to a specific illustration data set in section 7, we observe important variations between the models we have tested on the basis of these two criteria. Finally, we have also compared the computation time of these models on the same problem in order to estimate their numerical efficiency. Efficient algorithms are important not only when implementing a LUCC model on specific case studies when the study area is large and well-resolved, but also in order to carry out sensitivity analyses on the results by varying the problem parameters. Inefficient algorithmic implementations make such analyses prohibitive, in spite of their interest in assessing the robustness of the obtained results.

7. Illustrative Controlled Problem

Here we focus on a problem where all data are artificially created as explained in section 6. We use this problem to compare a number of LUCC modeling environments, namely, Dinamica EGO, CLUMondo, Idrisi LCM and our own allocation-estimation procedure, which is part of our complete LUCC program dubbed CLUMPY (Comprehensive Land Use [and cover] Model in PYthon, see appendix A). A single case study cannot fully judge the quality of a model. However, we have chosen a simple and general case for which the number of pixels and the number of explanatory variables can be easily parameterized (section 7.1). Section 7.2 then finds the relevant probability distributions produced by these LUCC modeling environments on this problem. We then conduct an extensive comparison of the behavior of the different methods studied by varying the problem parameters, in particular the initial number of pixels (section 7.3).

7.1. Problem Definition

The main parameters of our problem are the number of pixels n and the number of explanatory variables d, of unspecified nature; n is varied from 10^5 to 10^7 and d from 1 to 8 in order to compare the various methods that have been tested under more and more demanding conditions. Very large problems ($n \gg 10^8$) have not been explored because some of the existing methods do not converge in 24h or even fail to converge, but our own method still performs as well and fast (within minutes) for such large problems. In principle, it is possible to circumvent such convergence failures by sub-sampling the data, especially for parametric methods such as the logistic regression of CLUMondo (regressions on millions of data points are clearly unnecessary and even counter-productive). But this requires some extra pre- (and possibly post-) processing steps, and gives less precise results when non-parametric methods such as ours are used. It is therefore preferable to avoid this, if feasible. We did not try to implement such a sub-sampling for any of the calibration methods tested in order to compare them under the same conditions.

Our initial LUC map at time t_0 is uniform and has a single LUC state $u = 1$ as our method treats initial LUC states independently of one another. The probability density $\rho(\boldsymbol{z}|u)$ is specified by

$$\rho(\boldsymbol{z}|u=1) = \mathcal{N}_{\boldsymbol{\omega},\boldsymbol{\Sigma}}(\boldsymbol{z}) \tag{29}$$

where $\mathcal{N}$ is a multidimensional normal distribution; $\boldsymbol{\omega}$ is the vector of means along each dimension, of size d, and $\boldsymbol{\Sigma}$ is the covariance matrix of the distribution, of size $(d \times d)$, and whose elements are equal to 1.96

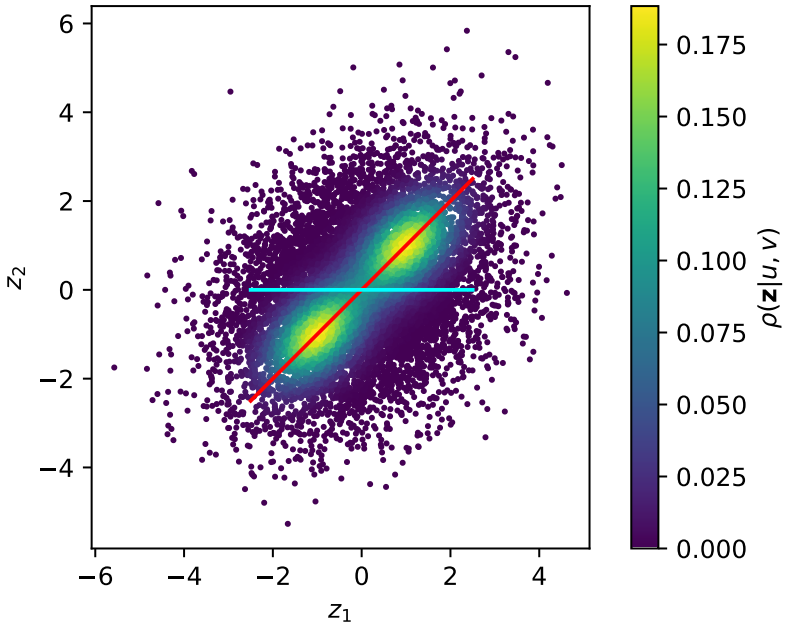


Figure 9: Representation of the probability distribution of Eq. (29) for $d = 2$. A subset of $10,000$ pixels has been randomly drawn to produce this graph. The color code corresponds to the probability density Eq. (30), considered as a function of $\boldsymbol{z}$. The red and cyan color lines refer to the one-dimensional cuts that will be used later on to compare the performances of the various LUC modeling environments.

on the diagonal and to 0.59 everywhere else. In line with our discussion on the whitening transformation (section 4.3), the vector of means $\boldsymbol{\omega} = \mathbf{0}$, so that our explanatory variables are centered but are correlated to each other. We randomly draw the explanatory variables associated to our n pixels according to this probability density.

Next, we define the probability density $\rho(\boldsymbol{z}|u,v)$:

$$\rho(\boldsymbol{z}|u,v) = \frac{1}{2}\,\mathcal{N}_{\boldsymbol{\omega}^-,\boldsymbol{\Sigma}'}(\boldsymbol{z}) + \frac{1}{2}\,\mathcal{N}_{\boldsymbol{\omega}^+,\boldsymbol{\Sigma}'}(\boldsymbol{z}), \tag{30}$$

where all the components of the d-dimensional mean vectors $\boldsymbol{\omega}^-$ and $\boldsymbol{\omega}^+$ are equal 1.0 and -1.0, respectively, and $\boldsymbol{\Sigma}'$ is the covariance matrix whose elements are equal to 0.49 on the diagonal and to 0.245 everywhere else.

Finally, we set $P^*(u|v) = 0.002$. With these definitions, the transition probability $P(v|u,\boldsymbol{z})$ is fully specified from Bayes rule, Eq (1). This allows us in turn to distribute our n calibration pixels between states v and u with the help of a simple multinomial sampling algorithm (see Appendix D). For example, with $d = 2$ (and associated explanatory variables z_0 and z_1) and $n = 800,000$, we have produced in this way a total of 1607 pixels that have transited from u to v, which is consistent with the overall transition probability $P^*(v|u) = 0.002$, considering the statistical noise due to finite numbers[30].

Fig. 9 represents a 10,000 subset of the calibration pixels in explanatory variable space. The red and cyan color lines are the chosen cuts that will be used for the one-dimensional cut comparisons of the original and reconstructed probability density distribution along these cuts for the various modeling environments we have chosen to assess, in order to have a more precise grasp of the accuracy of their calibration procedures and of their performance compared to ours.

In line with the structure displayed in Fig. 1, estimation is performed on a second sample of n pixels in initial state u. We again randomly draw their associated d-tuples of explanatory variable values according to Eq. (29), calculate the exact probability density $\rho(\boldsymbol{z}|u,v)$ on this set from Eq. (30) and use our calibration data and efficient KDE estimation procedure to produce $\tilde{\rho}(\boldsymbol{z}|u,v)$ on the same set. The use of a different

[30] We do not exactly enforce the expected number of transited pixels in the present analysis.

sample allows us to test the difference between the calibrated and exact transition probability on points in explanatory space that have not been used for calibration, and therefore to test the quality of the interpolation of each method.

We generate data for a set of parameters n and d varying from 10^5 to 10^7 and d from 1 to 8, respectively.

7.2. Transition Probability Estimation

We compare the probability density $\tilde{\rho}(\boldsymbol{z}|u,v)$ on the second set defined in the previous section, using different calibration methods: Dinamica EGO weights of evidence (Dinamica EGO WE), Idrisi LCM Multilayer Perceptron (LCM MLP), CluMondo Logistic Regression (ClUMondo LR) and our new CLUMPY efficient Kernel Density Estimation (CLUMPY eKDE) algorithm. LCM also provides a logistic regression algorithm that we prefer to avoid because of the generic poor performance of this method (no other method is proposed by CLUMondo). A poor performance is expected for all methods enforcing a specific *a priori* functional form for the probability distributions of any problem (parametric methods), as this is intrinsically less precise than non-parametric methods. The main advantage of parametric methods is that they can be used in contexts where non-parametric ones would fail due to the scarcity of data.

Dinamica EGO starts by binning explanatory variables; this binning process involves a number of parameters, that are usually left to their default choice by the user. In practice, we improved this default choice by setting the increment parameter to the Terrel bandwidth (section 4.5) multiplied by the variance of the explanatory variable (see Appendix E.1). We did find that the choice of this parameter has a substantial influence on the quality of the results, and without this rule of thumb, Dinamica EGO performs substantially less well than reported here.

For LCM MLP, we made two different series of runs. The first one kept the default parameters; these are specified in Appendix E.2. The other one aimed at improving the default behavior. In particular, the default number of neuron[31] used in the algorithm is 3. However, it is well-known in the machine learning literature that larger numbers do in general perform much better. The alternative set of parameters we have chosen for our second set of runs, noted MLP ref. in what follows, is also specified in Appendix E.2. This second parameterization with 10 neurons in the hidden layer leads to significant improvements in the performances of Idrisi LCM MLP.

CLUMondo parameters are those defined by default and are given in Appendix E.3.

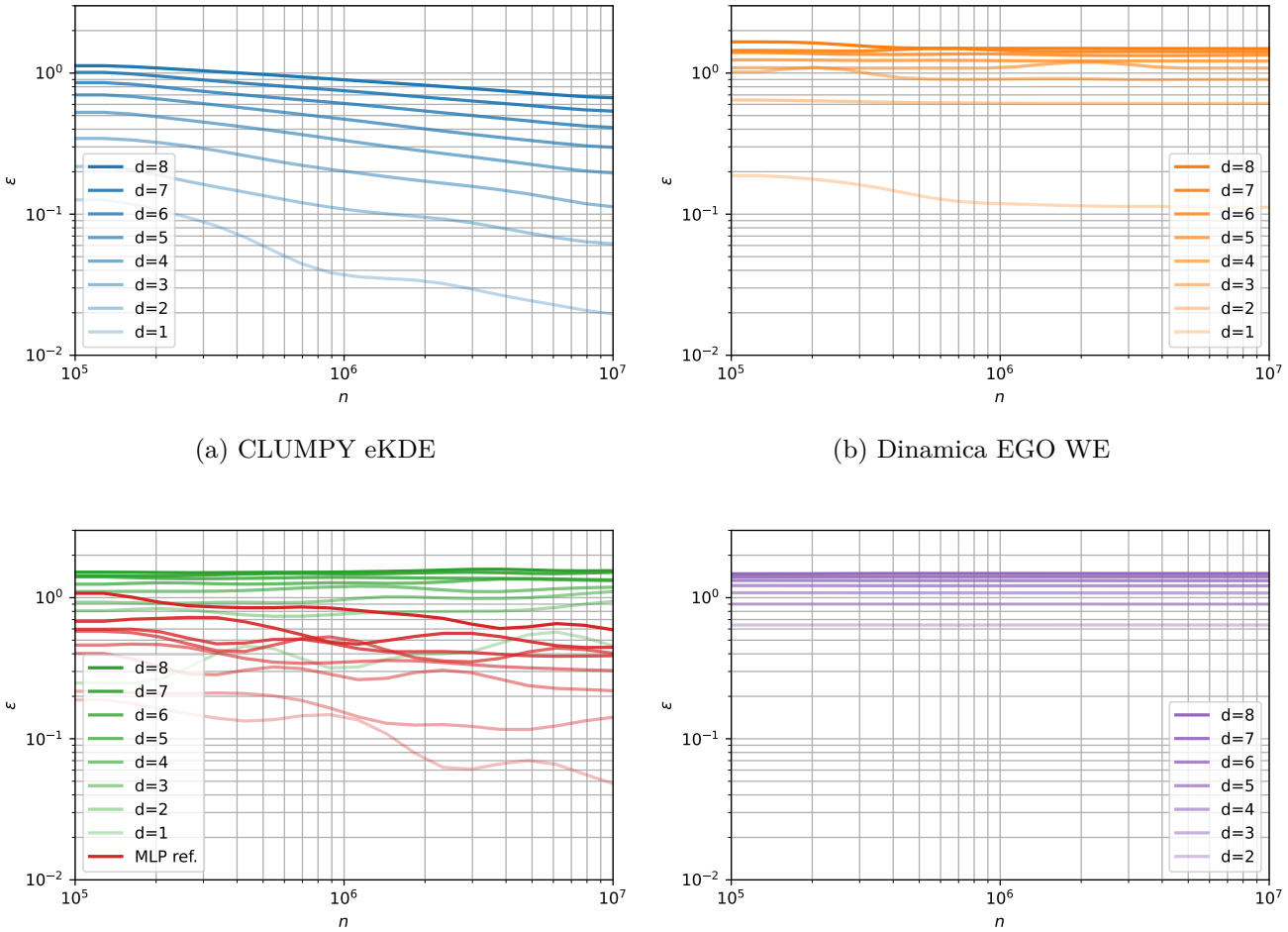
It is sometimes necessary to modify the quantities returned by the different models, as these may not be immediately comparable to the actual probability distributions. More precisely:

- Dinamica EGO does not provide an estimate $\tilde{\rho}(\boldsymbol{z}|u,v)$. Instead, from its weights of evidence, one can obtain $C\tilde{\rho}(\boldsymbol{z}|u,v)/\tilde{\rho}(\boldsymbol{z}|u)$ where C is a constant. From this and from the known $\rho(\boldsymbol{z}|u)$, one can therefore compute $C\tilde{\rho}(\boldsymbol{z}|u,v)\rho(\boldsymbol{z}|u)/\tilde{\rho}(\boldsymbol{z}|u)$, which is an estimate of $C\rho(\boldsymbol{z}|u,v)$. This estimate is then normalized to finally obtain the estimate of $\rho(\boldsymbol{z}|u,v)$.

- CLUMondo's logic is the farthest from our own of all models we have tested. CLUMondo transition probability is made of three contributions, and we have isolated the one produced from explanatory variables, which is the contribution corresponding to our calibration-estimation logic. This contribution to the transition probability to state v (for whatever state u) is proportional to a quantity that, in our notations, is a parametric logistic regression fit of $\rho(\boldsymbol{z}|v)$ at $t = t_1$ (i.e., on our second calibration map). Because we focus on a single initial state u, this quantity is thus also a parametric evaluation of $\rho(\boldsymbol{z}|u,v)$ up to a normalization factor.

- LCM's potential of change is directly proportional to $\tilde{\rho}(\boldsymbol{z}|u,v)$ so that it only needs to be normalized. The only difficulty in the use of LCM is to enforce a map that differs from the calibration maps at t_0 and t_1 for the estimation procedure. This is done by a manual substitution of the correct map at the right point in LCM's sequence of operations.

[31] In a Multi-Layer Perceptron, one can choose to parameterize a number of neurons distributed in one or more hidden layers. By default, we consider a single hidden layer (LCM choice) and the number of neurons indicated in the text corresponds to the number of neurons in this layer.



(a) CLUMPY eKDE

(b) Dinamica EGO WE

(c) LCM MLP

(d) CLUMondo LR

Figure 10: Mean Absolute Error (ε), as a function of the number of pixels n for various numbers of explanatory variables d, for the four calibration methods tested: (a) CLUMPY efficient Kernel Density Estimation, (b) Dinamica EGO Weights of Evidence, (c) Idrisi LCM Multilayer Perceptron and (d) CLUMondo Logistic Regression. In addition to LCM MLP, we also show in red lines the results of a MLP algorithm with judiciously chosen parameters for reference (MLP ref.). Note that both axis are represented in logarithmic scale, which tends to compress the differences of performance.

Based on this understanding of these modeling environment, the only operation that is finally required to obtain their estimates of $\rho(\boldsymbol{z}|u,v)$ is a normalization, in order to enforce Eq. (3). This is done along the lines of our Bayes adjustment algorithm (section 5.3), *i.e.* we rescale the quantities $\tilde{\rho}^i$ we extract from these software for each pixel i in the following way:

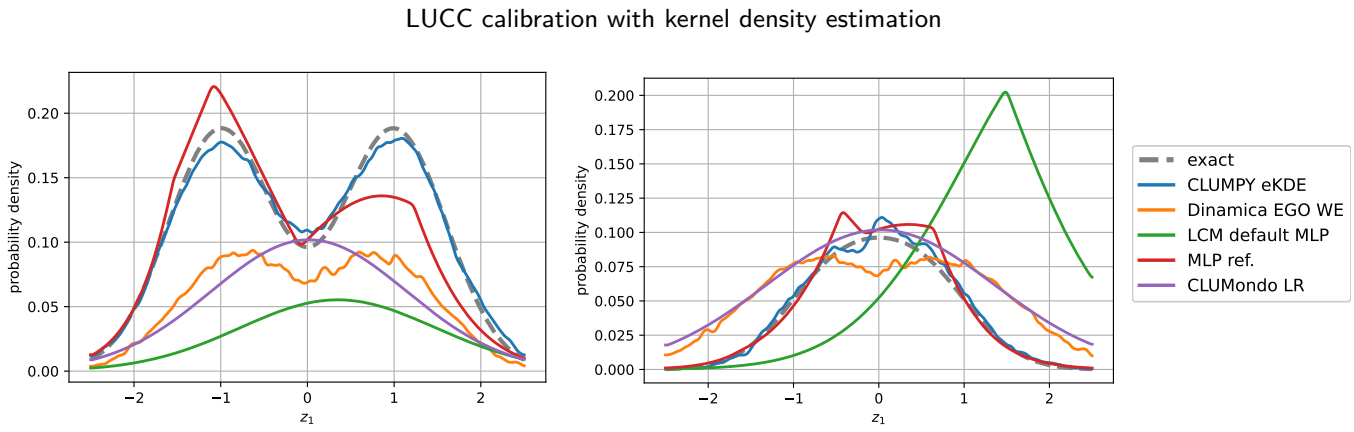
$$\tilde{\rho}(\boldsymbol{z}^i|u,v) = \frac{\tilde{\rho}^i}{\frac{1}{n}\sum_{i=1}^{n}\tilde{\rho}^i}, \tag{31}$$

Once this change of normalization is made, the estimated probability distributions can be compared to the exact probability distribution $\rho(\boldsymbol{z}|u,v)$.

7.3. Results and Discussion

The mean absolute error is calculated from Eq. (28), for various numbers d of explanatory variables (1 to 8), as a function of the number of pixels n. The results are shown on Fig. 10 for all four calibration methods. Note that the axes are represented in logarithmic scale.

The lower the value of mean absolute error ε, the more accurate the model estimates of the probability distribution $\rho(\boldsymbol{z}|u,v)$. Increasing the number of pixels increases the precision of the estimation (at constant d,



(a) red color cut (b) cyan color cut

Figure 11: Probability density estimation along the two one-dimensional lines (cuts) shown in red and cyan color on Fig. 9 with parameters $n = 800,000$ and $d = 2$, for four different calibration methods: CLUMPY efficient Kernel Density Estimation, Dinamica EGO Weights of Evidence, Idrisi LCM Multilayer Perceptron, Multilayer Perceptron with judiciously chosen parameters for reference (MLP ref.) and CLUMondo Logistic Regression.

a larger set of statistical events n reduces the noise level). Conversely, increasing the number of explanatory variables decreases it (for a constant number of pixels n, the studied space is larger if we increase d).

A more direct comparison of performance is obtained on Fig. 11, where the dependence of the transition probability on $\boldsymbol{z}$ along two one-dimensional cuts in $\boldsymbol{z}$ space is shown for $n = 800,000$ and $d = 2$; these cuts are those shown on Fig. 9. The exact transition probability is represented by the grey dashed line.

Finally, computation times in minutes are shown for each calibration method on Fig. 12 (note that the abscissa is represented in logarithmic scale).

More specific comments on the various methods are provided in the remainder of this section.

7.3.1. CLUMPY efficient Kernel Density Estimation

CLUMPY eKDE performs better to significantly better than all other methods. Interestingly, our method keeps improving for increasing n at a given d, while the optimized LCM MLP seems to saturate for a few 10^6 pixels (this however may be an effect of the choice of the number of nodes in this method, or a temporary "plateau" in n).

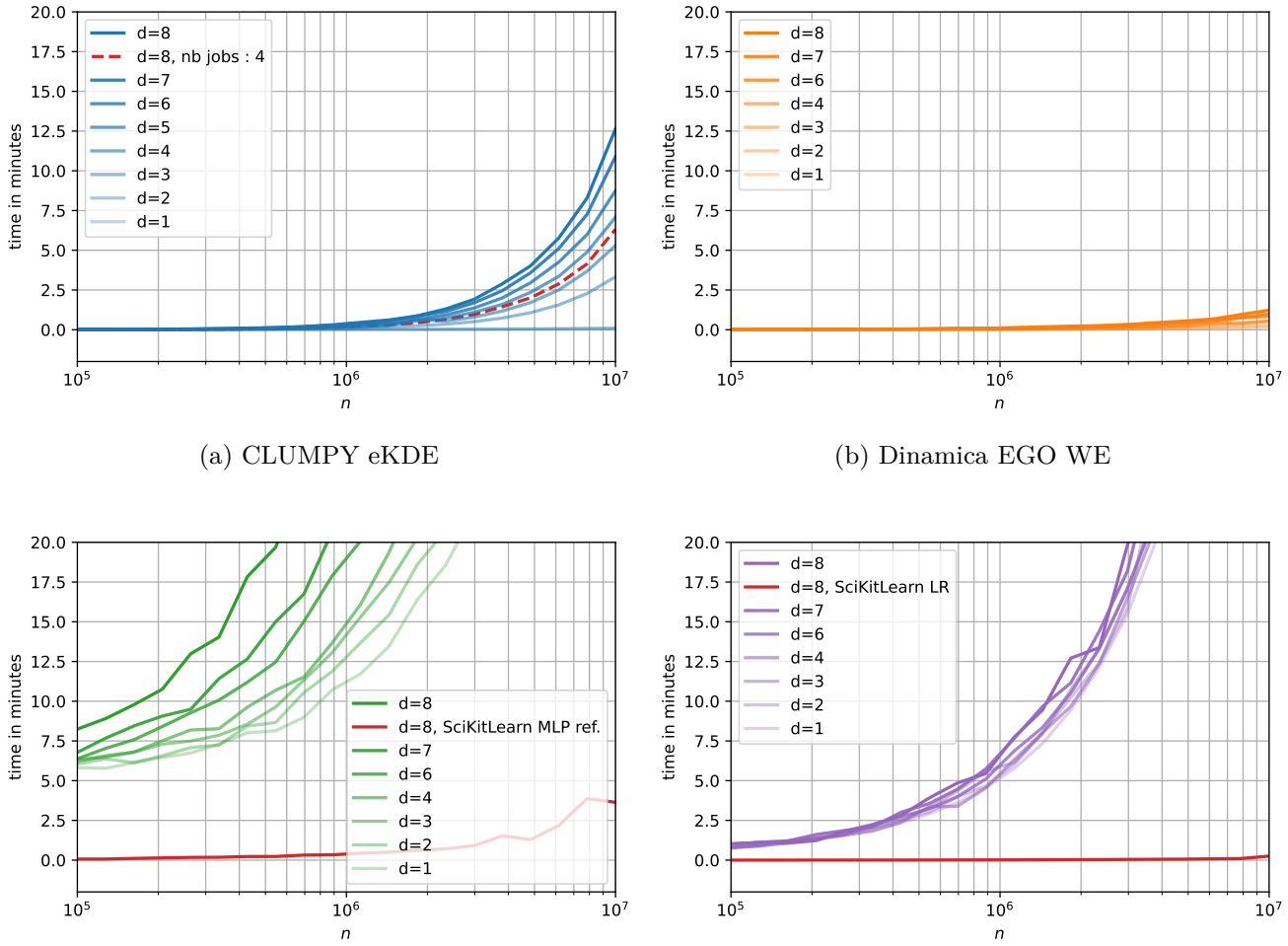
Also, the computation time is of the order of a few minutes if we stick to a reasonable number of explanatory variables (lower or equal to 6) and this remark is even more valid if we parallelize the computation (Fig. 12a).

7.3.2. Idrisi LCM MLP

The multilayer perceptron algorithm has interesting characteristics if one does not stick to the default parameters recommended by the developers.

Let us first focus on the results obtained for the default parameters, shown in green on figures 10c and 11. Increasing the number of dimensions reduces the accuracy as expected. However, increasing the number of pixels does not bring any significant improvement (Fig. 10c). Moreover, the estimator is unable to produce an acceptable approximation of the exact probability (Fig. 11). Finally, the computation time may become prohibitive: it peaks at 5h20 with 8 explanatory variables and 10 million pixels (out of range in Fig. 12c).

In red, we represent the results obtained with the same algorithm (multilayer perceptron) but with better chosen parameters (see Appendix E for a detailed description of the default and non-default parameter choices). We have used the Python open-source package SciKitLearn to produce the MLP ref. result. The improvement shown by the red curve in Fig. 10c and 11 was obtained by increasing the number of neurons (nodes) from 3 (LCM choice) to 10 (within a single hidden layer in both cases). The related average absolute deviation is even comparable to that returned by CLUMPY (Fig. 10a). Moreover, Fig. 11 shows that the result is much more precise than all other tested models, except our own CLUMPY eKDE algorithm, which is somewhat more precise (although increasing further the number of nodes in the hidden layer reduces the gap in performance). Finally, the MLP algorithm implemented by SciKitLearn leads to reasonable



(a) CLUMPY eKDE

(b) Dinamica EGO WE

(c) LCM MLP

(d) CLUMondo LR

Figure 12: Computation times of different calibration methods as a function of the number of pixels n, for various numbers of explanatory variables d: CLUMPY efficient Kernel Density Estimation, Dinamica EGO Weights of Evidence, Idrisi LCM Multilayer Perceptron (peaks outside the window at 5h20 for $d = 8$) and CLUMondo Logistic Regression (peaks outside the window at 1h for $d = 8$). Also, the computation time of CLUMPY KDE with 4 parallel jobs for $d = 8$ is represented. Note that the abscissa is represented in logarithmic scale.

computation times, similar to those of CLUMPY (figure 12c).

This being said, the MLP algorithm has a significant disadvantage: as for nearly all other artificial intelligence algorithms relying on neuron networks, it is impossible to understand and analyze how the model calibration is actually performed. Also, as will be shown in our dedicated allocation paper, LCM incorrectly uses $\rho(\boldsymbol{z}|u, v)$ for $P(v|u, \boldsymbol{z})$ for allocation. Still, considering the interest of a well-designed MLP algorithm, it has been implemented in our package as an alternative to our own eKDE algorithm.

7.3.3. Dinamica EGO Weight of Evidence

Dinamica EGO performs rather poorly, both on mean absolute error and clearly so on one-dimensional cuts. However, we point out that it is the next best alternative after CLUMPY eKDE and our optimized version of the MLP algorithm. A univariate problem ($d = 1$) may be the exception (Fig. 10b). Fig. 11b highlights Dinamica EGO's choice to consider the explanatory variables as independent: this leads to the existence of two maxima, whereas only one is present in this particular cut.

The speed of Dinamica EGO calibration method is worth pointing out (Fig. 12b). This is due to the hypothesis of statistical independence of the variables, so that the calibration procedure amounts to d times a one-dimensional calibration; conversely, CLUMPY eKDE refrains from making such an assumption, which makes the calibration problem truly d-dimensional and significantly more demanding numerically. Dinamica

EGO algorithmic efficiency comes out at the expense of the precision of the estimation.

It is also worth remembering that the parameterization adopted for Dinamica EGO is not the default one, but has been set according to a rule of thumb inspired by the KDE bandwidth (see Appendix E.1). Dinamica EGO does not provide any indication as to how the binning parameters should be chosen (i.e., the parameters used in the method specifying the bin size), while we found that their effect on the quality of the results was substantial.

7.3.4. CLUMondo Logistic Regression

CluMondo LR uses a parametric estimator based on logistic regressions, and is therefore strongly constrained. Quite clearly CLUMondo performs extremely poorly, on both criteria (Fig. 10d and Fig. 11). CLUMondo returns a quasi-constant probability distribution on the whole space of the explanatory variables. However, as indicated in section 7.2, we multiply the result obtained by the probability density of the explanatory variables $\rho(\boldsymbol{z}|u)$ in order to produce *in fine* $\rho(\boldsymbol{z}|u,v)$ and this is why the shape of the curve obtained corresponds more or less to $\rho(\boldsymbol{z}|u)$.

Dinamica EGO WE, LCM MLP as well as our CLUMPY KDE algorithm are non-parametric models and are more adapted to estimate a large panel of probability density functional forms. As pointed out earlier, the choice of a parametric method may be adapted when data are scarce, and indeed, CLUMondo makes use of a single land use map to identify suitable transition sites. For all practical purposes, CLUMondo uses the resulting pixel suitability scores as transition probabilities. As by definition no transition can be observed with a single map, this requires an extra assumption, namely, in our language, that $\rho(\boldsymbol{z}|v)$ and $\rho(\boldsymbol{z}|u,v)$ are proportional, and that the transition probability $P(v|u,\boldsymbol{z})$ is proportional to $\rho(\boldsymbol{z}|v)$ for all initial states u.

However, it is clear from Bayes rule [Eq. (1)] that this cannot be true, as this relation implies that $P(v|u,\boldsymbol{z}) \propto \rho(\boldsymbol{z}|u,v)/\rho(\boldsymbol{z}|u)$. Even for single initial state u for which $\rho(\boldsymbol{z}|u,v) = \rho(\boldsymbol{z}|v)$, the probability is proportional to $\rho(\boldsymbol{z}|v)$ only if $\rho(\boldsymbol{z}|u)$ is uniform. These are very restrictive conditions. Finally, the parametric method of estimation of $\rho(\boldsymbol{z}|u,v) = \rho(\boldsymbol{z}|v)$ (for a unique u) requires that the probability density is maximum at $\boldsymbol{z} = 0$. This is also violated by our choice, Eq. (30). Such a violation may be justified. Consider for example the transition between different species of plants with climate change. In a mountainous region, the change will clearly depend on elevation for any particular species, and has no reason to be maximum at the lowest elevation.

Finally, the computation time is d independent but surprisingly high for a parametric method with a 1h running time for $d = 8$ and $n = 10^7$ (outside the window in Fig. 12d). For comparison purposes, we represent in the same figure the time needed for $d = 8$ using the much more efficient python package SciKitLearn.

8. Conclusion

This article constitutes the first part of an investigation dedicated to the analysis of the conceptual biases, inaccuracies and inefficiencies of existing pattern-based LUCC software and models. Our objective is to propose systematic improvements in both calibration and allocation (the main building blocks in this LUCC approach), based on mathematically rigorous analyses of all aspects of the problem, and on efficient algorithmic implementations of the results of these analyses. This in turn will allow us to understand and reduce in a systematic way the difference of results obtained by different modeling environments on any given problem and set of data, a point discussed in the introduction. This series also describes in detail our LUCC model named CLUMPY (Comprehensive Land Use [and cover] Model in PYthon).

We focus here on the calibration of transition probabilities for this type of LUCC models. We introduce a new calibration method based on Bayes rule Eq. (1) and on an adaptation of a well-known machine learning method — kernel density estimation (KDE) — referred to as eKDE in the bulk of this paper (the full calibration-estimation method is abbreviated as Bayes-eKDE). We produced a series of artificial case studies from completely specified probability distributions, in order to compare the performance of this method to the calibration methods implemented in existing software, namely Dinamica EGO, CLUMondo and Idrisi LCM MLP (Multi-Layer Perceptron).

Our calibration-estimation algorithm involves a rather complex series of sub-tasks (whitening transformation, section 4.3; boundary bias correction, section 4.4; bandwidth selection, section 4.5). Moreover, its implementation requires other procedures to optimize the execution time (efficient KDE method, section 5.1;

parallelization, section 5.1.3) and to avoid numerical outliers (Bayes adjustment process, section 5.3). The architecture of these steps is presented in section 5.2. sections 4 and 5 present a self-contained and exhaustive description of this new method (in the LUCC modeling context) for estimating transition probabilities, summarized in Fig. 2.

We also define a new evaluation procedure for calibration methods (section 6) where all probabilities involved are known exactly. This context allows us to implement objective evaluation methods of the accuracy of the transition probabilities estimated with exiting calibration procedures, and compare different methods of calibration. For now, only simple comparison criteria are introduced (mean absolute error, one-dimensional cut and computation time) but the proposed evaluation framework is compatible with other comparison methods.

This evaluation procedure is not implemented in physical space, but in explanatory variable space. This is at variance with common validation practices in quantitative geography in general and in pattern-based LUCC modeling in particular. This choice has been motivated in the discussion of section 2, but the main points may be worth rephrasing here. Actual spatial locations of allocations in this type of modeling is statistical by design, i.e., many statistically equivalent sites may potentially be chosen for the same LUC state change during a time interval, but only a few may actually be observed to undergo such a change. The same argument goes for calibration: many other sites for state change could have been chosen instead of the observed ones, with equivalent statistical properties in terms of explanatory variables distribution. The only way to circumvent this feature would be to have large amounts of state change during a reference period (so that the statistical spread is reduced by necessity). There is therefore little reason to expect spatial location to be an appropriate measure of LUCC modeling framework relevance, although this is a desirable property of any LUCC framework instantiation in any case study. On the other hand, producing a robust evaluation of distribution in explanatory space is mandatory, as this is precisely what controls future allocation in a simulation.

This evaluation makes use of a completely controlled test problem (section 7). This relies on data created from completely specified transition probabilities. In this setting, we have varied both the total number of pixels (10^4 to 10^6) and explanatory variables (1 to 8) in order to perform our performance comparison under a wide range of conditions. We compare in this way exiting calibration procedures (Idrisi LCM MLP, Dinamica EGO and CLUMondo) to the CLUMPY eKDE method introduced in this work (section 7.3). The probability distributions estimated by our CLUMPY eKDE algorithm turn out to be substantially more accurate than the alternative we have tested, whatever the number of explanatory variables and the number of pixels. Also, it appears to be efficient in term of running time, especially if we increase the number of cores dedicated to this calculation; however, we found that an appropriate parameter choice for the MLP calibration method leads to satisfactory results as well. It would be useful to reproduce this evaluation on a larger and more varied number of test problems, but we are confident that our conclusions would stand.

We focused on the accuracy of the estimation and the computation time of the different models. We did not linger over the user experience provided. Without going into details, the configuration of a LUCC case study is often either tedious or constrained in existing software. CLUMondo offers very little flexibility, the default parameters of Idrisi LCM MLP are poorly chosen (section 7.3.2) and Dinamica EGO, although very versatile, gives very little indication on the choice of some parameters that are critical to the quality of the results; the versatility comes at the expense of a steeper learning curve. CLUMPY has its own learning curve. It is programmed in a very common language (Python), but programming is an issue in the LUCC modeling community, where most users have little programming experience. We are presently planning to circumvent this problem through the creation of a simple Graphic User Interface (GUI) for our complete modeling environment CLUMPY. Moreover, there is only one parameter for the transition probability estimation: q, the small-scale binning parameter of our efficient KDE implementation (section 5.1.1). From our tests, the recommended default value of this parameter seems to be appropriate in most cases.

This work exemplifies the strategy we adopt in our work on LUCC modeling. On the one hand, we aim at a higher level of mathematical and algorithmic rigor. On the other hand, we devise specifically tailored tests with the aim of achieving a more precise and in-depth comparison of computational performances, conceptual and algorithmic correctness and accuracy than existing methods. In the present work, for example, focusing on comparison in explanatory variable space instead of physical space while using artificial data allows us to benefit from a complete control on the comparisons performed, and to be much more discriminant on this

front.

References

Agarwal, C., Green, G.L., Grove, M., Evans, T., Schweik, C., 2000. A review and assessment of land-use change models: dynamics of space, time and human choice. Gen. Tech. Rep. NE-297. Newton Square, PA: U.S. Department of Agriculture, Forest Service, Northeastern Research Station. 61 p. doi:doi:10.2737/NE-GTR-297.

Alexander, P., Prestele, R., Verburg, P.H., Arneth, A., Baranzelli, C., Batista e Silva, F., Brown, C., Butler, A., Calvin, K., Dendoncker, N., Doelman, J.C., Dunford, R., Engström, K., Eitelberg, D., Fujimori, S., Harrison, P.A., Hasegawa, T., Havlik, P., Holzhauer, S., Humpenöder, F., Jacobs-Crisioni, C., Jain, A.K., Krisztin, T., Kyle, P., Lavalle, C., Lenton, T., Liu, J., Meiyappan, P., Popp, A., Powell, T., Sands, R.D., Schaldach, R., Stehfest, E., Steinbuks, J., Tabeau, A., van Meijl, H., Wise, M.A., Rounsevell, M.D.A., 2017. Assessing uncertainties in land cover projections. Global Change Biology 23, 767–781. doi:doi:10.1111/gcb.13447.

van Asselen, S., Verburg, P.H., 2013. Land cover change or land-use intensification: simulating land system change with a global-scale land change model. Global Change Biology 19, 3648–3667. doi:doi:10.1111/gcb.12331.

Backurs, A., Indyk, P., Wagner, T., 2019. Space and time efficient kernel density estimation in high dimensions, in: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (Eds.), Advances in Neural Information Processing Systems, Curran Associates, Inc.. pp. 15773–15782.

Bielecka, E., 2020. Gis spatial analysis modeling for land use change. a bibliometric analysis of the intellectual base and trends. Geosciences 10, 421. doi:doi:10.3390/geosciences10110421.

Camacho Olmedo, M.T., Paegelow, M., Mas, J.F., Escobar, F., 2018. Geomatic Approaches for Modeling Land Change Scenarios. An Introduction, in: Camacho Olmedo, M.T., Paegelow, M., Mas, J.F., Escobar, F. (Eds.), Geomatic Approaches for Modeling Land Change Scenarios. Springer International Publishing. Lecture Notes in Geoinformation and Cartography, pp. 1–8. doi:doi:10.1007/978-3-319-60801-3_1.

Chacón, J.E., Duong, T., 2018. Multivariate Kernel Smoothing and its Applications. Chapman and Hall/CRC. doi:doi:10.1201/9780429485572.

Charikar, M., Siminelakis, P., 2017. Hashing-Based-Estimators for Kernel Density in High Dimensions, in: 2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS), pp. 1032–1043. doi:doi:10.1109/FOCS.2017.99.

Cho, Y., Kim, S., 2020. Volume of Hypercubes Clipped by Hyperplanes and Combinatorial Identities. The Electronic Journal of Linear Algebra 36, 228–255. doi:doi:10.13001/ela.2020.5085.

Diggle, P., 1985. A Kernel Method for Smoothing Point Process Data. Journal of the Royal Statistical Society. Series C (Applied Statistics) 34, 138–147. doi:doi:10.2307/2347366.

Duong, T., Hazelton, M., 2003. Plug-in bandwidth matrices for bivariate kernel density estimation. Journal of Nonparametric Statistics 15, 17–30. doi:doi:10.1080/10485250306039.

Duong, T., Hazelton, M.L., 2005. Cross-validation Bandwidth Matrices for Multivariate Kernel Density Estimation. Scandinavian Journal of Statistics 32, 485–506. doi:doi:10.1111/j.1467-9469.2005.00445.x.

Eastman, J.R., Jin, W., Kyem, P., Toledano, J., 1995. Raster Procedure for Multi-Criteria/Multi-Objective Decisions. Photogrammetric Engineering & Remote Sensing 61, 539–547.

García-Álvarez, D., Camacho Olmedo, M.T., Van Delden, H., Mas, J.F., Paegelow, M., 2022. Comparing the structural uncertainty and uncertainty management in four common land use cover change (lucc) model software packages. Environmental Modelling & Software 153, 105411. doi:doi:https://doi.org/10.1016/j.envsoft.2022.105411.

Kessy, A., Lewin, A., Strimmer, K., 2018. Optimal Whitening and Decorrelation. The American Statistician 72, 309–314. doi:doi:10.1080/00031305.2016.1277159.

Knuth, D.E., 1998. The art of computer programming. Volume 2, seminumerical algorithms, 3rd edition. Addison-Wesley.

Langrené, N., Warin, X., 2019. Fast and Stable Multivariate Kernel Density Estimation by Fast Sum Updating. Journal of Computational and Graphical Statistics 28, 596–608. doi:doi:10.1080/10618600.2018.1549052.

Longaretti, P.Y., Mazy, F.R., 2022. Towards a Generic Theoretical Framework for Pattern-Based LUCC Modeling. A maximum relevance / minimum redundancy selection procedure of explanatory variables. Submitted to Environmental Software and Modeling .

Mas, J., García Álvarez, D., Paegelow, M., Domínguez-Vera, R., Castillo-Santiago, M., 2022. Metrics Based on a Cross-Tabulation Matrix to Validate Land Use Cover Maps, in: Land Use Cover Datasets and Validation Tools. Validation Practices with QGIS. Springer, Cham, pp. 127–151. doi:doi:10.1007/978-3-030-90998-7_8.

Mas, J.F., Kolb, M., Paegelow, M., Camacho Olmedo, M.T., Houet, T., 2014. Inductive pattern-based land use/cover change models: A comparison of four software packages. Environmental Modelling & Software 51, 94–111. doi:doi:10.1016/j.envsoft.2013.09.010.

Mazy, F.R., Longaretti, P.Y., 2022a. A Formally Correct and Algorithmically Efficient LULC Change Model-building Environment, in: Proceedings of the 8th International Conference on Geographical Information Systems Theory, Applications and Management - GISTAM, SciTePress. pp. 25–36.

Mazy, F.R., Longaretti, P.Y., 2022b. Towards a Generic Theoretical Framework for Pattern-Based LUCC Modeling. Allocation Revisited: Formal foundations and bias identification. Submitted to Environmental Software and Modeling .

Noszczyk, T., 2019. A review of approaches to land use changes modeling. Human and Ecological Risk Assessment: An International Journal 25, 1377–1405. doi:doi:10.1080/10807039.2018.1468994.

OBrien, T.A., Kashinath, K., Cavanaugh, N.R., Collins, W.D., OBrien, J.P., 2016. A fast and objective multidi-

mensional kernel density estimation method: fastKDE. Computational Statistics & Data Analysis 101, 148–160. doi:doi:10.1016/j.csda.2016.02.014.

Phillips, J.M., 2013. ϵ-samples for kernels, in: Proceedings of the 2013 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), Society for Industrial and Applied Mathematics. pp. 1622–1632. doi:doi:10.1137/1.9781611973105.116.

Phillips, J.M., Tai, W.T., 2018a. Improved coresets for kernel density estimates, in: Proceedings of the 2018 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 2718–2727. doi:doi:10.1137/1.9781611975031.173.

Phillips, J.M., Tai, W.T., 2018b. Near-Optimal Coresets of Kernel Density Estimates, in: Speckmann, B., D., T.C. (Eds.), 34th International Symposium on Computational Geometry (SoCG 2018), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. pp. 66:1–66:13. doi:doi:10.4230/LIPIcs.SoCG.2018.66.

Prestele, R., Alexander, P., Rounsevell, M.D.A., Arneth, A., Calvin, K., Doelman, J., Eitelberg, D.A., Engström, K., Fujimori, S., Hasegawa, T., Havlik, P., Humpenöder, F., Jain, A.K., Krisztin, T., Kyle, P., Meiyappan, P., Popp, A., Sands, R.D., Schaldach, R., Schüngel, J., Stehfest, E., Tabeau, A., Van Meijl, H., van Vliet, J., Verburg, P.H., 2016. Hotspots of uncertainty in land-use and land-cover change projections: a global-scale model comparison. Global Change Biology 22, 3967–3983. doi:doi:10.1111/gcb.13337.

Rudemo, M., 1982. Empirical Choice of Histograms and Kernel Density Estimators. Scandinavian Journal of Statistics 9, 65–78.

Sain, S.R., Baggerly, K.A., Scott, D.W., 1994. Cross-validation of multivariate densities. Journal of the American Statistical Association 89, 807–817. doi:doi:10.1080/01621459.1994.10476814.

Schindler, A., 2011. Bandwidth Selection in Nonparametric Kernel Estimation. Ph.D. thesis. Faculty of Economic Sciences of the Georg-August-Universität Göttingen.

Scott, D., 2015. Multivariate density estimation: Theory, practice, and visualization: Second edition. John Wiley. doi:doi:10.1002/9781118575574.

Siminelakis, P., Rong, K., Bailis, P., Charikar, M., Levis, P., 2019. Rehashing Kernel Evaluation in High Dimensions, in: International Conference on Machine Learning, PMLR. pp. 5789–5798.

Soares-Filho, B.S., Coutinho Cerqueira, G., Lopes Pennachin, C., 2002. Dinamica — A stochastic cellular automata model designed to simulate the landscape dynamics in an Amazonian colonization frontier. Ecological Modelling 154, 217–235. doi:doi:10.1016/S0304-3800(02)00059-5.

Tai, W.M., 2022. Optimal Coreset for Gaussian Kernel Density Estimation, in: Goaoc, X., Kerber, M. (Eds.), 38th International Symposium on Computational Geometry (SoCG 2022), Schloss Dagstuhl – Leibniz-Zentrum für Informatik. pp. 63:1–63:15. doi:doi:10.4230/LIPIcs.SoCG.2022.63.

Terrell, G., 1990. The Maximal Smoothing Principle in Density Estimation. Journal of the American Statistical Association 85, 470–477. doi:doi:10.1080/01621459.1990.10476223.

Verburg, P.H., Alexander, P., Evans, T., Magliocca, N.R., Malek, Z., Rounsevell, M.D.A., van Vliet, J., 2019. Beyond land cover change: towards a new generation of land use models. Current Opinion in Environmental Sustainability 38, 77–85. doi:doi:10.1016/j.cosust.2019.05.002.

Verburg, P.H., Kok, K., Pontius, R.G., Veldkamp, A., 2006. Modeling Land-Use and Land-Cover Change, in: Lambin, E.F., Geist, H. (Eds.), Land-Use and Land-Cover Change. Global Change - The IGBP Series. Springer. Global Change - The IGBP Series, pp. 117–135. doi:doi:10.1007/3-540-32202-7_5.

Verburg, P.H., de Koning, G.H.J., Kok, K., Veldkamp, A., Bouma, J., 1999. A spatial explicit allocation procedure for modelling the pattern of land use change based upon actual land use. Ecological Modelling 116, 45–61. doi:doi:10.1016/S0304-3800(98)00156-2.

van Vliet, J., Bregt, A.K., Brown, D.G., van Delden, H., Heckbert, S., Verburg, P.H., 2016. A review of current calibration and validation practices in land-change modeling. Environmental Modelling & Software 82, 174–182. doi:doi:10.1016/j.envsoft.2016.04.017.

van Vliet, J., Malek, Z., Verbug, P., 2015. The CLUMondo land use change model, manual and exercises.

Wand, M., Jones, C., 1994. Multivariate plug-in bandwidth selection. Computational Statistics 9, 97–116.

Wand, M.P., 1992. Error analysis for general multivariate kernel estimators. Journal of Nonparametric Statistics 2, 1–15. doi:doi:10.1080/10485259208832538.

Wells, J.R., Ting, K.M., 2019. A new simple and efficient density estimator that enables fast systematic search. Pattern Recognition Letters 122, 92–98. doi:doi:10.1016/j.patrec.2018.12.020.

Zheng, Y., Jestes, J., Phillips, J.M., Li, F., 2013. Quality and efficiency for kernel density estimates in large data, in: SIGMOD '13: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pp. 433–444. doi:doi:10.1145/2463676.2465319.

Appendix

A. Software

In this article we introduce our own environment for LUCC modeling, CLUMPY (Comprehensive Land Use [and cover] Model in PYthon). As indicated by the name, this is a Python package which provides a complete framework that includes our own calibration, estimation and allocation algorithms, as well as

an optimized semi-automatic method of selection of explanatory variables. This open-source software is provided under a GNU public license and is freely available on the GitLab of our institution: `https://gitlab.inria.fr/fmazy/clumpy/` (no username or password required). It is currently developed by François-Rémi Mazy and a beta version is being tested as part of the work as presented here. A first stable version along with its documentation is planned for the fall of 2022 and a Graphical User Interface (GUI) will be also provided at the same time. No specific hardware is required and CLUMPY aims to be user-friendly and usable on any computer, although it has only been tested on Linux OS at the time of writing. The software is mainly written in Python, but some critical elements such as the eKDE method presented in this work have been written in Cython to improve computation efficiency. The whole program with its documentation weighs 477kb (zip file) at the time of writing.

B. A more formal definition of probability densities

Continuous variables probabilities are only properly defined on finite intervals of the variables values. Formally, these probabilities are obtained from a so-called probability density that is integrated over these intervals to obtain the required probability. For a univariate continuous random variable z, one can show that there exists a unique function $\rho(z)$ called probability density such that the probability $P(z, \Delta z)$ of obtaining the value z within the range $[z, z + \Delta z]$ is given by

$$P(z, \Delta z) = \int_{z}^{z+\Delta z} \rho(z) dz. \tag{B1}$$

Equivalently, the probability density is defined by an appropriate partial derivative of P:

$$\rho(z) = \frac{\partial P}{\partial \Delta z}, \tag{B2}$$

where the partial derivative is evaluated at $\Delta z = 0$.

This definition is extended to multivariate random variables $\boldsymbol{z} = (z_1, z_2, ..., z_d)$ through the use of a multivariable probability density $\rho(\boldsymbol{z})$ and the probability is obtained by a multiple integral:

$$P(\boldsymbol{z}, \Delta \boldsymbol{z}) = \int_{z_1}^{z_1+\Delta z_1} dz_1 \int_{z_2}^{z_2+\Delta z_2} dz_2 ... \int_{z_d}^{z_d+\Delta z_d} dz_d \rho(\boldsymbol{z}). \tag{B3}$$

The derivative expression of the probability density is generalized in the following way:

$$\rho(\boldsymbol{z}) = \lim_{\Delta z_i \to 0} \frac{P(\boldsymbol{z}, \Delta \boldsymbol{z})}{\Delta z_1 \Delta z_2 ... \Delta z_d}. \tag{B4}$$

C. Extending the eKDE calibration procedure to a mix of discrete and continuous explanatory variables

We distinguish continuous explanatory variables z_k and discrete ones $\gamma_{k'}$ ($1 \leq k \leq d$ and $1 \leq k' \leq d'$). A range $\Gamma_{k'}$ of possible integer values is associated to each explanatory variable $\gamma_{k'}$. From these quantities, one can form an arbitrary t-uple of discrete variables $\boldsymbol{\gamma} = (\gamma_1, \gamma_2, ..., \gamma_{d'})$.

The probability distribution of the discrete variables $P(\boldsymbol{\gamma}|u)$ and $P(\boldsymbol{\gamma}|u, v)$ are straightforwardly obtained from pixel counting on calibration maps. The number of pixels with discrete explanatory variables $\boldsymbol{\gamma}$ and initial state u at t_0 is noted $n_{\boldsymbol{\gamma},u}$ while $n_{\boldsymbol{\gamma},u,v}$ is the number of these pixels having undergone a LUC state transition $u \to v$ between t_0 and t_1 in the calibration maps. Recalling that n_{I_v} is the number of pixels in state u at t_0 and state v at t_1 and defining n_I as the number of pixels in state u at t_0, one has

$$P(\boldsymbol{\gamma}|u, v) = \frac{n_{\boldsymbol{\gamma},u,v}}{n_{I_v}}, \tag{C1}$$

$$P(\boldsymbol{\gamma}|u) = \frac{n_{\boldsymbol{\gamma},u}}{n_I}. \tag{C2}$$

Let us now consider a mixed set of discrete and continuous explanatory variables. One has (Bayes rule)

$$P(v|u,\boldsymbol{z},\boldsymbol{\gamma}) = P^*(v|u)\frac{\rho(\boldsymbol{z},\boldsymbol{\gamma}|u,v)}{\rho(\boldsymbol{z},\boldsymbol{\gamma}|u)} = P^*(v|u) \times \frac{P(\boldsymbol{\gamma}|u,v)}{P(\boldsymbol{\gamma}|u)} \times \frac{\rho(\boldsymbol{z}|u,v,\boldsymbol{\gamma})}{\rho(\boldsymbol{z}|u,\boldsymbol{\gamma})}. \tag{C3}$$

The first factor is provided by the user (or from calibration data if one is only interested in extrapolations of past trends), the second factor is calibrated as specified above. Therefore one just has to calibrate $\rho(\boldsymbol{z}|u,v,\boldsymbol{\gamma})$ and $\rho(\boldsymbol{z}|u,\boldsymbol{\gamma})$.

For completeness let us indicate how the closure relations are modified when discrete variables are taken into account for the quantities that are calibrated in a mixed continuous/discrete variables context:

$$\int_{\boldsymbol{z}\in\mathbb{R}^d} \tilde{\rho}(\boldsymbol{z}|u,\boldsymbol{\gamma})d\boldsymbol{z} = 1, \tag{C4}$$

$$\int_{\boldsymbol{z}\in\mathbb{R}^d} \tilde{\rho}(\boldsymbol{z}|u,v,\boldsymbol{\gamma})d\boldsymbol{z} = 1, \tag{C5}$$

$$\sum_{\boldsymbol{\gamma}} \tilde{P}(\boldsymbol{\gamma}|u) \int_{\boldsymbol{z}\in\mathbb{R}^d} \tilde{P}(v|u,\boldsymbol{z},\boldsymbol{\gamma}) \times \tilde{\rho}(\boldsymbol{z}|u,\boldsymbol{\gamma})d\boldsymbol{z} = \int_{\boldsymbol{z}\in\mathbb{R}^d} \tilde{\rho}(v,\boldsymbol{z}|u)d\boldsymbol{z} = P^*(v|u). \tag{C6}$$

For a given transition $u \to v$, taking into account discrete variables reduces to applying our eKDE method for each possible $\boldsymbol{\gamma}$. Formally, this amounts to substitute $\rho(\boldsymbol{z}|u,v,\boldsymbol{\gamma})$ to $\rho(\boldsymbol{z}|u,v)$, $\rho(\boldsymbol{z}|u,\boldsymbol{\gamma})$ to $\rho(\boldsymbol{z}|u)$ and $P(v|u,\boldsymbol{z},\boldsymbol{\gamma})$ to $P(v|u,\boldsymbol{z})$ (and similarly for distributions expressed in terms of $\boldsymbol{y}$ instead of $\boldsymbol{z}$) in all expressions from section 4 onward. The updated closure relations and updated Bayes rule above must be used where relevant instead of the original ones.

At first sight this calibration strategy seems computationally demanding. It is in fact much less demanding than making two eKDE calibrations for $d + d'$ continuous variables, although this is clearly more demanding than making two calibrations for d continuous variables.

D. Multinomial Sampling Test algorithm

The algorithm presented here is well-known (see, e.g., Knuth 1998, section 3.4.1). Let be a pixel and the set of transition probabilities $P(v|u,\boldsymbol{z})$ for all v. Let us define η_p as the cumulative sum of $P(v|u,\boldsymbol{z})$:

$$\eta_0 = 0, \tag{D1}$$

$$\forall k, \quad \eta_k = \sum_{v\leq k} P(v|u,\boldsymbol{z}). \tag{D2}$$

Then, a unique random float is sufficient to test simultaneously all possible transitions for this pixel. For any r, a random float in the half-open interval $[0,1)$, one can find an index k such that $\eta_{k-1} \leq r < \eta_p$. The state $v = k$ in then affected to this pixel.

In plain words, we consider output states as sections of the unit interval with length equal to the various transition probabilities; the total length is then equal to one (as the transition $u \to u$ is included in the process). Intuitively, it is clear that distributing pixels at random on this unit interval will produce relative number of pixels in each possible final state proportional to the length of each interval, on average (Fig. D1), and therefore comply with $P(v|u^j,\boldsymbol{z}^j)$, on average. This property will be formally demonstrated in our paper dedicated to allocation (Mazy and Longaretti, 2022b).

This test presents several advantages: all possible final states are tested at the same time, and the order of the sections in the unit interval is irrelevant.

E. Models Parameters

In section 7, we use various LUC change models with the following parameters.

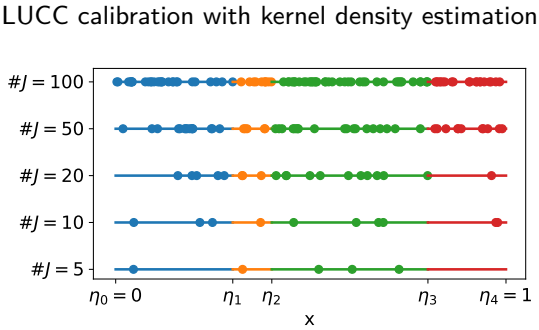


Figure D1: Outcome of the multinomial sampling test in a simple idealized setting with four possible final output states. These output states are sections of the unit interval. The length of these sections have been chosen arbitrarily. The distribution of random draws along the unit section are represented from bottom to top for an increasing number of draws (5, 10, 20, 50, 100 respectively).

parameter	value
Increment	$C_{k,k}h_{\text{Terrel}}$
Minimum Delta	50
Maximum Delta	500000
Tolerance angle (^{o})	5.0

Table E1
Dinamica EGO discretization parameters for the synthetic data of section 7.

E.1. Dinamica EGO

We used version 6.1.0 of the software. The calibration parameters of Dinamica EGO bear on the binning of the explanatory variables. For each LULC state transition and for each explanatory variable, four parameters are specified: *Increment*, *Minimum Delta*, *Maximum Delta* and *Tolerance angle*. These parameters ensure an independent and adaptive binning with a minimum number of pixels per bin and limited inter-bin statistical variations. However, Dinamica EGO does not provide any guidance on how to determine these parameters. We found a very large spread in the obtained results when these parameters are modified. The most important parameter is the one named *increment*. As a rule of thumb to choose a more relevant bin size, we have used the size of a box kernel bandwidth defined for our own eKDE algorithm (section 4.5) but applied here to the size of the bins; this size was furthermore multiplied by the variance of the marginal distribution of the considered explanatory variable. The rationale of this correction is that the eKDE bandwidth applies to normalized explanatory variables (in particular the variance has been normalized to unity) and must therefore be rescaled to apply to the original data. The other parameters are left at their default value. Table E1 summarizes our settings. Our rule of thumb provides satisfactory results in the case study of section 7.

E.2. Idrisi LCM

We have used a somewhat old Idrisi Selva license (version 17.00). Among all the calibration methods provided by LCM, we have only tested the Multilayer Perceptron (MLP) method. In section 7 and for comparison purposes, we have also used another MLP (MLP ref.) provided by the SciKitLearn Python package, with different parameters (see Tab. E2).

E.3. CLUMondo

We have used version 1.4.0 of the software. The sampling parameter is fixed to 30% of all observations. The *number of cells distance between samples* is fixed to 1 (because no spatial correlations are introduced in the case study – see section 7.1) with *no data values excluding* and *balanced sample* enabled.

E.4. CLUMPY

The eKDE parameter q (section 4.1) is fixed to 51. This is the only user-defined parameter in our procedure. This default value should be appropriate for most applications.

parameter	MLP value	MLP ref. value
Solver	SGD	Adam
Layer 1 nodes – Hidden layer sizes	$(3,)$	$(10,)$
Sample size	50%	10%
Use automatic training	yes	
Use dynamic learning rate – Learning rate strategy	dynamic – adaptative	constant
Sigmoid constant a	1.0	
Alpha		10^{-5}
Start learning rate	10^{-2}	10^{-3}
Max iterations	1000	200
Momentum factor	0.5	0.9
RMS – Tolerance	10^{-2}	10^{-4}
Accuracy Rate	100%	

Table E2
MLP parameters for the synthetic data of section 7. The terminology of Idrisi LCM MLP is noted in black and the corresponding terminology of the MLP classifier provided by SciKitLearn appears in grey. Some parameters are configurable only with one or the other environment.

Supplementary material
An accurate and powerful calibration-estimation method based on Kernel density estimation.

Pierre-Yves Longaretti, François-Rémi Mazy
STEEP – INRIA – Grenoble

September 30, 2022

Abstract

This supplementary material presents in some detail the architecture and algorithmic implementation of the fit and estimation procedure discussed in main text.

1 Kernel Density Estimation Implementation

The paper presented in a general way a KDE approximation method that starts from the idea of finely binarizing the studied space. The fit and estimation steps have been introduced and we reproduce here the algorithm outline in order to allow the reader who wishes to do so to finely understand and reproduce the method. To help understanding, we represent in figure 1 the call tree of the functions. The main key to obtaining a numerically efficient implementation lies in the way we store the sparse matrices of the observed bins during the fitting step (Alg. 1) and during the estimation step (Alg. 2). Indeed, we extend the principle of Compressed Sparse Row (also called Yale Format) to more than 2 dimensions (Alg. 5). Thus, it is then efficient to explore (Alg. 8) these sparse matrices using simple binary searches (Alg. 10). This method is implemented in an open-source python module called EKDE[1].

[1] `https://gitlab.inria.fr/fmazy/ekde`

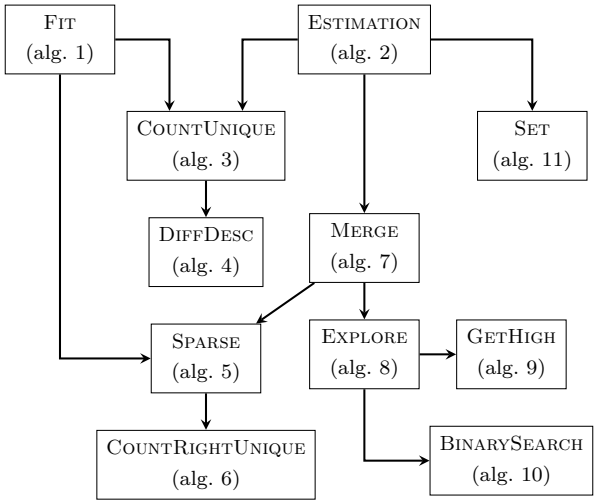


Figure 1: Call tree of all the functions involved in the implementation of the KDE method. An arrow means that the function at the origin of the arrow calls function at the arrow point.

Algorithm 1: Fit function involved in the KDE implementation. It is an algorithm outline. This is the main algorithm of the fitting step described in the main text.

Input: $\tilde{X}$, h, s, q

$\forall j < d,\ a_j = \min_{i<n}(x_{i,j})$

$\forall i, j,\ \hat{x}_{i,j} = \textsc{Floor}\left(\dfrac{\tilde{x}_{i,j} - a_i}{hs/q}\right)$

$\hat{\boldsymbol{X}} := \textsc{Sort}(\hat{\boldsymbol{X}})$

$\hat{\boldsymbol{X}}^*,\ \boldsymbol{\mu},\ \hat{\boldsymbol{X}}_{\downarrow} = \textsc{CountUnique}(\hat{\boldsymbol{X}})$

$\mathbb{S} := \textsc{Sparse}(\hat{\boldsymbol{X}}^*,\ \hat{\boldsymbol{X}}_{\downarrow})$

return $\mathbb{S}$, $\boldsymbol{\mu}$, $\boldsymbol{a}$

Algorithm 2: ESTIMATE function involved in the KDE implementation. It is an algorithm outline.

Input: $\tilde{\boldsymbol{Y}}$, $\hat{X}^*$, μ, X, h, s, q, $\boldsymbol{a}$.

1 $\forall i, j,\ \hat{y}_{i,j} = \text{FLOOR}\left(\dfrac{\tilde{y}_{i,j} - a_j}{hs/q}\right)$

2 $\hat{\boldsymbol{Y}} := \text{SORT}(\hat{\boldsymbol{Y}})$

3 $\hat{\boldsymbol{Y}}^*,\ \_,\ \hat{\boldsymbol{Y}}_{\downarrow} := \text{COUNTUNIQUE}(\hat{\boldsymbol{Y}})$ (the second return is not kept)

4 $\boldsymbol{g} := \text{MERGE}(\mathbb{S},\ \boldsymbol{\mu},\ \boldsymbol{Y},\ \hat{\boldsymbol{Y}}_{\downarrow},\ q)$

5 $\boldsymbol{f} := \text{SET}(\hat{\boldsymbol{Y}}_{\downarrow},\ \boldsymbol{g})$

6 $\boldsymbol{f} := \dfrac{1}{n(2h)^d}\boldsymbol{f}$

7 $\boldsymbol{f}$ is ordered in order to inverse the sort operation made at line 2

8 **return** f

Algorithm 3: COUNTUNIQUE function involved in the KDE implementation. It is an algorithm outline.

Input: $\boldsymbol{A}$ of size (n, d).

1 $\boldsymbol{A} = \text{SORT}(\boldsymbol{A})$

2 $\boldsymbol{B} = \text{DIFFDESC}(\boldsymbol{A})$

3 Let $\boldsymbol{A}^*$ be an empty array of d columns

4 Let $\boldsymbol{\mu}$ be an empty vector

5 $i := 0$

6 **while** $i < n$ **do**

7 append $\boldsymbol{a}_i$ to $\boldsymbol{A}^*$

8 append $b_{i,d-1}$ to $\boldsymbol{\mu}$

9 $i := i + b_{i,d-1}$

10 **return** $\boldsymbol{A}^*$, $\boldsymbol{\mu}$, $\boldsymbol{B}$

Algorithm 4: DIFFDESC function involved in the KDE implementation. It is an algorithm outline.

Input: $\boldsymbol{A}$ of size (n, d) (sorted).

1 Let $\boldsymbol{B}$ be an array of shape (n, d) full of 1

2 **for** $i \in \{n-2, n-3, \ldots, 1, 0\}$ **do**

3 **for** $j \in \text{RANGE}(d)$ **do**

4 **if** $a_{i,j} = a_{i+1,j}$ **then**

5 **if** $j = 0$ **then**

6 $b_{i,j} := b_{i+1,j} + 1$

7 **else if** $b_{i,j-1} > 1$ **then**

8 $b_{i,j} := b_{i+1,j} + 1$

9 **return** $\boldsymbol{B}$

Algorithm 5: SPARSE function involved in the KDE implementation. It is an algorithm outline.

Input:
$\boldsymbol{A}$, an array of size (n, d) (sorted)
$\boldsymbol{B}$ its DIFFDESC array.

$C :=$ COUNTRIGHTUNIQUE(B)
For all $j < d$, let $\boldsymbol{S}^j$ be an empty array of shape $(n, 2)$
$\forall i < n,\ s^{d-1}_{i,0} = a_{i,d-1}$ and $s^{d-1}_{i,1} = i$
$\forall j < d-1,\ s^{j}_{0,0} = a_{0,j}$ and $s^{j}_{0,1} = 0$
for $j \in$ *RANGE*$(d-1)$ **do**
    $i_a := 0$
    $i_s := 0$
    while $i_A < n$ **do**
        $s^j_{i_s,1} := s^j_{i_s-1,1} + c_{i_a,j}$
        $i_a := i_a + b_{i_a,j}$
        $s^j_{i_s,0} := a_{i_a,j}$
$\mathbb{S} := \{\boldsymbol{S}^0, \boldsymbol{S}^1, \ldots, \boldsymbol{S}^{d-1}\}$
return $\mathbb{S}$

Algorithm 6: COUNTRIGHTUNIQUE function involved in the KDE implementation. It is an algorithm outline.

Input: $\boldsymbol{A}$, a DIFFDESC array of size (n, d) (sorted).

Let $\boldsymbol{B}$ be an array of shape $(n, d-1)$
for $j \in$ *RANGE*$(d-1)$ **do**
    $s := 0$
    for $i \in \{n-1, n-2, \ldots, 1, 0\}$ **do**
        if $a_{i,j} = 1$ **then**
            $s := 0$
        if $a_{i,j+1} = 1$ **then**
            $s := s + 1$
        $b_{i,j} := s$
return $\boldsymbol{B}$

Algorithm 7: MERGE function involved in the KDE implementation. It is an algorithm outline.

Input:
$\mathbb{S} = \{\boldsymbol{S}^j,\ \forall\ j < d\}$, the sparse collection of unique observed elements,
μ the weights vector of size n,
$\boldsymbol{Y}$, the array to evaluate of shape (m, d) (sorted)
$\hat{\boldsymbol{Y}}_{\downarrow}$, DIFFDESC array of $\boldsymbol{Y}$,
q, the binarization parameter.

$\mathbb{T} :=$ SPARSE$(\boldsymbol{Y},\ \boldsymbol{Y}_{\downarrow})$
Let $\boldsymbol{f}$ be a vector of size n full of 0
$a_s := 0$
$b_s := \#\boldsymbol{S}^j$
for $i_t \in$ *RANGE*$(\#\boldsymbol{T}^0)$ **do**
    $\boldsymbol{f}, a_s :=$ EXPLORE $(\mathbb{S},\ \boldsymbol{\mu},\ \mathbb{T},\ \boldsymbol{f},\ j = 0,\ a_s,\ b_s,\ i_t,\ q)$
return $\boldsymbol{f}$

Algorithm 8: EXPLORE function involved in the KDE implementation (box kernel case). It is an algorithm outline.

Input:
$\mathbb{S} = \{\boldsymbol{S}^j,\ \forall\ j < d\}$, the sparse collection of unique observed elements,
$\boldsymbol{\mu}$, the weights of unique observed elements
$\mathbb{T} = \{\boldsymbol{T}^j,\ \forall\ j < d\}$, the sparse collection of unique elements to estimate,
$\boldsymbol{f}$, the density vector being completed,
j, the current column,
a_s, the low index bound for $\boldsymbol{S}^j$,
b_s, the high index bound for $\boldsymbol{S}^j$,
i_t, the current index in $\boldsymbol{T}^j$,
q, the binarization parameter.

$a_t := T^j_{i_t,1}$
$b_t :=$ GETHIGH$(\mathbb{T},\ j,\ i_t)$
$a_s :=$ BINARYSEARCH $\left(S^j,\ t^j_{i_t,0} - (q-1)/2,\ a_s,\ b_s,\ \text{"left"}\right)$
$b_s :=$ BINARYSEARCH $\left(S^j,\ t^j_{i_t,0} + (q-1)/2,\ a_s,\ b_s,\ \text{"right"}\right)$
if $j < d - 1$ **then**
    for $i_s \in$ *RANGE*(a_s, b_s) **do**
        $a'_s := s^j_{i_s,1}$
        $b'_s :=$ GETHIGH$(\mathbb{S},\ j,\ i_s)$
        for $i'_t \in$ *RANGE*$(a_t,\ b_t)$ **do**
            $\boldsymbol{f}, a'_s :=$ EXPLORE $(\mathbb{S},\ \boldsymbol{\mu},\ \mathbb{T},\ \boldsymbol{f},\ j+1,\ a'_s, b'_s, i'_t, q)$
else
    for $i_s \in$ *RANGE*(a_s, b_s) **do**
        $f_{i_t} := f_{i_t} + \mu_{i_t}$
return $\boldsymbol{f}, a_s$

Algorithm 9: GetHigh function involved in the KDE implementation.

Input:
$\mathbb{S} = \{\boldsymbol{S}^j,\ \forall\ j < d\}$, the sparse collection of unique observed elements,
j, the current column index,
i, the current row index.

if $i < \#\boldsymbol{S}^j - 1$ **then**
  return $s^j_{i+1,1}$
else
  if $j < d - 1$ **then**
    return $\#\boldsymbol{S^{j+1}}$
  else
    return $s^j_{i,1} + 1$

Algorithm 10: BinarySearch function involved in the KDE implementation. It is an algorithm outline.

Input:
$\boldsymbol{R}$, an array of shape $(n, 2)$ (sorted)
x the target,
a the lower index bound ($a \geq 0$),
b the upper index bound ($b \leq n$)
side $\in$ {"left", "right"} the binary search side.

if $a \geq b$ **then**
  if *left side* **then**
    return a
  else
    return b
if $x < r_{a,0}$ **then**
  return a
if $x > r_{b-1,0}$ **then**
  return b
while $a < b$ **do**
  $m :=$ FLOOR $\left(\dfrac{a+b}{2}\right)$
  if $r_{m,0} > x$ **then**
    $b := m$
  else
    $a := m + 1$
return m

Algorithm 11: SET function involved in the KDE implementation.

Input:
$\hat{\boldsymbol{Y}}_{\downarrow}$, the DIFFDESC of shape (n, d)
$\boldsymbol{g}$, the density estimation of unique rows.

Let $\boldsymbol{f}$ be an empty vector of size n
$i_g := 0$
while $i < n$ **do**
    for $k \in \mathit{RANGE}(y_{\downarrow,i,d-1})$ **do**
        $f_{i+k} := g_{i_g}$
    $i_g := i_g + 1$
    $i := i + y_{\downarrow,i,d-1}$