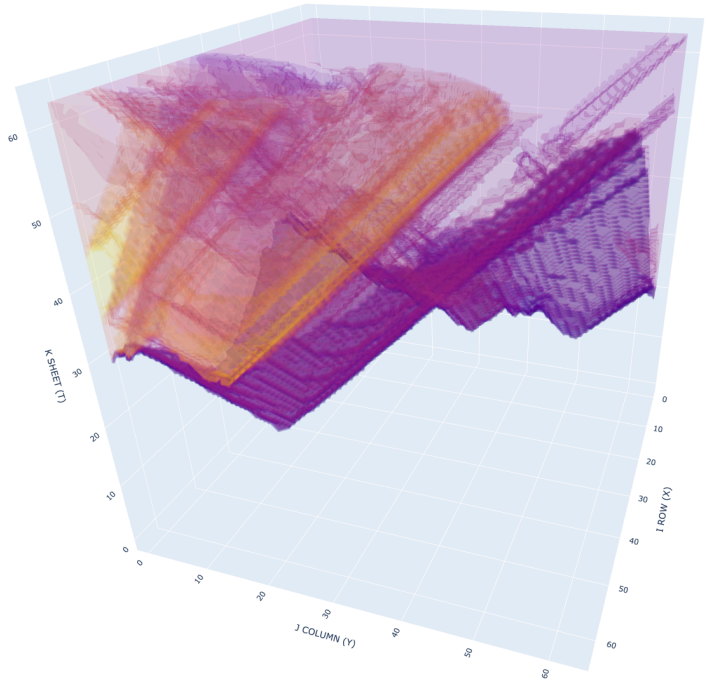
Marco Ciolfi

TIMESCAPE



Python Module - rev. 1.1 Manual

Timescape

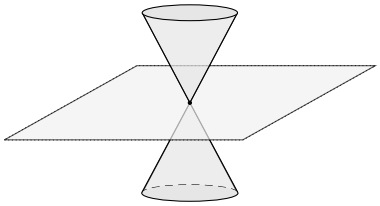
A Simple Spatiotemporal Modeling Tool

Python module - rev. 1.1

Author: Marco Ciolfi – marco.ciolfi@cnr.it
Institute of Research on Terrestrial Ecosystems
of the National Research Council, Italy

Typeset in LaTeX– September 2020

Contents



The Timescape Python module has been developed with the support of the Italian National Research Council grants STM-0051519, IBAF-003-2013-POR, IBAF-02-2017-TR, and the European Cooperation in Science and Technology grant COST STSM-TD1209-17435.

Foreword

The Timescape algorithm has been developed for the modeling of a single variable over a given spatiotemporal domain, whenever the spatial and temporal variability of the observations is equally important, and the established machinery of time series analysis and geostatistics fails to give a satisfactory result. The algorithm is based on the concept of topological filtering of the input dataset through the definition of an appropriate distance function.

To define the Timescape distance function, the time is converted into a third spatial dimension and a causal structure is introduced topologically. For each *spatiotemporal point* (or better *event*) a double cone partitions all the other events as potential causes, potential outcomes, and all the remaining causally-disconnected events. This structure, as the term event itself, is borrowed from the relativistic structure of the Minkowskian spacetime, where such a causal structure emerges naturally due to the finiteness of the speed of light. In a Timescape model, a time to space conversion factor (i.e. a velocity) has to be provided as a parameter of the algorithm.

A Timescape model –a Timescape, in short– is a regular lattice of values (the *target*), interpolated from a set of observations (the *source*) whose coordinates and time of measurement are known.

The Timescape algorithm Python module is a complete geostatistical toolbox. It has been developed following these requirements: it has to be

1. not to choosy about the distribution of the input data
2. capable to accommodate seasonal variability
3. suitable for both both projected and geographical coordinates
4. capable to run on affordable hardware in decent times

The resulting module is a collection of objects and functions, aimed at the novice/intermediate Python user. The output can be exported as a multilayered geotiff image to be included in a GIS workflow.

1 Timescape Modeling

A Timescape can be thought of as a stack of layered spatial models: a sort of landscape varying over time; a Timescape model can be seen as well as a collection of time series on a regular lattice of spatial sites, a sort of "cube" of 3D cells of values of a real random variable (a scalar field).

Let us imagine the spatiotemporal distribution of some input observations, arranged as in figure 1: the time goes upwards vertically, the space is represented horizontally. In such a representation, each point $\boldsymbol{p}$ (more properly each *event*[1] $\boldsymbol{p}$) is placed into a *spacetime* whose coordinates are time and position. Seen from any given $\boldsymbol{p}$, the spacetime is a stack of sheets, each of which is a replica of the spatial region of interest. Some sheets lie in the past and some in the future of $\boldsymbol{p}$, the contemporaneity of $\boldsymbol{p}$ is the particular sheet passing through $\boldsymbol{p}$ itself. The distribution of a given variable associated with the spatiotemporal events is the object of our algorithm: given a set of observations (the *source*, i.e. the set of values measured at given events), we produce a set of estimated values (the *target*, typically a regular lattice of events) whose values depend on the observations and on a chosen geostatistical algorithm.

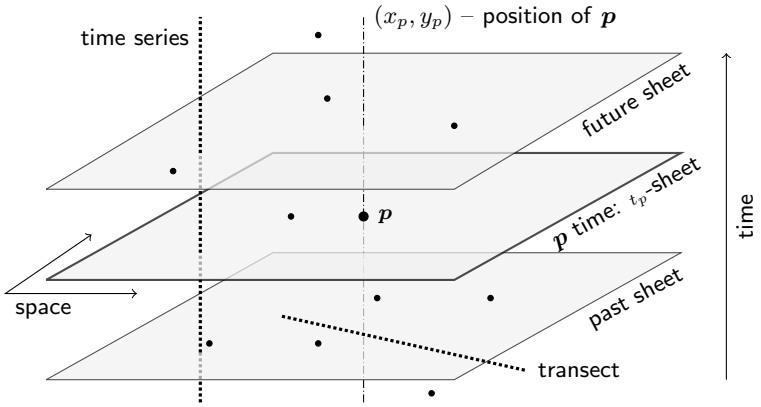


Figure 1: An event $\boldsymbol{p}$ of coordinates (t_p, x_p, y_p) surrounded by some other events (small dots).

Once evaluated the output spatiotemporal distribution, it is possible to extract a spatial distribution of values at a given time out of the target events sheets. It is also possible to extract an estimated time series at each point in space as a sort of core of the whole spacetime (figure 1).

[1]The term *event* comes from the jargon of relativistic physics, meaning an elemento of the so-called *Minkowski space* [1, 9, 24].

In a nutshell, the Timescape algorithm enriches the spacetime with a notion of causality: first of all, in order to homogenise the units, the time coordinate is multiplied by a factor c with the dimensions of a velocity (or an angular velocity, if spherical coordinates are concerned). Then given any event $\boldsymbol{p}$, only those other events that lie in its past can be considered as possible causes of $\boldsymbol{p}$, provided they are not too far from it. But how far? We introduce a sort of double cone, the *causal cone*[2] (one per each event $\boldsymbol{p}$) that sorts all the elements of our spacetime into three categories: the possible causes, the possible outcomes, and all the remaining events.

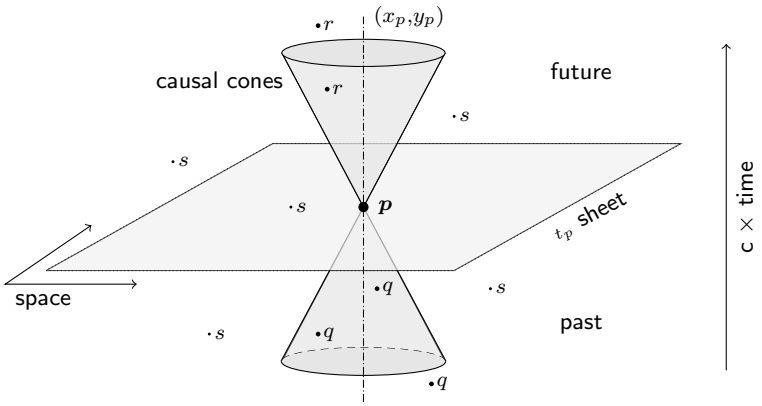


Figure 2: A visual representation of the spatiotemporal causal structure centered on $\boldsymbol{p}$.

Figure 2 shows the spacetime as seen from $\boldsymbol{p}$: the events $\boldsymbol{q}$ that lie in the past of $\boldsymbol{p}$, but not too far from it, can be considered as possible causes (i.e. sources of knowledge) for $\boldsymbol{p}$. Conversely, the possible outcomes of $\boldsymbol{p}$ are only those events $\boldsymbol{r}$ in the future of $\boldsymbol{p}$ that are not too far from $\boldsymbol{p}$ itself. The rest of the events, marked $\boldsymbol{s}$ in figure 2, have no causal connection with $\boldsymbol{p}$, no matter how spatially close to $\boldsymbol{p}$ they are. It is apparent in the figure how changing the width (i.e. the tip angle) of the cone more or less events are included in, or fall outside of the causal cone: the choice of the width of the cone is of paramount importance in the Timescape algorithm and is controlled by the cone width parameter k. Each parameters pair (c, k) generates a unique Timescape model, given the same input dataset and the same spatial interpolator.

[2]The *double cone* is the fundamental structure of causal connectedness in relativity, it emerges naturally from the role that the speed of light plays in such context as a limiting speed [1]. in our case, however, such a causal structure has to be introduced somewhat artificially, since there is nothing like a limiting speed.

The basic idea of the Timescape algorithm consists in interpolating the unknown values of the target events using only those source events (the observations) that lie inside each target event's past causal cone. This is achieved through topological filtering, i.e. a selection based on the connectedness imposed by the causal cones structure, rather than the distances: the causal structure can be encoded in a spatiotemporal distance function $D(\boldsymbol{p}, \boldsymbol{q})$ that allows the use of any geostatistical algorithm, filtering *a priori* the possible causes for any $\boldsymbol{p}$ and correcting the spatial distance with the elapsed time. This spatiotemporal distance induces a quasi-metric structure [23] on the model's space, such a structure allows the topological filtering above mentioned, as detailed in section 2.3, without interfering with the subsequent application of any conventional geostatistical algorithm.

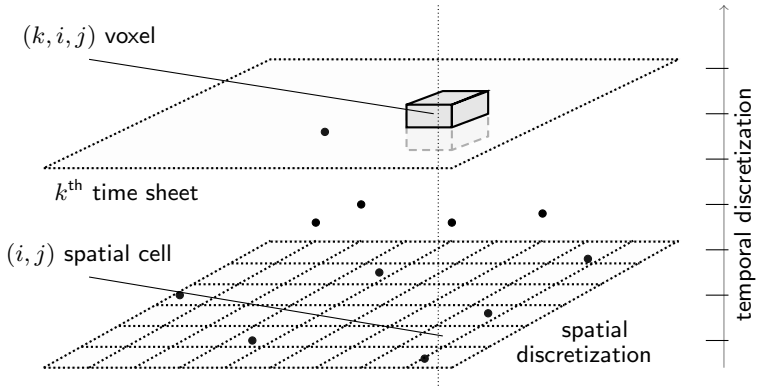


Figure 3: A Timescape model: the *target* events are a lattice of space-time voxels; the *source* events are represented as scattered dots. The voxel is identified with its center's coordinates.

A Timescape model, operatively, consists is a discrete lattice of spatiotemporal cells, or voxels (the *target*), whose values are evaluated from a set of observations at given places and times (the *source*) – see figure 3. The time span of the model typically exceeds the observations' times, thus resulting in a forecast of the modeled variable. Some dedicated functions allow the extraction from a target lattice of single-time, ordinary spatial models (horizontal layers of voxels) and time series (vertical stacks of voxels).

Model evaluation Each single voxel's value is independent of the other ones, so the model's target computation could be easily parallelized. A voxel is identified by a temporal index k and a pair of spatial indices (i, j), corresponding to a target lattice node (figure 3). The evaluation goes schematically as sketched in algorithm 1. The logic is straightforward but the com-

putational load is quite heavy, the number of distances to be evaluated, in particular, scales as the square of the number of source events.

Algorithm 1 Timescape

```
Read the source events: {q} ← input file
Define the target size and boundaries
for all (k, i, j) ∈ target do
    Prepare p_kij as an empty voxel (value = null)
    for all q ∈ source do
        Evaluate D(q → p_kij)          ▷ The distance from q to p – see Algorithm 4
    if {D(q → p_kij)} ≠ {null} then
        p_kij's value ← Estimate with any spatial interpolator          ▷ See Algorithm 3
return target ← {p_kij}
```

Since the output is a volatile python object, it is advisable to save it immediately in a more robust form; the module provides a binary file format as the standard solution, but a Timescape can also be exported as ascii plain text or as a multi-layered geotiff image [18], the latter is especially intended to fit the geographical information systems standards. The ascii text is meant for advanced statistics, e.g. to be imported into R [22].

Quality assessment Since a Timescape model is eventually evaluated through some established three-dimensional spatial statistical method – e.g. Kriging (section 3.2.2), an accuracy estimate can be obtained as the method's own accuracy. Furthermore, the module offers a residual estimation function for the tuning of the causal cone parameters (sections 2.7 and 4.2) which can be used as a quality estimate whenever the spatial interpolation method does not provide itself an accuracy estimate, as is the case with the Inverse Distance Weighting interpolators (section 3.2.1).

If the number of source events is not too scarce and the running time is acceptable, it is possible to estimate the model accuracy through jackknife resampling [6] or bootstrapping [7]. It is advisable to work on downscaled Timescape models, keeping the desired spatial and temporal boundaries, but decreasing the number of voxels; alternatively, if one is interested in a smaller area or a particular time, the model boundaries can be restricted accordingly.

The Timescape algorithm is described step-by-step in section 3, the way it is implemented in the module, while the its building blocks are dealt with in section 2 in detail. Section 4 is the actual Timescape module manual.

2 Mathematical Foundations

This rather technical section covers all the building blocks of the Timescape algorithm in detail. A short informal summary is provided in sec. 2.9. The section's main goal is the definition of the spatiotemporal distance (eq. 14) at the base of the algorithm.

2.1 Spatial Metrics

Given a set A, a *distance function* or *metric* $d : A \times A \longrightarrow [0, +\infty)$ is a function satisfying, for any[3] $\mathsf{p}, \mathsf{q} \in A$, the following conditions:

- identity: $d(\mathsf{p}, \mathsf{q}) = 0 \iff \mathsf{p} \equiv \mathsf{q}$
- non-negativity: $d(\mathsf{p}, \mathsf{q}) \geq 0$
- symmetry: $d(\mathsf{p}, \mathsf{q}) = d(\mathsf{q}, \mathsf{p})$
- subadditivity: $d(\mathsf{p}, \mathsf{q}) \leq d(\mathsf{p}, \mathsf{s}) + d(\mathsf{s}, \mathsf{q}) \quad \forall \mathsf{s} \in A$

The latter (also known as triangle inequality), in particular, gives the metric its distinctive character of measurement of nearness and it is often the key property for the convergence of the interpolation algorithms.

The spatial coordinates encountered in ecological datasets are most often projected ones (e.g. Universal Transverse Mercator or Lambert conic), so the spaces are locally isomorphic to $\mathbb{R}^2$. Sometimes one has to use the geographical coordinates, if the spatial extent requires so, in this case the Earth is usually approximated with the S^2 sphere. The Timescape Python mosule provides the user with the most widespread[4] $\mathbb{R}^2$ and S^2 metrics:

- `EUCLID` - the Euclidean metric on $\mathbb{R}^2$:

$$d_E(\mathsf{p}, \mathsf{q}) \;=\; \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2} \tag{1}$$

- `SQUARE` - the square, or Chebyshev metric on $\mathbb{R}^2$:

$$d_S(\mathsf{p}, \mathsf{q}) \;=\; \max\Big\{\, |x_p - x_q|,\, |y_p - y_q| \Big\} \tag{2}$$

[3] Upright $\mathsf{p}, \mathsf{q} \ldots$ are used for the spatial points to distinguish them from the spatiotemporal events $\boldsymbol{p}, \boldsymbol{q}, \ldots$, to be introduced later.

[4] The three $\mathbb{R}^2$ metrics are all induced by the standard l_p-norms $\|\boldsymbol{x}\|_p = \sqrt[p]{\sum_k |x_k|^p}$ with $p{=}1$ (equation 3), $p{=}2$ (equation 1) and $p{\to}\infty$ (equation 2), see figure 4.

- `DIAMOND` - the diamond, or *Manhattan metric* on $\mathbb{R}^2$:

$$d_D\big(\mathsf{p},\mathsf{q}\big) \;=\; |x_p - x_q| \;+\; |y_p - y_q| \tag{3}$$

- `SPHERE` - the geodesic distance on S^2:

$$d\big(\mathsf{p},\mathsf{q}\big) \;=\; R\,\Theta\big(\mathsf{p},\mathsf{q}\big) \qquad \text{where} \tag{4}$$

$$\Theta\big(\mathsf{p},\mathsf{q}\big) \;=\; \arccos\Big[\sin\phi_p \sin\phi_q \;+\; \cos\phi_p \cos\phi_q \cos\big(\lambda_p - \lambda_q\big)\Big] \tag{5}$$

where the points p and q are represented by projected ($\mathbb{R}^2$ embedded) coordinates $\mathsf{p} = (x_p, y_p)$, $\mathsf{q} = (x_q, y_q)$ or by spherical[5] ones in S^2: $\mathsf{p} = (\lambda_p, \phi_p)$, $\mathsf{q} = (\lambda_q, \phi_q)$. R is the radius of the sphere and Θ is the central angle spanned by the geodesic connecting p and q, a section of a great circle [2].

Equations 1, 2 and 3 define three different, but equivalent[6] neighbourhoods of circular, square and diamond shape, respectively.

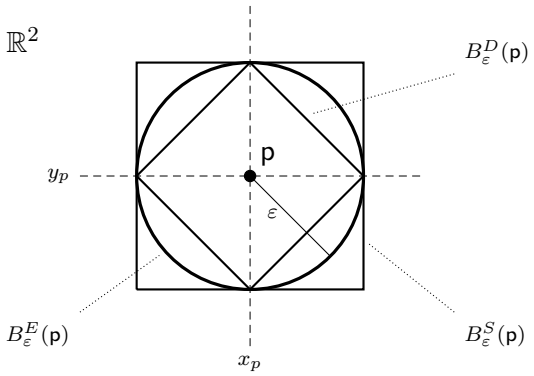


Figure 4: Balls of radius ε around p in the Euclidean, square and diamond metric: $B_\varepsilon^E(\mathsf{p}) = \{\mathsf{q} \in \mathbb{R}^2 | d_E(\mathsf{p},\mathsf{q}) < \varepsilon\}$, $B_\varepsilon^S(\mathsf{p}) = \{\mathsf{q} \in \mathbb{R}^2 | d_S(\mathsf{p},\mathsf{q}) < \varepsilon\}$, $B_\varepsilon^D(\mathsf{p}) = \{\mathsf{q} \in \mathbb{R}^2 | d_D(\mathsf{p},\mathsf{q}) < \varepsilon\}$.

Three nested balls with the same radius ε, $B_\varepsilon^D(\mathsf{p}) \subset B_\varepsilon^E(\mathsf{p}) \subset B_\varepsilon^S(\mathsf{p})$, are shown in figure 4. This nesting implies that the convergence in the diamond metric is somewhat stronger than that in the Euclidean or square

[5] $\lambda \in [0, 2\pi]$ is the longitude and $\phi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ is the latitude. S^2 is to be dealt with some care since λ is periodic because $\lambda = 0 \equiv 2\pi$ and $\big(\lambda, \phi = \pm\frac{\pi}{2}\big)$ are singular points – the poles, while $\phi = 0$ defines the equator. This definition is closer to the geographical needs than most of the geometry textbooks, where generally ϕ goes from 0 (the north pole, here $+\frac{\pi}{2}$) to π (the south pole, here $-\frac{\pi}{2}$).

[6] Loosely speaking, two metrics are said to be equivalent if their neighbourhoods are reciprocally nested, i.e. if for any ball about any point in one metric, it exists a ball in the other metric entirely contained in the first one, and vice versa.

ones, since $\mathsf{q} \in B_\varepsilon^D(\mathsf{p}) \Rightarrow \mathsf{q} \in B_\varepsilon^E(\mathsf{p}) \Rightarrow \mathsf{q} \in B_\varepsilon^S(\mathsf{p})$. In the majority of cases in ecological datasets the metric of choice is the Euclidean one, as it is the ordinary distance as measured on the map. The other metrics are provided for using other kinds of non-geographical distances.

If the scale of the problem includes the whole Earth or substantial portions of it (continental scale at least) the use of geographical coordinates is mandatory. In this case, as a matter of principle, the distance between two points is to be found by solving the geodesic equation on the manifold representing the surface of the Earth. In most cases of ecological interest, however, the Earth surface can be approximated to a sphere: equation 4 is a good estimate of the distance as the length of the section of great circle passing by its extremes. The `SPHERE` metric is provided in the Timescape software; users can use the default value[7] for the Earth radius or they can provide their custom radius.

2.2 Causality and Spatiotemporal Metrics

There are many ways to extend an ordinary spatial metric to a spatiotemporal one. The simplest one consist in transforming the temporal coordinate to a spatial one, multiplying the time by a constant c with the dimensions of a velocity: $t \to ct$. The new coordinate can be used as a spatial one $(t, x, y) \to (ct, x, y)$, naturally extending the $\mathbb{R}^2$ metric to $\mathbb{R}^3$: e.g. for the Euclidean metric $\sqrt{\Delta x^2 + \Delta y^2} \to \sqrt{c^2\Delta t^2 + \Delta x^2 + \Delta y^2}$. The use of geographical coordinates is not much different, in principle, extending the S^2 metric to $\mathrm{S}^2 \times \mathbb{R}$: $R\Theta \to \sqrt{c^2\Delta t^2 + R^2\Theta^2}$, where the angle Θ is defined in (5). This simple spatialisation of time, as described so far, takes no care of any possible causal relationship among the elements of $\mathbb{R}^3$ or $\mathrm{S}^2 \times \mathbb{R}$.

From here on, the term *point* is referred only to the spatial coordinates, e.g. $\mathsf{p} = (x_p, y_p)$ or $\mathsf{q} = (\lambda_q, \phi_q)$, while *event* is referred to the spatial coordinates along with the times[8] they are attached to: $\boldsymbol{p} = (t_p, \mathsf{p}) = (t_p, x_p, y_p)$ in $\mathbb{R}^3$ or $\boldsymbol{p} = (t_p, \mathsf{p}) = (t_p, \lambda_p, \phi_p)$ in $\mathrm{S}^2 \times \mathbb{R}$.

Said $\mathcal{E}$ the space of all the events[9], given $\boldsymbol{p}, \boldsymbol{q} \in \mathcal{E}$ their *temporal dis-*

[7] The default average radius of the Earth is set to $R_\oplus = 6378100$ m [5].

[8] An *event* is any point of the Minkowskian relativistic spacetime. Also *causal cone*, introduced later, comes from the relativistic physics jargon [1, 24].

[9] $\mathcal{E}$ can be thought of as a subset of $\mathbb{R}^3$; more generally, it is a subset of $\mathcal{M} \times \mathbb{R}$ where $\mathcal{M}$ is a suitable manifold that maps the geometry of the actual space [9, 10].

tance[10] is defined as $D_t(\boldsymbol{p},\boldsymbol{q}) = c\,|t_p - t_q|$, and their *spatial distance* $D_s(\boldsymbol{p},\boldsymbol{q}) = d(\mathsf{p},\mathsf{q})$, where $d(\mathsf{p},\mathsf{q})$ is any suitable distance, like in equations 1 to 4. The distance between any pair of events $\boldsymbol{p},\boldsymbol{q} \in \mathcal{E}$ is then defined as

$$D(\boldsymbol{p},\boldsymbol{q}) = \sqrt{D_t^2(\boldsymbol{p},\boldsymbol{q}) + D_s^2(\boldsymbol{p},\boldsymbol{q})} \tag{6}$$

this function $D : \mathcal{E} \times \mathcal{E} \longrightarrow [0,+\infty)$ inherits all the distance properties from D_t and D_s: identity, non-negativity, symmetry and subadditivity: this ensures that any spatial interpolation technique based on the notion of distance can be applied on $\mathcal{E}$ as well (figure 5).

There is a natural notion of ordering in $\mathcal{E}$: $\boldsymbol{p}$ follows $\boldsymbol{q}$ whenever $t_p \geq t_q$ and $\boldsymbol{p}$ precedes $\boldsymbol{q}$ iff $\boldsymbol{q}$ follows $\boldsymbol{p}$, i.e. $t_p \leq t_q$. For any event $\boldsymbol{p} \in \mathcal{E}$ we can define the past and the future of $\boldsymbol{p}$ as $\mathcal{P}_p = \{\boldsymbol{q} \in \mathcal{E} \mid \boldsymbol{q} \text{ precedes } \boldsymbol{p}\}$ and $\mathcal{F}_p = \{\boldsymbol{q} \in \mathcal{E} \mid \boldsymbol{q} \text{ follows } \boldsymbol{p}\}$. We have, trivially, $\mathcal{F}_p \cup \mathcal{P}_p = \mathcal{E}$ and we can define the contemporaneity of $\boldsymbol{p}$ as $\mathcal{N}_p = \mathcal{F}_p \cap \mathcal{P}_p = \{\boldsymbol{q} \in \mathcal{E} \mid t_q = t_p\}$. Figure 5 shows the partition of $\mathcal{E}$ into $\mathcal{P}_p$ and $\mathcal{F}_p$ for each given $\boldsymbol{p}$.

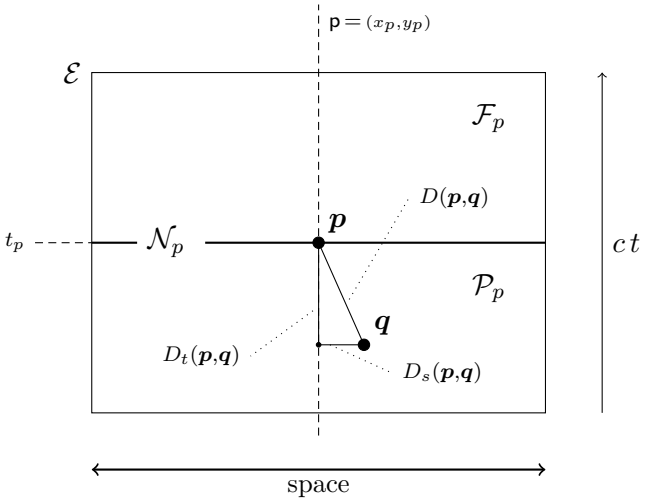


Figure 5: A partition of $\mathcal{E}$: the time goes upwards while the space ($\mathbb{R}^2$, S^2 or more generally a 2-manifold) is schematically contracted to a single horizontal line. $\boldsymbol{q}$ can be a cause of $\boldsymbol{p}$.

A naïve idea of causality suggests that something occurring at $\boldsymbol{p}$ cannot be caused by anything going to happen wherever in $\mathcal{F}_p$, but it can be caused by something happened in its past $\mathcal{P}_p$, regardless the (spatial) distance that separates $\boldsymbol{p}$ from a possible cause. This is a bit loose since it allows the instantaneous propagation of the cause-effect chain.

[10] D_t is just like (2) and (3) in one dimension.

In order to incorporate the notion of causal finiteness (i.e. it takes time for a cause to produce an effect), we must impose a constraint which translates *the later, the farther* intuitive idea of propagation into a geometrical feature. To do so, we introduce a *causal cone* structure over $\mathcal{E}$ that reduces the possible causes of $\boldsymbol{p}$ to a suitable subset of $\mathcal{P}_p$. We seek two subsets of $\mathcal{F}_p$ and $\mathcal{P}_p$ behaving as follows: an event $\boldsymbol{q}$ can be a cause or an outcome of $\boldsymbol{p}$ if $D_s(\boldsymbol{p},\boldsymbol{q}) \leq k\, D_t(\boldsymbol{p},\boldsymbol{q})$, for a given adimensional $k \geq 0$. We define the *future causal cone*[11] of $\boldsymbol{p}$ as

$$\mathcal{C}_p^{k+} = \left\{ \boldsymbol{q} \in \mathcal{E} \,\middle|\, D_s(\boldsymbol{p},\boldsymbol{q}) \leq k\, D_t(\boldsymbol{p},\boldsymbol{q}) \;\wedge\; t_q \geq t_p \right\} \tag{7}$$

and the *past causal cone* of $\boldsymbol{p}$ as

$$\mathcal{C}_p^{k-} = \left\{ \boldsymbol{q} \in \mathcal{E} \,\middle|\, D_s(\boldsymbol{p},\boldsymbol{q}) \leq k\, D_t(\boldsymbol{p},\boldsymbol{q}) \;\wedge\; t_q \leq t_p \right\} \tag{8}$$

Trivially, $\boldsymbol{p} \in \mathcal{C}_q^+$ iff $\boldsymbol{q} \in \mathcal{C}_p^-$ and $\mathcal{C}_p^+ \cap \mathcal{C}_p^- = \{\boldsymbol{p}\}$ but $\mathcal{C}_p^+ \cup \mathcal{C}_p^- \subset \mathcal{E}$ strictly, unless $k \to \infty$, see figure 6.

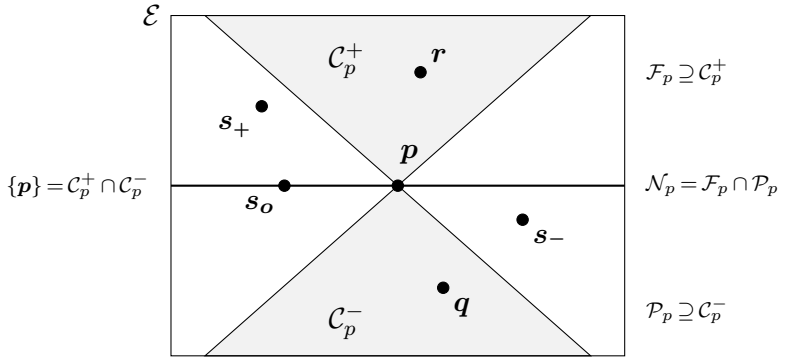


Figure 6: Causal cones: each $\boldsymbol{p}$ partitions $\mathcal{E}$ in three subsets. The event $\boldsymbol{q} \in \mathcal{C}_p^{k-}$ can be a cause of $\boldsymbol{p}$, $\boldsymbol{r} \in \mathcal{C}_p^{k+}$ can be an outcome of $\boldsymbol{p}$, while $\boldsymbol{s}_+ \in \mathcal{F}_p$, $\boldsymbol{s_o} \in \mathcal{N}_p$ and $\boldsymbol{s}_- \in \mathcal{P}_p$ have no causal connection with $\boldsymbol{p}$.

The *causal parameter* k above defined tunes the strictness of causality (figure 7): the bigger k, the looser the constraint. We have, on the loose side:

$$\lim_{k\to\infty} \mathcal{C}_p^{k+} \;=\; \mathcal{F}_p \qquad \text{and} \qquad \lim_{k\to\infty} \mathcal{C}_p^{k-} \;=\; \mathcal{P}_p \tag{9}$$

[11] $\mathcal{C}_p^{k\pm}$ will be shortened to $\mathcal{C}_p^{\pm}$ whenever no confusion arises.

and on the strict side:

$$\begin{aligned}\lim_{k\to 0^+} \mathcal{C}_p^{k+} &= \left\{\boldsymbol{q} \in \mathcal{E} \,\middle|\, t_q \geq t_p,\, x_q = x_p,\, y_q = y_p\right\} \\ \lim_{k\to 0^+} \mathcal{C}_p^{k-} &= \left\{\boldsymbol{q} \in \mathcal{E} \,\middle|\, t_q \leq t_p,\, x_q = x_p,\, y_q = y_p\right\}\end{aligned} \tag{10}$$

so $\boldsymbol{p}$ and k define a sheaf of double cones joined at their common tip $\boldsymbol{p}$, covering $\mathcal{E}$ for each $\boldsymbol{p}$ since $\mathcal{C}_p^{k\pm} \supseteq \mathcal{C}_p^{k_o\pm}$ for $k \geq k_o$, so $\bigcup_k \mathcal{C}_p^{k-} = \lim_{k\to\infty} \mathcal{C}_p^{k-} = \mathcal{P}_p$ and $\bigcup_k \mathcal{C}_p^{k+} = \lim_{k\to\infty} \mathcal{C}_p^{k+} = \mathcal{F}_p$.

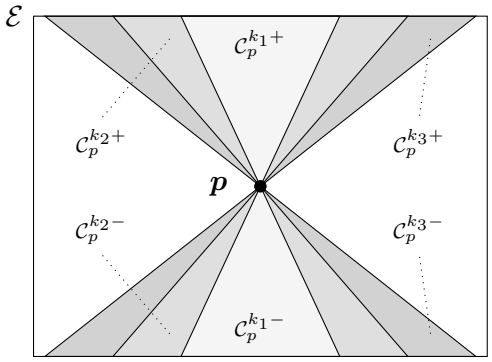


Figure 7: Causality strictness, here $k_1 < k_2 < k_3$.

We can define a topology $\mathcal{T}_\mathcal{E}$ on $\mathcal{E}$ taking the $\mathcal{C}_p^{k\pm}$ as closed sets: $\mathcal{T}_\mathcal{E} = \left\{\mathcal{C}_p^{k\pm}, \boldsymbol{p} \in \mathcal{E}\right\}$ since any finite union and any intersection of $\mathcal{C}_p^{k\pm}$ is in $\mathcal{T}_\mathcal{E}$ since

$$\bigcup_{k\in K\text{finite}} \mathcal{C}_p^{k\pm} = \mathcal{C}_p^{\max\{K\}\pm} \qquad \text{and} \qquad \bigcap_{k\in K} \mathcal{C}_p^{k\pm} = \mathcal{C}_p^{\inf\{K\}\pm} \tag{11}$$

Thus the topology $\mathcal{T}_\mathcal{E}$ is partially ordered by the inclusion relation. in the following we exploit this ordering to filter, for each $\boldsymbol{p}$, its possible causes, picking from all the source events those lying in some $\mathcal{C}_p^-$.

The analytical expression for the distance from event $\boldsymbol{q}$ to $\boldsymbol{p}$ is thus

$$D(\boldsymbol{q} \to \boldsymbol{p}) = \frac{\sqrt{D_t^2(\boldsymbol{p},\boldsymbol{q}) \;+\; D_s^2(\boldsymbol{p},\boldsymbol{q})}}{\theta\Big[k\,\big|D_t(\boldsymbol{p},\boldsymbol{q})\big| - D_s(\boldsymbol{p},\boldsymbol{q})\Big]} \tag{12}$$

where $\theta(\cdot)$ is the Heaviside step function[12] and $\frac{\text{finite}}{0} = \infty$ is understood. In

[12]The Heaviside function is defined as $\theta(x) = \begin{cases} 1 \text{ if } x \geq 0 \\ 0 \text{ otherwise} \end{cases}$.

the most common case of Euclidean distance, (12) reduces to

$$D(\boldsymbol{q} \to \boldsymbol{p}) = \frac{\sqrt{c^2(t_p - t_q)^2 + (x_p - x_q)^2 + (y_p - y_q)^2}}{\theta\Big[k\,c(t_p - t_q) - \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}\Big]} \tag{13}$$

Notice that, although D_s and D_t are symmetrical, $D(\boldsymbol{q} \to \boldsymbol{p})$ is not[13]

$$D(\boldsymbol{q} \to \boldsymbol{p}) \text{ finite } \implies D(\boldsymbol{p} \to \boldsymbol{q}) = \infty$$

since $t_p - t_q \geq 0$ (i.e. $\boldsymbol{q} \in \mathcal{P}_p$) obviously implies $t_q - t_p < 0$ (i.e. $\boldsymbol{p} \in \mathcal{F}_q$), so that the denominator vanishes. On the other hand, the opposite

$$D(\boldsymbol{q} \to \boldsymbol{p}) = \infty \implies D(\boldsymbol{p} \to \boldsymbol{q}) \text{ finite}$$

only holds when $\boldsymbol{q} \in \mathcal{P}_p$ or $k \to \infty$ (i.e. $\mathcal{C}_p^- \equiv \mathcal{P}_p$ and $\mathcal{C}_p^+ \equiv \mathcal{F}_p$). If $\boldsymbol{p}$ and $\boldsymbol{q}$ lie outside each other's cones[14], both $D(\boldsymbol{q} \to \boldsymbol{p})$ and $D(\boldsymbol{p} \to \boldsymbol{q}) = \infty$.

According to the particular programming language's taste, (12) can be implemented as-is, or inverted; the inversion is particularly effective if the interpolation is carried by an inverse distance weighting algorithm:

$$D^{-1}(\boldsymbol{q} \to \boldsymbol{p}) = \frac{\theta\Big[k\,\big|D_t(\boldsymbol{p}, \boldsymbol{q})\big| - D_s(\boldsymbol{p}, \boldsymbol{q})\Big]}{\sqrt{D_t^2(\boldsymbol{p}, \boldsymbol{q}) + D_s^2(\boldsymbol{p}, \boldsymbol{q})}}, \quad \text{positing } \frac{0}{0} = 0 \tag{14}$$

Another possible adjustment to (12) consists in defining

$$D(\boldsymbol{q} \to \boldsymbol{p}) = \begin{cases} \sqrt{D_t^2(\boldsymbol{p}, \boldsymbol{q}) + D_s^2(\boldsymbol{p}, \boldsymbol{q})} & \text{if } D_s(\boldsymbol{p}, \boldsymbol{q}) \leq k D_t(\boldsymbol{p}, \boldsymbol{q}) \\ \mathsf{null} & \text{otherwise} \end{cases} \tag{15}$$

exploiting the null type offered by most languages. Equation (15) has been chosen for the Python implementation – see (23). Formally, (12) and (15) are restrictions of (6) to $\mathcal{C}_p^-$: $D(\boldsymbol{q} \to \boldsymbol{p}) = D\restriction_{\mathcal{C}_p^-}(\boldsymbol{p}, \boldsymbol{q})$. The distance (12) is naturally ordered, while is (15) is ordered positing $\forall x : \mathsf{null} \succ x$, so that $\mathcal{E} \setminus \{\boldsymbol{p}\}$ is a poset for each $\boldsymbol{p} \in \mathcal{E}$. With the above introduced distance, $\mathcal{E}$

[13] This is sometimes called a *quasi-metric*: a metric missing the symmetry property [23].

[14] In this case $\boldsymbol{p}$ and $\boldsymbol{q}$ are said to be separated by a *spacelike interval* in relativity, as opposed to a *timelike interval* whenever $\boldsymbol{p} \in \mathcal{C}_q^{\pm}$ [1].

is given a rich topological structure, which is the supporting space for the filtering that lies at the base of the Timescape algorithm.

2.3 Topological Filtering

In order to exploit all the machinery of spatial statistics, it is advisable not to modify the metric properties of the base space involved [17]. The Timescape algorithm, in a nutshell, consists in using only the events in $\mathcal{C}_p^-$ to compute any geostatistical algorithm out of the users' toolbox: events are considered causally connected or not according to the user-tunable parameter[15] k. This *topological filtering* operates on the connectedness of the elements of $\mathcal{E}$, without modifying their respective distances. Pictorially, it is a bit like introducing cracks, wrinkles and tunnels in an otherwise smooth space.

There is an important issue which is often encountered in ecological studies: the time dependence of a variable shows a seasonal (i.e. periodic) character: the causal cone needs to accommodate the events selectively, according tho their "seasonal nearness". A topological solution consists in shrinking and expanding two new cone-like surfaces, $\widetilde{\mathcal{C}}_p^{\pm}$, inside their respective $\mathcal{C}_p^{\pm}$ (figure 8).

These two new surfaces $\widetilde{\mathcal{C}}_p^{k\pm}$ depend parametrically on the period T other than k: $\widetilde{\mathcal{C}}_p^{k,T\pm}$, their convex hull is given by $\widetilde{\mathcal{C}}_p^{k\pm}$, the relative maxima of $\widetilde{\mathcal{C}}_p^{k,T\pm}$ are tangent to $\widetilde{\mathcal{C}}_p^{k\pm}$, they grow as k grows, but they are periodically shrunk to the (x_p, y_p) axis of the cones. This behaviour is obtained through an adjustment of D_t:

$$D_t(\boldsymbol{p},\boldsymbol{q}) \;\longrightarrow\; \widetilde{D}_t(\boldsymbol{p},\boldsymbol{q}) \;=\; c\,|t_p - t_q|\,\cos^2\left(\pi\,\frac{t_p - t_q}{T}\right) \qquad (16)$$

where T is the period of the seasonal variation to be modeled. The shape resulting from equation 16 is shown in figure 8. The number of events in $\widetilde{\mathcal{C}}_p^{k,T\pm}$ is apparently less than that in $\mathcal{C}_p^{k\pm}$ for any given k.

Should this correction be too selective, its effect could be tempered bal-

[15] This is borrowed from the topological structure of the Minkowskian spacetime [1], but there is a crucial difference: In relativistic physics the speed of light c plays both the roles of conversion factor and causal constraint, like fixing our $k = 1$. Moreover, the Minkowskian relativistic metric $D^2(\boldsymbol{p},\boldsymbol{q}) = D_t^2(\boldsymbol{p},\boldsymbol{q}) - D_s^2(\boldsymbol{p},\boldsymbol{q})$ is not positive-definite so, strictly speaking, it is not even a distance; in a broad sense, the role of c in relativistic physics is shared by Timescape's c and k.

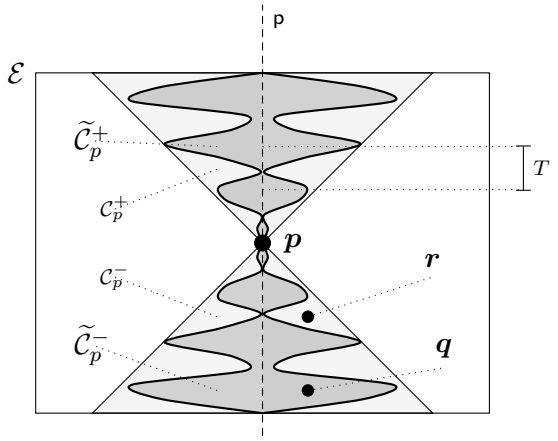


Figure 8: Periodic shrinking of $\widetilde{\mathcal{C}}_p^{\pm}$. Notice $\boldsymbol{q}, \boldsymbol{r} \in \mathcal{C}_p^-$ but $\boldsymbol{r} \notin \widetilde{\mathcal{C}}_p^-$, because $\boldsymbol{r}$ is "off season" with respect to $\boldsymbol{p}$ so it is not considered as a cause for $\boldsymbol{p}$ despite $D(\boldsymbol{p}, \boldsymbol{r}) < D(\boldsymbol{p}, \boldsymbol{q})$.

ancing D_t and $\widetilde{D}_t$ by a convex combination parametrized by $\alpha \in [0, 1]$:

$$\widetilde{D}_t^{\alpha}(\boldsymbol{p}, \boldsymbol{q}) \;=\; (1 - \alpha)\, D_t(\boldsymbol{p}, \boldsymbol{q}) \;+\; \alpha\, \widetilde{D}_t(\boldsymbol{p}, \boldsymbol{q}) \tag{17}$$

so that from $\widetilde{D}_t^0(\boldsymbol{p}, \boldsymbol{q}) = D_t(\boldsymbol{p}, \boldsymbol{q})$ it goes smoothly to $\widetilde{D}_t^1(\boldsymbol{p}, \boldsymbol{q}) = \widetilde{D}_t(\boldsymbol{p}, \boldsymbol{q})$. Practically, as α goes from 0 to 1, the causal cone is shrunk more and more deeply towards its axis. Generally it is possible to express a rotationally symmetric conical shape as the product of $D_t(\boldsymbol{p}, \boldsymbol{q})$ times a form factor, a function $\Psi(k, \Delta t)$ whose image is $\mathbb{R}^+$, that does not depend on the spatial coordinates, thus extending $D_t(\boldsymbol{p}, \boldsymbol{q})$ to

$$D_t(\boldsymbol{p}, \boldsymbol{q}) \;\;\longrightarrow\;\; \widetilde{D}_t^{\Psi}(\boldsymbol{p}, \boldsymbol{q}) \;=\; c\, \Psi(k, \Delta t)\, \left|\Delta t\right| \tag{18}$$

the effect of Ψ in (18) is an expansion for $\Psi > 1$ and a contraction, as in (17), for $\Psi < 1$. If Ψ depends explicitly on t_p and t_q rather than Δt, it means that the causality is time-dependent. A further extension that includes also the spatial coordinates as arguments of Ψ would not preserve the rotational symmetry of the $\widetilde{\mathcal{C}}_p^{\pm}$. The Python implementation of the Timescape algorithm gives the choice between two form factors: a straight causal cone ($\Psi \equiv 1$) and a periodically shrunk one, like (16), with a given period T.

2.4 Scalar Fields

A scalar field associates a scalar value representing a physical quantity (e.g. an ecological variable) to each point of the space(time) of interest. Formally, a scalar field on $\mathcal{E}$ is a real-valued measurable function whose domain is $\mathcal{E}$ and whose image coincides with the range of possible values for the variable [4]. In the language of probability, a scalar field is a real-valued random variable $V : \mathcal{E} \longrightarrow \mathbb{R}$ assigning a value v_p to each event, $V : \boldsymbol{p} \longmapsto v_p$. Since in our case $\mathcal{E}$ is (a subset of) $\mathbb{R}^3$ or $S^2 \times \mathbb{R}$, we will take it for granted that all the requests for a properly defined probability space are satisfied [8, 27].

Attaching to each event in $\mathcal{E}$ a value v of the scalar field (measured or to be modelled) we can define the set $\boldsymbol{E} = \big\{(t, x, y; v),\, (t, x, y) \in \mathcal{E},\, v \in \mathbb{R}\big\}$. We will also call *event* an event-field value pair $(\boldsymbol{p}\,; v_p)$, *dressing* the event with its associated value, so that there is a one-one correspondence $(\boldsymbol{p}\,; v_p) \leftrightarrow \boldsymbol{p}$. In order to accomodate the concept of no-knowledge of the value of the field at a given event $\boldsymbol{p}$, we allow v to get a null value, so we can have $(t_p, x_p, y_p; v_p)$ but also $(t_q, x_q, y_q; \mathsf{null})$; formally, this is a bit loose and we should restrict the domain of the random variable X to the subset of $\mathcal{E}$ of properly-valued events, but the choice of allowing null values is grounded on practical computational needs: a null value marks an event whose value has not been measured and cannot be calculated, without pulling such event out of $\boldsymbol{E}$.

The Timescape algorithm reconstructs the value of a scalar field on a target set $\boldsymbol{T}$, given its knowledge on a source set $\boldsymbol{S}$, both $\boldsymbol{S}, \boldsymbol{T} \subseteq \boldsymbol{E}$. Practically, $\boldsymbol{S}$ is a finite set (the observations) and $\boldsymbol{T}$ is some kind of lattice of events whose values are to be interpolated; it is the spatiotemporal equivalent of building a terrain model from a set of elevation points.

Figure 9 shows an example of source and target events. Two source events of $\boldsymbol{S}$ lie in the past causal cone $\mathcal{C}_p^-$ of $\boldsymbol{p} \in \boldsymbol{T}$: v_p can be interpolated from the known values of the events in $\mathcal{C}_p^- \cap \boldsymbol{S}$. The figure also shows two events marked as $\times$, these are not causally connected to any of the events in $\boldsymbol{S}$, so their field value cannot be evaluated: $\mathcal{C}_r^- \cap \boldsymbol{S} = \varnothing \implies v_r = \mathsf{null}$. Figure 9 also shows a typical events' configuration in ecological datasets, where one has many repeated measurements at the same place s.

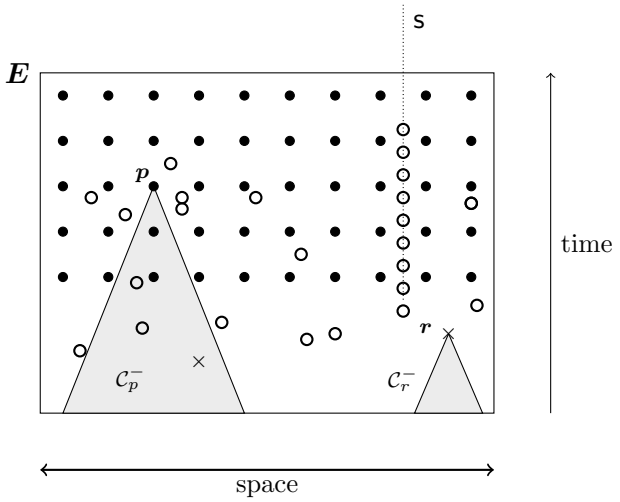


Figure 9: Source and target events: $\boldsymbol{S} = \{\circ\}$, $\boldsymbol{T} = \{\bullet\}$. The $\times$ events have null value.

2.5 Lattice Models

A spatial (or spatiotemporal) interpolation consists typically in a finite arrangement of values on a regularly spaced lattice. The three-dimensional cells are called voxels (from volume elements), we will call a spatiotemporal cell a voxel as well. In a Timescape output a voxel is a region of space of $\delta x \times \delta y$ width with a time span of δt, so its total volume[16] is $c\,\delta t\,\delta x\,\delta y$. Each voxel is identified with its centroid (t_k, x_i, y_j) where the indices triple (k, i, j) labels the voxel, the actual coordinates of a $\boldsymbol{p} \in \boldsymbol{T}$ can be easily calculated:

$$\begin{aligned} t_k &= t_{\mathsf{min}} + \Big(k - \frac{1}{2}\Big)\delta t \quad \text{where} \quad \delta t = \frac{t_{\mathsf{max}} - t_{\mathsf{min}}}{K} \\ x_i &= x_{\mathsf{min}} + \Big(i - \frac{1}{2}\Big)\delta x \quad \text{where} \quad \delta x = \frac{x_{\mathsf{max}} - x_{\mathsf{min}}}{I} \\ y_j &= y_{\mathsf{min}} + \Big(j - \frac{1}{2}\Big)\delta y \quad \text{where} \quad \delta y = \frac{y_{\mathsf{max}} - y_{\mathsf{min}}}{J} \end{aligned} \tag{19}$$

i.e. the centre of the (k, i, j)-th voxel. We will use indifferently (t_k, x_i, y_j) and (k, i, j) to mean the target event's coordinates. The whole lattice is bond spatially by $[x_{\mathsf{min}}, x_{\mathsf{max}}] \times [y_{\mathsf{min}}, y_{\mathsf{max}}]$ and temporally by $[t_{\mathsf{min}}, t_{\mathsf{max}}]$. Its volume is $c\,(t_{\mathsf{max}} - t_{\mathsf{min}})(x_{\mathsf{max}} - x_{\mathsf{min}})(y_{\mathsf{max}} - y_{\mathsf{min}})$ in length^3 units.

If the lattice consists in K sheets of I rows and J columns each, the total

[16] This is correct only if $(x, y) \in \mathbb{R}^2$, in the case of S^2 we can approximate the volume with $c\,R^2\,\delta t\,\delta\lambda\,\delta\phi$ but it is just a rough estimate for small $\delta\lambda$ and $\delta\phi$, expressed in radians. The volume is given in length^3 units.

number of voxels is KIJ; the number of voxels grows very rapidly as any of the K, I or J is increased.

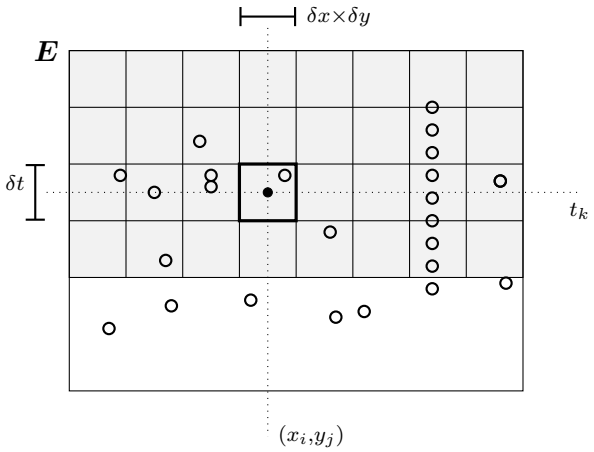


Figure 10: $\boldsymbol{T}$ as a voxels lattice. Each element is labelled with the indices triple (k, i, j) that identifies the element's centre coordinates (t_k, x_i, y_j).

The actual interpolation of the value of each element of $\boldsymbol{T}$ is carried on by any spatial interpolation algorithm. Basically, we have deterministic and statistical models [17], each algorithm has its own admirers and detractors so our python Timescape software comes with pre-packaged inverse distance weighting (IDW) and Kriging interpolators; generally, the statistical interpolators can estimate the "goodness" of the guess in terms of variance, while the deterministic methods cannot do so. Users can extend this toolbox with their favourite interpolators – see section 4.6.

In order to accommodate the estimated variance the target events are further equipped with the accuracy of the interpolation (as $\sqrt{\text{variance}}$): $\boldsymbol{p}_{kij} = (k, i, j; v, s)$, the deterministic interpolators will leave $s = \mathsf{null}$. For statistical analysis purposes the target events are also furnished with the number n of elements within their $\mathcal{C}_p^-$ (see figure 10):

$$\boldsymbol{T} \;=\; \left\{ \boldsymbol{p}_{kij} = \big(k,\, i,\, j;\, v,\, s,\, n\big),\; k = 1 \ldots K,\; i = 1 \ldots I,\; j = 1 \ldots J \right\} \tag{20}$$

where v is $\boldsymbol{p}_{kij}$'s value or null, if $n = 0$ or if the interpolation fails to estimate a value (e.g. Kriging with too few elements or a bad geometry); s is v's accuracy, estimated during v's evaluation or *a posteriori* by jackknifing the source; n is $\#\,\mathcal{C}_p^-$.

2.6 Target Events Evaluation

Each target event $\boldsymbol{p}$'s value is interpolated considering only the source events in its $\mathcal{C}_p^-$. The procedure goes as follows:

1. for each $\boldsymbol{q}_a \in \boldsymbol{S} \cap \mathcal{C}_p^-$, consider the pair $(\boldsymbol{q}_a, d_a)$ where $d_a = D(\boldsymbol{q}_a \to \boldsymbol{p})$,
2. the set of causally $\boldsymbol{p}$-connected events, defined as

$$S_p = \left\{ (\boldsymbol{q}_a, d_a),\ \boldsymbol{q}_a \in \boldsymbol{S} \cap \mathcal{C}_p^- \right\} \tag{21}$$

   is ordered by d_a (increasing distance from $\boldsymbol{p}$), so that infinite/null distances (equation 12 and 15) are sent to the tail of S_p; optionally, we can impose a cutoff on the number of elements in S_p, retaining only the first n near primes (*pruning*): $S_p \to S_p^{(n)}$, notice that this cutoff is imposed on the number of events and not on their distance from $\boldsymbol{p}$,
3. we apply any spatial interpolator, as if $\boldsymbol{S}$ were just S_p or $S_p^{(n)}$, thus assigning the value v to $\boldsymbol{p}$. The value is computed as a weighted average of the values of the events in S_p:

$$v \ = \ \sum_{\boldsymbol{q} \in S_p} w(\boldsymbol{p}, \boldsymbol{q})\ v_q \tag{22}$$

   where $w(\boldsymbol{p}, \boldsymbol{q})$ are appropriate weight functions[17]. If the interpolator allows so, it is also assigned an accuracy estimate s to $\boldsymbol{p}$.

With a slight abuse of symbology – and lack of rigour – we will allow also the source events out of $\mathcal{C}_p^-$ to be considered in (21), labelling them with a null distance, thus implementing (15):

$$\begin{aligned} S_p \ &= \ \left\{ (\boldsymbol{q}_a, d_a),\ \boldsymbol{q}_a \in \boldsymbol{S},\ d_a \in \mathbb{R}^+ \cup \{\mathsf{null}\} \right\} \quad \text{where} \\ d_a \ &= \ \begin{cases} \sqrt{D_t^2(\boldsymbol{p}, \boldsymbol{q}_a) + D_s^2(\boldsymbol{p}, \boldsymbol{q}_a)} & \text{if } \boldsymbol{q}_a \in \mathcal{C}_p^- \\ \mathsf{null} & \text{otherwise} \end{cases} \end{aligned} \tag{23}$$

and modifying the summation condition of (22) to

$$v \ = \ \sum_{d_a \neq \mathsf{null}} w(\boldsymbol{p}, \boldsymbol{q}_a)\ v_a \tag{24}$$

[17] As mentioned before, we have basically two classes of interpolators: deterministic and statistical ones. The differences lie in the weight evaluation, see section 3.2.

where v_a is the value of $\boldsymbol{q}_a$. The advantage of (23) and (24) over (21) and (22) is just a computational one; all the S_p are in fact replicas of $\boldsymbol{S}$ with the appropriate distances to $\boldsymbol{p}$. Notice that, although $D(\boldsymbol{p}, \boldsymbol{q}_a) = D(\boldsymbol{q}_a, \boldsymbol{p})$ as the symmetry property demands, we seemingly broke the symmetry forcing d_a to be null whenever $\boldsymbol{q}_a \notin \mathcal{C}_p^-$: this trick is just the topological filtering in action, the way it is implemented in the Timescape software (sec. 2.3).

2.7 Choosing c and k

The distinctive character of a Timescape model is given by the c and k parameters, which control the distance and the causal connections among the events: tuning these values is of paramount importance for a successful interpolation. A trial-and-error guess is not recommended, though. Ideally, one should try a set of parameters' values and pick the best performing ones – the question is shifted to the meaning of *best performing* in this context [3] – The Timescape Python module includes a procedure of residual calculation, which is also a rough estimate of the model's overall accuracy.

A dedicated function shows the behavior of the interpolation on a suitable statistical ensemble indexed by a set of c and k values, taking as a measure of goodness the residuals of the Timescape algorithm applied on the source events themselves. The procedure goes as follows:

1. for each event $\mathbf{q} \in \mathbf{S}$ of known value v_q define its Timescape estimate $\widehat{v}_q(c,k)$ based on the reduced source events set $\mathbf{S} \setminus \{\mathbf{q}\}$,

2. define the overall squared residual as

$$R^2(c,k) \;=\; \sum_{\mathbf{q}\in\mathbf{S}} \big[v_q - \widehat{v}_q(c,k)\big]^2 \tag{25}$$

   and the number of non-evaluable events as[18]

$$N(c,k) \;=\; \#\big\{\mathbf{q} \in \mathbf{S} \;\big|\; \widehat{v}_q(c,k) = \mathsf{null}\big\} \tag{26}$$

3. the main measure of how close this sort of auto-interpolation is to the actual values is defined as the average residual per not-null event

[18] $\#\{\cdots\}$ denotes the number of elements of the set $\{\cdots\}$.

$r(c,k)$:

$$r(c,k) \;=\; \sqrt{\frac{R^2(c,k)}{\#\mathbf{S} - N(c,k)}} \tag{27}$$

another useful measure is the number of non-evaluable events with respect to the total n(c,k):

$$n(c,k) \;=\; \frac{N(c,k)}{\#\mathbf{S}} \tag{28}$$

4. the best performing parameters pair $(\tilde{c}, \tilde{k})$ is the one to be used for the interpolation, symbolically:

$$(\tilde{c}, \tilde{k}) \;= (c,k) \;\Big|\; r(c,k) \underset{\text{ensemble}}{\longrightarrow} \min \;\wedge\; n(c,k) \underset{\text{ensemble}}{\longrightarrow} \min$$

Any optimization strategy is allowed on the ensemble of the interpolations. It is worth trying an ensemble given by a regular lattice of (c_i, k_j) values, starting with a small number of pairs with a broad range of values, then refining the minimization with a finer lattice of suitable values and so on, recursively. A dedicated function (see section 4.2) helps finding the optimal $(\tilde{c}, \tilde{k})$ pair visually, plotting $r(c,k)$ and $n(c,k)$ as surfaces for a set of regularly spaced (c_i, k_j), $i = 1 \ldots M, j = 1 \ldots N$.

As of the very first estimate of c and k, it is advisable to include $k = 1$ in the ensemble, while c can sometimes be related to some kind of transport or propagation, in this case one can start with the typical velocity related to such a phenomenon. In lack of any physical clue, consider a starting c_o such that c_o times the temporal span of $\mathbf{S}$ roughly equals its spatial extent[19]:

$$c_o = \frac{\max\left\{\, \max(x_q) - \min(x_q),\;\; \max(y_q) - \min(y_q) \right\}}{\max(t_q) - \min(t_q)} \tag{29}$$

where $\mathbf{q} \in \mathbf{S}$. In this case, keeping $k \simeq 1$ ensures that the causal cone inner volume is comparable with the whole $\mathbf{S}$ volume.

2.8 Accuracy

The accuracy of a Timescape model can be estimated by the spatial interpolator itself, if it is capable to do so, like Kriging (section 3.2.2). If

[19] Remember that, while k is dimensionless, c has the dimensions of a velocity or an angular velocity, in the case of the `SPHERE` metric.

the interpolator cannot give a standard error figure, it can be estimated by jackknifing the source [6], this is a time-consuming operation that cannot be done, practically, with a one-by-one sample removal. Following Knight [11] and McIntosh [16] it is advisable to remove d samples per run, with $\sqrt{n} < d < n-1$, where $n = \#\mathbf{S}$, the number of source events. A consistent standard error estimator (i.e. the accuracy a_{kij} associated to the single voxel's value $\hat{v}_{kij}$) is thus

$$a_{kij} \;=\; \sqrt{\frac{n-d}{\binom{n}{d}\,d}\sum_{\mathbf{S}'}\left(\hat{v}'_{kij}-\hat{v}_{kij}\right)^2} \tag{30}$$

where $\hat{v}'_{kij}$ is the voxel's value estimate with d events removed from $\mathbf{S}$. Notice that this accuracy estimate can only be done *a posteriori*, upon the completion of the bunch of jackkninfed Timescape models.

If the model is based on spherical coordinates, mind that equal angles do not span equal distances, in particular, the polar areas are somewhat oversampled with respect to the tropical ones. Whenever the spatial coverage of the model includes a large latitudinal gradient, it is advisable to split the model in an equatorial–mid latitudes one and a polar one.

On practical grounds, the summation in equation 30 has to be be performed only on a few $\mathbf{S}'$ sub-sources, since the running time of each subset is comparable with that of the full Timescape model.

2.9 Summary

In order to represent an actual measurement of a physical quantity, including the moment and the position at the time of recording, it is introduced the term *event* $\boldsymbol{p}$ as an element of a *spacetime* with coordinates (x, y) at time t: $\boldsymbol{p} = (t, x, y;\, v)$, where v is the value of the measured quantity. $\boldsymbol{E}$ is the set of all the events, it is a subset of $\mathbb{R}^3$ (projected coordinates) or $S^2 \times \mathbb{R}$ (spherical coordinates) times one more $\mathbb{R}$ for the values – supposing the measure can be represented as a real scalar: a so-called *scalar field* over $\boldsymbol{E}$.

Firstly, a distance on $\boldsymbol{E}$ is defined, smoothly extending the metric structure of the two-dimensional spaces (eq. 1 to 4) to the time coordinate, converted in spatial units. Then a causality constraint is included in such distance: the general idea is that every phenomenon takes time to propagate its effects, i.e. the scalar field has a finite speed of propagation. This is

achieved by filtering topologically (section 2.3) the elements of $\boldsymbol{E}$: for each $\boldsymbol{p} \in \boldsymbol{E}$ the spacetime is partitioned into three distinct subsets (figure 6):

- the *past causal cone* – it is the set of the possible causes of $\boldsymbol{p}$, labelled as $\boldsymbol{q}$ in the figure, this set is called S_p in equation 23, these elements have to be considered in the interpolation of the value of $\boldsymbol{p}$.

- the *future causal cone* – it is the set of the possible outcomes of $\boldsymbol{p}$, labelled as $\boldsymbol{r}$ in the figure, since these elements of $\boldsymbol{E}$ occur after $\boldsymbol{p}$, they are not included in its value's interpolation.

- all the rest – these elements are not causally connected to $\boldsymbol{p}$, since they are too far from $\boldsymbol{p}$ to propagate their influence, these elements are labelled as $\boldsymbol{s}$ in the figure.

The purpose of a spatiotemporal interpolation algorithm is to predict the value of a variable for some events, given the knowledge of such a variable for some other events. Call $\boldsymbol{S}$ (for *source*) the finite input set of measurements and $\boldsymbol{T}$ (the *target*) a set of events whose values are to be guessed. The Timescape target, or *model bulk* consists in a regular lattice of voxels (i.e. volume elements) within given bounds. The $\boldsymbol{T}$ events can be indexed equivalently by their coordinates (t, x, y) or by their lattice indices (k, i, j) – see equation 19.

Operatively[20] the Timescape algorithm consists in applying one's favourite two-dimensional spatial interpolation algorithm considering, for each $\boldsymbol{p} \in \boldsymbol{T}$, only its S_p (i.e. the subset of events in $\boldsymbol{S}$ that are causally connected to $\boldsymbol{p}$) instead of the whole $\boldsymbol{S}$. This selection is governed by two parameters, c and k, whose roles are:

- The time-space conversion factor c transforms $t \longmapsto c\,t$, dimensionally, it is a velocity[21]. It is loosely related to the speed of propagation of the measured quantity or –better said– to our knowledge of it.

- The causal constraint k (adimensional) controls the aperture of the causal cones; in terms of cone tip solid angle, $\Omega = 2\pi\big(1 - \cos\arctan k\big)$ srad, so $k = 0 \Longrightarrow \Omega = 0$ and $k \to \infty \Longrightarrow \Omega \to 2\pi$. The bigger k, the looser the constraint, the larger S_p (figure 7).

[20] For the sake of clarity, in this section we consider only the Euclidean distance.
[21] Or an angular velocity if geographical coordinates are used.

The value of $\boldsymbol{p}$ is a weighted sum of the values of the elements in S_p, i.e.

$$v_p \;=\; \sum_{\boldsymbol{q}\in S_p} w\big[D(\boldsymbol{p},\boldsymbol{q})\big]\, v_q \tag{31}$$

where the distance

$$D(\boldsymbol{p},\boldsymbol{q}) \;=\; \sqrt{c^2(t_p-t_q)^2+(x_p-x_q)^2+(y_p-y_q)^2} \tag{32}$$

and $w[\cdot]$ is a suitable distance-based weight function (see sections 2.6 and 3.2). The condition for $\boldsymbol{q}$ for being or being not in S_p is[22]

$$\begin{aligned} &\boldsymbol{q}\in S_p \;\text{ if }\; t_p\geq t_q \;\text{ and }\; \sqrt{(x_p-x_q)^2+(y_p-y_q)^2} \;\leq\; k\,c\,(t_p-t_q) \\ &\boldsymbol{q}\notin S_p \;\text{ otherwise} \end{aligned} \tag{33}$$

Notice that $c\,|t_p-t_q|$ and $\sqrt{(x_p-x_q)^2+(y_p-y_q)^2}$ are the sides of the triangle whose hypotenuse is the distance (32) as in figure 5: in fact $k\,c\,|t_p-t_q|$ is the maximum allowed spatial distance in the time span t_p-t_q – this construction is borrowed form the Minkowskian relativistic spacetime structure [1]. It is also possible to account for seasonal variability, shrinking and expanding periodically the causal cone – see section 2.3 for details.

[22] Equation (33) restricts (13) to the simpler Euclidean metric case.

3 The Algorithm

In this section the procedural details of the Timescape algorithm are discussed, independently of the actual implementation. Section 4 is entirely devoted to the Python module functions.

3.1 Modeling

A *Timescape model* is a set consisting of:

- an input array of N source events $\boldsymbol{S}$, whose values are known from measurement: $\boldsymbol{S} = \big\{\boldsymbol{q}_a = (t_a, x_a, y_a; v_a),\ a = 1 \dots N\big\}$,
- an output lattice $\boldsymbol{T}$ of $K \times I \times J$ target events $\big\{\boldsymbol{p}_{kij} = (k, i, j; v, s, n)\big\}$ where $k = 1 \dots K,\ i = 1 \dots I,\ j = 1 \dots J$, these voxels[23] form the *model bulk* bounded by $[t_{\mathsf{min}}, t_{\mathsf{max}}] \times [x_{\mathsf{min}}, x_{\mathsf{max}}] \times [y_{\mathsf{min}}, y_{\mathsf{max}}]$,
- a parameter c, the time to space conversion factor $t \longmapsto c\,t$,
- a parameter k and (optionally) the period T, that define the width and shape of the causal cones,
- the maximum number of source events to be considered as near primes n_{max} (also an optional parameter),
- a metric function for the spatial distance D_s chosen among `EUCLID`, `SQUARE`, `DIAMOND` and `SPHERE` (equations 1 to 5),
- an ordinary spatial interpolation algorithm, like Kriging or Inverse Distance Weighting.

The Timescape algorithm interpolates the values of every $\boldsymbol{p} \in \boldsymbol{T}$ independently of each other; the resulting value is null if the causal constraints are not satisfied, or the output of the chosen spatial interpolator[24]. The user provides an input file containing the model parameters and the source events. Operatively, a program's run goes as follows:

[23] Note that, while the events of $\boldsymbol{S}$ are explicitly referred to their coordinates t, x, y, the events of $\boldsymbol{T}$ are identified via their lattice node indices k, i, j (eq. 19). v and s are initialised as null, while n is initially 0 and will eventually store the number of events in S_p.

[24] Statistical interpolators provide a standard deviation estimate, too.

1. the model parameters and source events $\boldsymbol{S}$ are read from the input file (algorithm 2)

2. the target $\boldsymbol{T}$ is initialised with $K \times I \times J$ empty voxels (algorithm 2)

3. for each event $\boldsymbol{p} \in \boldsymbol{T}$, its S_p (eq. 21) is extracted from $\boldsymbol{S}$ (algorithm 3)

4. the value v_p of $\boldsymbol{p}$ is evaluated by the chosen interpolator running only on the S_p events (algorithm 3 and 4, section 3.2)

5. the target $\boldsymbol{T}$ is returned to the user (algorithm 2)

Algorithm 2 Timescape

procedure TIMESCAPE (`infile`)
  algorithm, metric, $R \leftarrow$ `infile` ▷ default `KRIG`, `EUCLID`, $R_{\oplus}$
  $K, I, J, [t_{\mathsf{min}}, t_{\mathsf{max}}], [x_{\mathsf{min}}, x_{\mathsf{max}}], [y_{\mathsf{min}}, y_{\mathsf{max}}] \leftarrow$ `infile` ▷ target size and limits
  $c, k, T, n_{\mathsf{max}} \leftarrow$ `infile` ▷ causal parameters, default $n_{\mathsf{max}} = 0$
  $\boldsymbol{S} \leftarrow$ `infile` ▷ read source events
  $\omega \leftarrow T$ is null ? null : $\frac{\pi}{cT}$ ▷ convert period T to frequency ω
  $\boldsymbol{T} \leftarrow \varnothing$ ▷ empty target triple array [[[]]]
  for $k, i, j = 1 \dots K, 1 \dots I, 1 \dots J$ **do**
    $\boldsymbol{T} \leftarrow \boldsymbol{T} \bigcup \{\boldsymbol{p} = (k, i, j;\, \mathsf{null}, \mathsf{null}, 0)\}$ ▷ add new empty $\boldsymbol{p}$ to $\boldsymbol{T}$
  for all $\boldsymbol{p} \in \boldsymbol{T}$ **do**
    $\boldsymbol{p} \leftarrow$ EVAL $(\boldsymbol{p}; \mathsf{algorithm}, \mathsf{metric}, R, \boldsymbol{S}, c, k, \omega, n_{\mathsf{max}})$ ▷ evaluate $\boldsymbol{p}$ (alg. 3)
  return $\boldsymbol{T}$ ▷ the output object

The memory management strategy consists in instancing a *target bulk* prior to the actual interpolation of the target events, the reason is that this operation can fail due to the large number of elements of $\boldsymbol{T}$. Next, every $\boldsymbol{p} \in \boldsymbol{T}$ is interpolated (this phase could be easily parallelised): first, all the elements of $\boldsymbol{S}$ and their distances are temporarily included in S_p, a null distance means that such given element of $\boldsymbol{S}$ lies outside $\mathcal{C}_p^-$, than all the $d = \mathsf{null}$ elements are removed from S_p (algorithm 3). The actual evaluation is carried over by any suitable spatial interpolator, see section 3.2.

A complex and time-consuming phase is the evaluation of the distance (23) of each $\boldsymbol{q} \in \boldsymbol{S}$ from each $\boldsymbol{p} \in \boldsymbol{T}$ (algorithm 4), since the distance-evaluating function has to be called $N \times K \times I \times J$ times. The null value is allowed, meaning that $\boldsymbol{q} \notin \mathcal{C}_p^-$. The distance function also computes the form factor $\Psi(k, D_t)$ that is 1 for a straight cone (i.e. $T = \mathsf{null}$) or $\cos^2 \frac{\pi D_t}{cT}$ if the cone is periodic ($T \neq \mathsf{null}$). Operatively, the frequency $\omega = \frac{\pi}{cT}$ is calculated beforehand[25] and used instead of T, so the simpler (and faster)

[25]The ternary operator in alg.2 and 4 reads: condition ? value_if_true : value_if_false.

Algorithm 3 Target voxel evaluation

procedure EVAL ($\boldsymbol{p}$; algorithm, metric, R, $\boldsymbol{S}$, c, k, ω, $n_{\max}$)
  $S_p \leftarrow \varnothing$ ▷ prepare the causal cone of $\boldsymbol{p}$
  for all $\boldsymbol{q}_a \in \boldsymbol{S}$ **do**
    $d_a \leftarrow$ DIST ($\boldsymbol{p}$, $\boldsymbol{q}_a$, metric, R, c, k, ω) ▷ $D(\boldsymbol{p}, \boldsymbol{q}_a)$ see eq. 6 (alg. 4)
    $S_p \leftarrow S_p \bigcup \{ (\boldsymbol{q}_a, d_a) \}$ ▷ add the $(\boldsymbol{q}_a, d_a)$ pair to S_p
  $S_p \leftarrow \{ (\boldsymbol{q}_a, d_a) \in S_p \mid d_a \neq \text{null} \}$ ▷ remove $\boldsymbol{q}_a \notin \mathcal{C}_p^-$ (null distances)
  $S_p \leftarrow \text{order}(S_p, d)$ ▷ order S_p by distance
  if $n_{\max} \neq 0$ **then**
    $S_p \leftarrow \text{prune}(S_p, n_{\max})$ ▷ retain only the first $n_{\max}$ elements
  if S_p is empty **then**
    $v_p, s_p = \text{null}, \text{null}$
  else
    $n_p \leftarrow \# S_p$ ▷ the number of events in S_p
    $v_p, s_p \leftarrow \text{algorithm} (S_p)$ ▷ apply the interpolator (sec. 3.2 and 4.6)

$\cos^2 \omega D_t$ is evaluated instead of $\cos^2 \frac{\pi D_t}{cT}$. Figure 8 shows a straight ($\Psi = 1$) and a periodic cone; the half-period $\frac{\pi}{cT}$ is used instead of $\frac{2\pi}{cT}$ since the period of the cosine squared is π.

Algorithm 4 Distance from $\boldsymbol{q}$ to $\boldsymbol{p}$: $D(\boldsymbol{q} \rightarrow \boldsymbol{p})$

procedure DIST ($\boldsymbol{p}$, $\boldsymbol{q}$; metric, R, c, k, ω)
  $d \leftarrow$ null
  if $t_p \geq t_q$ **then** ▷ d is null if $\boldsymbol{q} \in \mathcal{F}_p$, fig. 6
    $D_t \leftarrow c\,(t_p - t_q)$
    $\delta x, \delta y \leftarrow |x_p - x_q|, |y_p - y_q|$
    $\Psi \leftarrow \omega$ is null ? 1 : $\cos^2 \omega D_t$ ▷ form factor, eq. 18
    if metric is `EUCLID` **then**
      $D_s \leftarrow \sqrt{\delta x^2 + \delta y^2}$ ▷ spatial metric, eq. 1
    else if metric is `SQUARE` **then**
      $D_s \leftarrow \max \{ \delta x, \delta y \}$ ▷ eq. 2
    else if metric is `DIAMOND` **then**
      $D_s \leftarrow \delta x + \delta y$ ▷ eq. 3
    else if metric is `SPHERE` **then**
      $\lambda_p, \phi_p \leftarrow \pi x_p/180, \pi y_p/180$; $\lambda_q, \phi_q \leftarrow \pi x_q/180, \pi y_q/180$
      $D_s \leftarrow R \arccos \left[\sin \phi_p \sin \phi_q + \cos \phi_p \cos \phi_q \cos (\lambda_p - \lambda_q) \right]$ ▷ eq. 4
    if $D_s \leq k\, \Psi\, D_t$ **then**
      $d \leftarrow \sqrt{D_t^2 + D_s^2}$ ▷ distance, eq. 23; d is null if $\boldsymbol{q} \notin \mathcal{C}_p^-$, fig. 6
  return d

3.2 Interpolators

The line $v_p, s_p \leftarrow$ algorithm (S_p) in algorithm 3 points to the last logical step of the Timescape algorithm: the actual evaluation of the value of the target events. Each value v_p is a function of the causes of $\boldsymbol{p}$, to be evaluated by the proper algorithm.

Operatively, all the interpolators will be implemented as methods of a dedicated class, see section 4.6. Basically, there are two ways in which the

elements of S_p can be used: it is possible that only the distance-value pairs suffice to calculate v_p, or that all the coordinates of the $\boldsymbol{q}_a$ have to be used. The former case corresponds to the Inverse Distance Weighting family of interpolators, the latter, generally speaking, to the statistic ones. The great advantage of statistical interpolators is that they provide an estimate of the goodness of the guessed value v_p as a variance figure $\sigma^2(v_p)$: in fact, these interpolators minimise such variance through optimisation. This confidence estimate is reported in the target events $\boldsymbol{p} \in \boldsymbol{T}$ as $s_p = \sqrt{\sigma^2(v_p)}$.

3.2.1 Deterministic Interpolators

The deterministic methods included in the Timescape software are a sharp and a smoother version of the inverse distance interpolation [17]. These methods apply a weighted sum to the values of the elements of S_p. The weights are decreasing functions of the distance and must sum to one. For each $(\boldsymbol{s}_a, d_a) = ((t_a, x_a, y_a; v_a), d_a) \in S_p$ it is computed a weight $w(d_a)$, so that the value of $\boldsymbol{p}$ is given by

$$v_p \;=\; \sum_a w(d_a) v_a \;, \qquad \sum_a w(d_a) \;=\; 1 \tag{34}$$

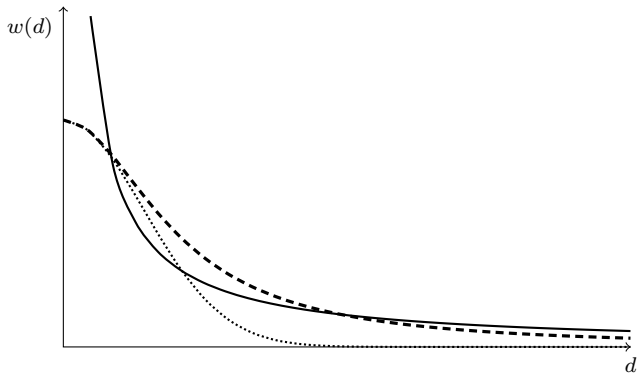


Figure 11: Sharp (solid) and smooth (dashed) weight functions, compared with e^{-d^2} (dotted).

The `IDW` interpolator uses a sharp weight function (figure 11)

$$w_{\mathsf{sharp}}(d) \;=\; \frac{1}{d} \tag{35}$$

some caution is due to avoid a division by zero in case $d_a = 0$, this is only possible if the centre of the voxel representing $\boldsymbol{p}$ is exactly at the same place

and at the same time of some $\boldsymbol{q}_a$: if this is the case, one can put[26] $v_p = v_a$.

The `SIDW` interpolator uses a smooth weight function instead

$$w_{\mathsf{smooth}}(d) \;=\; \frac{1}{d^2 + m^2} \tag{36}$$

the positive parameter m^2 can be set by the user, its default value is 1. Figure 11 shows how w_{smooth} compares with w_{sharp} and with a Gaussian weight e^{-d^2}. There is no divergence for $d = 0$ and, in general, the w_{smooth} weight does not increase as fast as w_{sharp} for small d.

The `IDW` and `SIDW` algorithms evaluate v_p, s_p as

$$v_p \;=\; \frac{\sum_a w_a\, v_a}{\sum_a w_a}\;, \qquad\qquad s_p = \mathsf{null} \tag{37}$$

the sums run over S_p. The vast IDW family is populated with many variants of (35) that can be easily added to the Timescape Python module [17].

3.2.2 Statistical Interpolators

On the side of statistical interpolators, the universal Kriging method is implemented as `KRIG`, exploiting the PyKrige package [21] that provides 3D ordinary Kriging and 3D universal Kriging [17]; the `KRIG` implementation is minimal as it is intended as a mere starting point to be tuned to the users' needs. An informed use of the Kriging method requires the knowledge of the properties of the distribution of the values, in particular, one needs to know the variogram of the values vs their distance and the trend (if any) in space and time.

Before considering Kriging, it is advisable to check the variogram of the source events. The Timescape Python module comes with a function to analyze the variogram, trend and distribution of the source events values (figure 12). After partitioning the set of all the spatiotemporal distances $D(\boldsymbol{q} \to \boldsymbol{p})$ into n lags of equal size, identifyed by their average distance h,

[26] If there are multiple $\boldsymbol{s}_a$ at the same place ($D_s = 0$) but at different times, only one of them can have $D = 0$.

define the variogram[27]

$$\gamma(h) \;=\; \frac{1}{\#N(h)} \sum_{\boldsymbol{q},\boldsymbol{p}\in N(h)} \big(\, v_q - v_p \,\big)^2 \tag{38}$$

where

$$N(h) \;=\; \Big\{ \boldsymbol{q},\boldsymbol{p} \in \boldsymbol{S} \;\Big|\; D(\boldsymbol{q}\to\boldsymbol{p}) \neq \mathsf{null} \wedge h - \frac{\delta}{2} \leq D(\boldsymbol{q}\to\boldsymbol{p}) \leq h + \frac{\delta}{2} \Big\} \tag{39}$$

with a lag width

$$\delta \;=\; \frac{1}{n}\Big[\max\big\{D(\boldsymbol{q},\boldsymbol{p})\big\} - \min\big\{D(\boldsymbol{q},\boldsymbol{p})\big\} \Big] \tag{40}$$

and typically $h = \frac{1}{2}\delta, \frac{3}{2}\delta, \ldots, \frac{n-1}{2}\delta$ (the centres of the lags). $\#N(h)$ is the number of elements in $N(h)$ – see figure 12: the little grey dots are all the single $(D, |\Delta v|)$ pairs, while variogram $\gamma(h)$ is represented by the red dots, whose size is proportional to $\#N(h)$. $\gamma(h)$ is based on the distance (algorithm 4), so it varies according to the parameters c, k, T, and to the chosen metric.

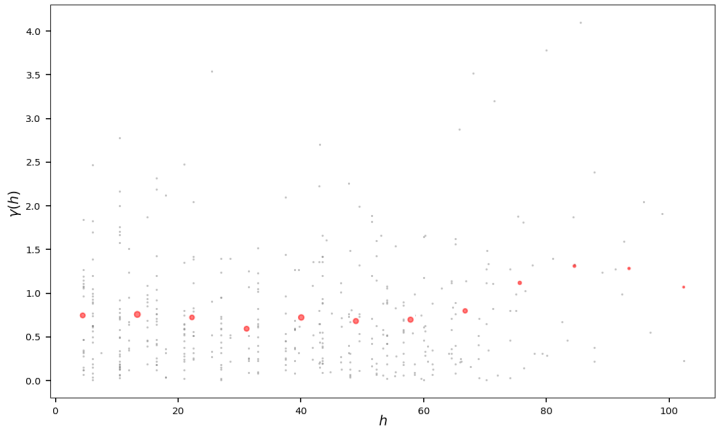


Figure 12: A spatiotemporal variogram of $\boldsymbol{S}$.

The key point here is that, while (38) defines the variogram of the whole $\boldsymbol{S}$, each $\boldsymbol{p}$ has its own variogram $\gamma_p(h)$, that is a downscaled replica of (38),

[27] Equation 38 differs from the standard $\gamma(h) \;=\; \frac{1}{2N}\sum \big(v_q - v_r\big)^2$ for a factor $\frac{1}{2}$, this is due to the the fact that there is not double-counting in our case, since either $D(\boldsymbol{q},\boldsymbol{r}) = \mathsf{null}$ or $D(\boldsymbol{r},\boldsymbol{q}) = \mathsf{null}$ – see (23).

restricted to $\mathcal{C}_p^-$ only:

$$\gamma_p(h) = \frac{1}{\#N_p(h)} \sum_{N_p(h)} \big(v_q - v_p\big)^2 \quad \text{where} \quad N_p(h) \,=\, N(h) \cap \mathcal{C}_p^- \tag{41}$$

note, however, that since $N_p(h)$ is much smaller than $N(h)$, the behaviour of $\gamma_p(h)$ cannot be accurately inferred from $\gamma(h)$.

The 3D universal Kriging [17, 21] needs all the coordinates to evaluate the variogram, using a linear variogram model[28]. A minimum of three elements in S_p are needed to give a result, although the statistical methods are expected to be used with a larger number of samples.

As in the case of deterministic interpolators, also Kriging estimates the unknown values of the scalar field v_p at $\boldsymbol{p} \in \boldsymbol{T}$ as weighted sums

$$v_p \;=\; \sum_{\boldsymbol{q} \in S_p} w_p(\boldsymbol{q})\; v_q \tag{42}$$

but in this case the weights $w_q(\boldsymbol{p})$ are not just decreasing functions of the distance, like (35) or (36). These $w_p(\boldsymbol{q})$ are inferred from the $\gamma_p(h)$ following some assumptions (41). Universal Kriging, in particular, assumes a polynomial trend model [17], which is often suited for ecological datasets. A word of caution is in order in the case of geographical coordinates: since the latitude is singular at the poles and the longitude is periodic, the spatial trend of the values is often periodic too, so a polynomial interpolation trend is hardly reliable.

[28] The PyKrige package offers much more opportunities [21]. In general, the variogram model and the best-suited Kriging variant depend on the nature of the data [17].

4 The Python Module

The Timescape Python software philosophy consists in keeping the usage complexity and the hardware requirements at a minimum. The distribution consists in a single lightweight module, named `timescape.py`. The code has been developed in Python3, version 3.7.

The usage can be terminal-based; the same module's functions can be imported in the users' own code as well. The output model can be saved in a number of ways, including multi-layered geotiff imagess [18] to be used in a GIS environment. The source events and the model's parameters are supposed to be formatted in a single input text file.

The Timescape module usage is described with a novice/intermediate user in mind. The module has to be imported as usual: for example, one can import the module with `import timescape as ts`, than call the functions as `ts.function_name(arguments)`.

4.1 Source Preparation

A Timescape model evaluation is a complex task that involves a series of steps to be done before actually running the interpolation. The `timescape` package comes with a few source analysis functions to help users prepare their input files.

The input dataset (the source) has to be provided as a plain ASCII text file. The input file has to be formatted according to the following schema:

```
#this is a comment - parameters first
ALGORITHM=KRIG, NEIGH=0
METRIC=EUCLID, C=1.5, K=1.0
NT=50, MINT=0.0, MAXT=100.0
NX=100, MINX=0.0, MAXX=144.01
NY=100, MINY=0.0, MAXY=122.59
#then the source events follow
ID,T,X,Y,VAL
A_LABEL,34.00,144.01,0.00,8.87
ANOTHER_LABEL,37.00,144.01,0.00,7.03
   ...
```

The rules are:

- spaces and blank lines are ignored,

- comment lines must start with a `#`,
- the parsing is case-insensitive,
- the parameters can be written in any order, separated by a comma or a new line,
- the events follow the mandatory header line `ID,T,X,Y,VAL`; each event consists in:
  - `ID`: an alphanumeric label[29]
  - `T`: the time coordinate, in time units
  - `X`: the horizontal coordinate, in space units, or the longitude, in degrees, + East, - West
  - `Y`: the vertical coordinate, in space units, or the latitude, in degrees, + North, - South
  - `VAL`: the source event value – cannot be empty

The parameters that should be provided are detailed in tables 1 and 2:

parameter		type	default	description
`ALGORITHM`	algorithm	string	`KRIG`	The spatial interpolator: `KRIG`, `IDW`, `SIDW`, `MYINTERP` or a user-defined one (sec. 4.6).
`NEIGH`	n_{max}	int ≥ 0	0	The neighbourhood elements cutoff – zero means no limit (alg. 3).
`METRIC`	metric	string	`EUCLID`	the spatial metric: `EUCLID`, `SQUARE`, `DIAMOND` or `SPHERE` (eq. 1 to 5, sec. 2.1).
`RADIUS`	R	float > 0	$R_\oplus$	The radius, if metric is `SPHERE` [5].
`C`	c	float ≥ 0	—	The time to space conversion factor.
`K`	k	float ≥ 0	—	the causal cone aperture, adimensional.
`KPERIOD`	T	float > 0	null	The cone shrinking period (eq. 16, fig. 8).
`MYPAR_*`		string	—	user-defined parameters

Table 1: The Timescape algorithm parameters. The first column shows the parameters names as they should be written in the input file, the second column contains their names as used in sections 2 and 3.

Table 1 contains all the algorithm-related parameters, their detailed description can be found in sections 2 and 3. Some of these depend on the physical nature of the source (the metric and possibly a seasonal period), some are a matter of choice (the algorithm and optionally the user-defined

[29] `ID` is intended as the event-id i.e. it should be unique and not-null, however this field is never used by Timescape.

parameters). The most important parameters, c and k, can be set with the help of the dedicated `source_ensemble` function.

parameter		type	description
`NT`	K	int > 0	The bulk sheets (eq.19, sec.2.5).
`NX`	I	int > 0	The bulk rows (eq.19, sec.2.5).
`NY`	J	int > 0	The bulk columns (eq.19, sec.2.5).
`MINT, MAXT`	$[t_{\mathsf{min}}, t_{\mathsf{max}}]$	float	The time bounds – must be $t_{\mathsf{min}} \leq t_{\mathsf{max}}$.
`MINX, MAXX`	$[x_{\mathsf{min}}, x_{\mathsf{max}}]$	float	The x bounds – must be $x_{\mathsf{min}} \leq x_{\mathsf{max}}$.
`MINY, MAXY`	$[y_{\mathsf{min}}, y_{\mathsf{max}}]$	float	The y bounds – must be $y_{\mathsf{min}} \leq y_{\mathsf{max}}$.

Table 2: The target events lattice (i.e. *model bulk*) parameters.

Table 2 contains the target bulk related parameters. These parameters describe the geometry of the target voxels lattice (section 2.5). The total number of elements is $K \times I \times J$, there is no need to keep I, J, and K of the same magnitude: if one is more interested in temporal rather than spatial resolution K can be much bigger than I and J, and vice versa.

The parameters starting with `MYPAR_` are supposed to be used in custom interpolators. The interpolators are treated in detail in section 4.6.

4.2 Source Analysis

The `timescape` module contains a bunch of functions to help defining the parameters values. The first step consists in writing an input file as discussed above, this file serves as the input[30] of the source analysis functions.

`source_dist` – this function plots (via matplotlib [25]) the spatiotemporal variogram, an interactive 3d-scatterplot of the source events, color-coded, and box-violin plot of the distribution of all the values, as in figure 13.

The function syntax is `source_dist(input_file)`.

this function also outputs in the standard output (i.e. the terminal) the $\big(h, \gamma(h)\big)$ pairs of the variogram (the red dots in figure 13) to be used for spatial statistical analysis, in particular if Kriging is involved – see section 3.2.2, since the variogram values are only a dozen of pairs, they are not saved to a file. The box plot shows the usual quartiles, the mean (in blue) and the outliers, while the median and the mode(s) are shown as bump(s) of the outer violin plot. The events' values color scale is provided for reference.

[30] All the functions' argument default is `ts_in.txt`.

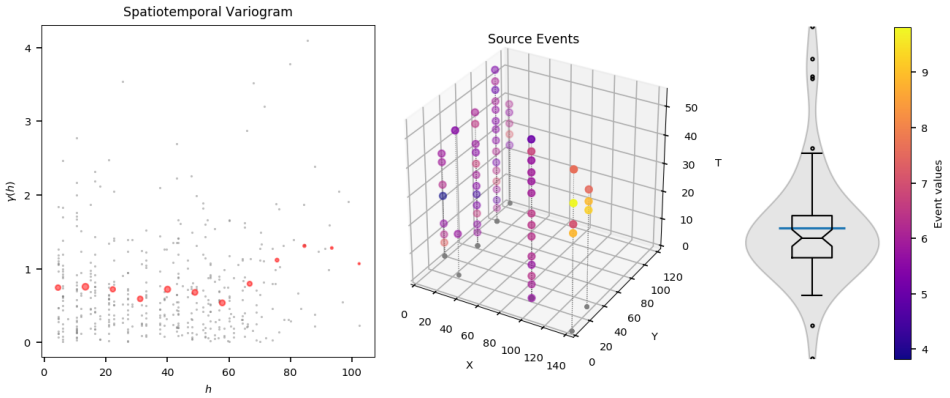


Figure 13: The variogram (left), scatterplot (centre) and box-violin plot (right).

`source_trend` – this function plots the trend of the source values against time and the x and y spatial coordinates as in figure 14. The function syntax is `source_trend(input_file, output_file)`.

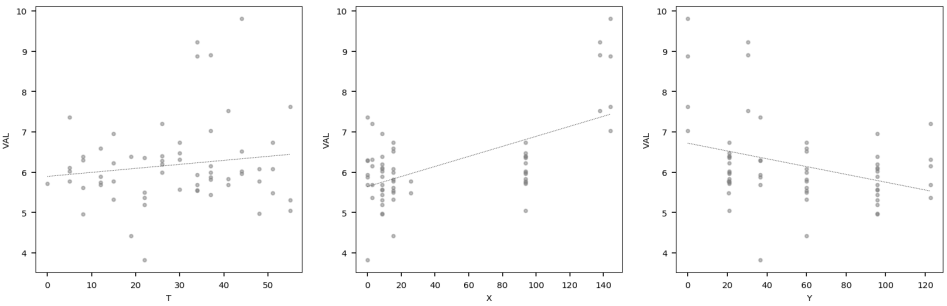


Figure 14: The time (left), x (centre) y (right) trend of the source events values.

this function also saves a text file with a linear trend analysis of the values made with StatsModels [26], the trend is also shown in the plot. The output file[31] is in plain text format.

`source_footprint` – this function plots the distribution of the source values against time and the x and y spatial coordinates, as in figure 15. The function syntax is `source_footprint(input_file)`. This function's purpose is to find if and how the events distribution is skewed, as it is often the case mixing old fashioned and newly collected data.

The spatial footprint bounds are read from the input file, the values on the axes are the actual coordinates. The temporal distribution is shown as

[31] The default output file name is `ts_trend.txt`.

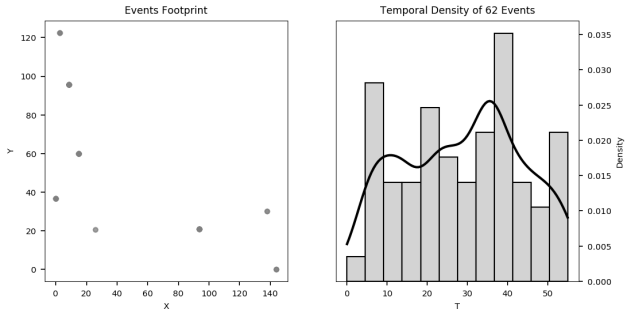


Figure 15: The spatial (left), and temporal (right) distribution of the source events.

an histogram, the values are relative frequencies.

`source_ensemble` – this function is intended to help users find the optimal values of the c and k parameters before running the Timescape interpolation. The function estimates $r(c, k)$ (equation 27) and $n(c, k)$ (equation 28), it also estimates the fraction of `-BAD` events for each element of the ensemble and the throughput as the number of interpolated voxels per second.

The function syntax is[32]

```
source_ensemble(nc, cmin, cmax, nk, kmin, kmax, in_file, out_file)
```

The arguments define an ensemble of c and k values in the $[c_{\mathsf{min}}, c_{\mathsf{max}}]$, $[k_{\mathsf{min}}, k_{\mathsf{max}}]$ range, arranged in an $N_c \times N_k$ pairs lattice. The resulting ensemble is thus indexed by (c_i, k_j) where $i = 1 \ldots N_c$, $j = 1 \ldots N_k$. The output is presented graphically, as three surface plots (figure 16) and also as comma separated values in the output file.

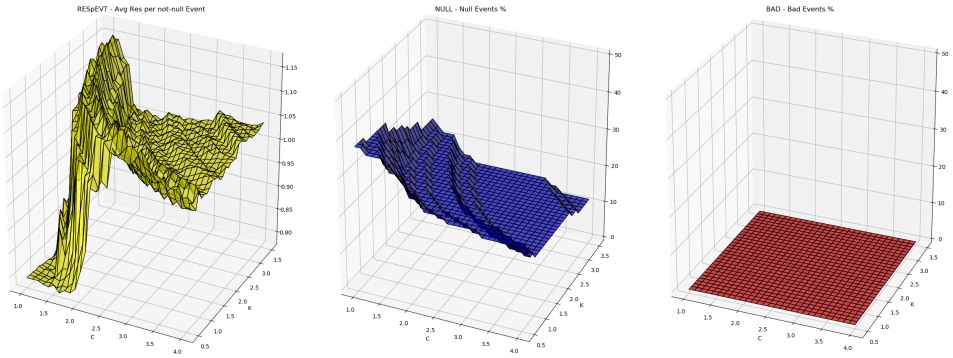


Figure 16: The output of `source_ensemble`: `RESpEVT` (left, in value's units), `NULL` (centre, in %) and `BAD` (right, in %). `BAD` should be completely flat.

[32]The default input and output files are `ts_in.txt` and `ts_res.txt`.

The output file record fields are:

- `C` – the c_i value.
- `K` – the k_j value.
- `SQRES` – the $R^2(c,k)$ of equation 25.
- `RESpEVT` – the $r(c,k)$ of equation 27, shown in figure 16, left.
- `NULL` – the $N(c,k)$ of equation 26, it is also shown in figure 16, center, as percentage of the total number of events, i.e. $[100 \times n(c,k)]$.
- `BAD` – the number of evaluation errors, it is also shown in figure 16, right, as percentage of the total number of events.
- `VXpS` – the algorithm throughput, in voxels per second.

As a matter of principle, the number of evaluation errors should be zero and the number of `null` events as small as possible; a significant `BAD` count means that something in the choice of the algorithm needs to be changed. The throughput is also important as an estimate of the time needed for the Timescape model to be completed, on the same hardware with a similar workload.

4.3 Model Evaluation

A finished Timescape model is the output of the main function of the module, this function takes the input file as input and outputs a `Model` object, the function syntax is[33]:

```
my_model = build_model(source_file)
```

During the function run, a textual progress bar is shown; upon completion the terminal output will be like this:

```
---------- START ----------
Input file parsed, 62 source events found
Bulk space allocated 1,543,503,872 bytes
Timescaping |||||||||||||||||||||||||||||| 100% done
Null events count is 274384 out of 1048576 (26%)
Warning: 3 errors occurred, target events marked as -BAD
Timescape finished in 4974 seconds
Throughput = 210.81 events / s
----------- END -----------
```

[33]The `source_file` argument default is `ts_in.txt`.

The `Bulk space allocated` line tells how large is the memory occupied by the model bulk[34], other useful informations are the `null` events count and percentage, and the number of errors encountered during the program run[35]. A description of each error can be found in the `timescape.log` file[36]. A few target event evaluation errors should not be considered as blocking.

The parameters values are checked before the bulk allocation and any inconsistency is considered a blocking error which is reported to the user, like `Fatal error: Bad T interval [0.0,-100.0]` or `Fatal error: K cannot be negative`. No interpolation is performed whenever a blocking error occurs.

The output object can be explored and subsetted in a number of ways (section 4.4), but mind that it lives in the current Python session: it is advisable to save it as soon as it is build[37]. To save it as a zipped pickle [19] binary file[38] just type

```
my_model.write_tsm(file_name)
```

To retrieve a saved model, analogously type

```
my_model = read_tsm(file_name)
```

All the model-related functions can be called also as methods of the model object, as the `write_tsm` function above demonstrates: it is the equivalent of `write_tsm(my_model, file_name)`.

The `describe_model` function prints a synthetic textual description of the model. The syntax is `describe_model(my_model)` or `my_model.desc()`. The output text reads as this:

```
Timescape description
  62 Source events
  1048576 Target events
  Null target events count is 474384
  Model parameters:
     Algorithm: KRIG, Neighborhood: all Metric: EUCLID
     Time to space conversion factor C=1.5
```

[34] This is close to the output object size.

[35] If no error is encountered, this line reads `Zero errors occurred in target events evaluation`

[36] The log lines are formatted like this: `20191119:16:21:10 T5-X4-Y6 evaluation error: division by zero @line_142`, reporting the timestamp, the target voxel id, the cause of the error and the line number of the instruction that raised the exception.

[37] A somewhat different Timescape Java version is also available [12, 13, 14, 15], based on separate database storage.

[38] The `file_name` argument default is `ts_out.tsm`

```
   Causal cone is straight with K=1.0
   tip angle=1.57 rad, Omega=1.84 srad
   cone coverage is 29% of half-plane
T from 0.0 to 80.0 in 64 sheets: Tk, k=0...63
X from 0.0 to 144.01 in 128 rows: Xi, i=0...127
Y from 0.0 to 122.59 in 128 cells: Yj, j=0...127
Target events voxel size (each):
dT=1.25 (time units) or 1.875 (length units)
dX=1.13, dY=0.96, Area=1.08
Volume=2.02 (length^3 units)
```

The `write_txt` function converts the model's target to a text file. The syntax is `write_txt(my_model, file_name)` or `my_model.write_txt(file_name)`, where `file_name` is the output file[39] name.

The text file header is a replica of the above model description, then the target events are listed according to the following record fields:

- `LABEL` (string) is the event id, it is obtained by the voxel indices (the first voxel is `T0-X0-Y0`). Optionally a `-BAD` suffix is added to mark an evaluation error for that particular event,
- `K` (int) is the sheet index $k = 0 \ldots K-1$,
- `I` (int) is the row index $i = 0 \ldots I-1$,
- `J` (int) is the column index $j = 0 \ldots J-1$,
- `T` (float) is the time t_k corresponding to k (eq. 19),
- `X` (float) is the coordinate x_i corresponding to i (eq. 19),
- `Y` (float) is the coordinate y_j corresponding to j (eq. 19),
- `VAL` (float, `null`) is the event value,
- `STDEV` (float, `null`) is the event accuracy[40],
- `NEIGH` (int) is the number of events in S_p, i.e. the $\boldsymbol{p}$-neighbors.

The `write_tiff` function converts the model's target to three geotiff [18] files. The syntax is `write_tiff(my_model, files_prefix)` or as a method of the model: `my_model.write_tiff(files_prefix)`, where `files_prefix` is the output files prefix[41]. The output files are:

[39] The default output file name is `ts_out.txt`.

[40] The `STDEV` field is `null` whenever the `VAL` field is `null` (obviously) and if the interpolator does not provide an accuracy estimate (sec. 3.2.1).

[41] The default output files prefix is `ts_out`.

- `files_prefix_val.tiff` (float) the target events' values, it is the `VAL` field of the text output.
- `files_prefix_acc.tiff` (float) the target events' accuracies, it is the `STDEV` field of the text output – can be `null` if the accuracy is not estimated.
- `files_prefix_num.tiff` (int) the target events' neighbors, it is the `NEIGH` field of the text output.

Each file has as many layers as the number K of sheets, i.e. `NT`. Each layer is labelled like `TIME=123.456`, reporting the corresponding time.

This function is provided as a link to a standard GIS workflow that most users will probably pursue after their Timescape model is complete. Notice that the coordinates of the geotiff files are correct, but the geographical projection has to be set by the user once the files are imported in her/his GIS.

4.4 Target Exploration

The target exploration functions provided with the `timescape` package include index to coordinate conversion and target subsetting and plotting in a variety of ways.

4.4.1 Index to coordinate conversion

These methods can be called to retrieve the time and coordinates from the lattice indices. The methods are:

- `time` – the syntax is `my_model.time(k)` where `k` (int) is the time sheet index, the output is the corresponding time as a float.
- `place` – the syntax is `my_model.place(i,j)` where `i j` (int) are the spatial indices, the output is the corresponding (x, y) as a pair of floats.
- `coordinates` – the syntax is `my_model.place(k,i,j)` where `k`, `i j` (int) are the three lattice indices, the output is the corresponding (t, x, y) as a triple of floats.

4.4.2 Target Subsetting

These functions extract all the fields of a model as 1-, 2- or 3-dimensional numpy arrays. Users can extract the coordinates, the value, the accuracy and the neighbors of each target voxel. The functions are:

- `extract_bulk` – this function extracts all the target events as a 3-dimensional array. The syntax is `extract_bulk(my_model, field)` or equivalently `my_model.extract_bulk(field)`. The `field` argument[42] can be one of:

  - `FIELD_T` or `'T'`: the time,
  - `FIELD_X` or `'Y'`: the x coordinate,
  - `FIELD_Y` or `'Y'`: the y coordinate,
  - `FIELD_VAL` or `'VAL'`: the voxel value (can be `None`),
  - `FIELD_ACC` or `'ACC'`: the voxel accuracy (can be `None`),
  - `FIELD_NUM` or `'NUM'`: the number of voxel neighbors (int).

  The output is the requested numpy array.

- `extract_sheet` – this function extracts a constant time sheet as a 2-dimensional array. The syntax is `extract_sheet(my_model, k, field)` or `my_model.extract_sheet(k, field)`, where `k` is the sheet index and `field` is the same as above. The output is a numpy array.

- `extract_core` – this function extracts a series of values at a given place as a 1-dimensional array. The syntax is `extract_core(my_model, i, j, field)` or `my_model.extract_core(i, j, field)`, where `i` and `j` are the core indices and `field` is the same as above. The output is a numpy array.

- `time_series` – this function writes a text file with the time series extracted at a given place. The syntax is `time_series(my_model, i, j, file_name)` or `my_model.time_series(i, j, file_name)`, where `i` and `j` are the core indices and `file_name` is the output file name[43]. The output fields are time, value, accuracy and neighborhood.

[42] The default field is `FIELD_VAL`.
[43] The default output file name is `ts_time_series.txt`.

4.4.3 Target Plotting

These functions exploit the matplotlib [25] and Plotly [20] modules to show the target lattice or a subset of it. The functions are:

- `histogram` – this function plots a histogram of the target's events values, as in figure 17. The syntax is `target_histogram(my_model)` or alternatively `my_model.histogram()`. This histogram includes all

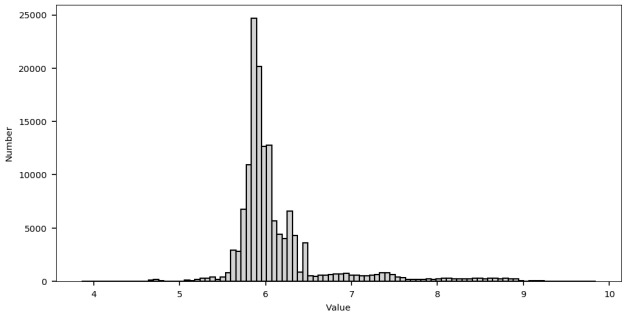


Figure 17: The output of `target_histogram`. The scale shows the values count.

  the target values, for advanced statistical analysis it is recommended to extract the values with the `extract_*` functions described above.

- `plot_bulk` – this function plots a 3d-scatterplot of the target values, just like the scatterplot of the source values. The syntax is `plot_bulk(my_model)` or `my_model.plot_bulk()`. This function is serviceable only with a small number of target voxels, otherwise the output is painfully slow. Refer to the following `show` function for a viable alternative.

- `plot_sheet` – this function plots a constant time sheet's values as color-coded points and isolines. The syntax is `plot_sheet(my_model, k)` or `my_model.plot_sheet(k)`, where `k` is the sheet index. The resulting plot is shown in figure 18. The image can be navigated and saved as usual with a matplotlib output.

- `plot_core` – this function plots a time series as connected color-coded points. The syntax is `plot_core(my_model, i, j)`, where `i` and `j` are the core indices. The resulting plot is shown in figure 18. This is in fact an interpolated time series, the corresponding values for a proper analysis can be obtained with `extract_core(k, FIELD_T)` for the times and `extract_core(k)` for the values.

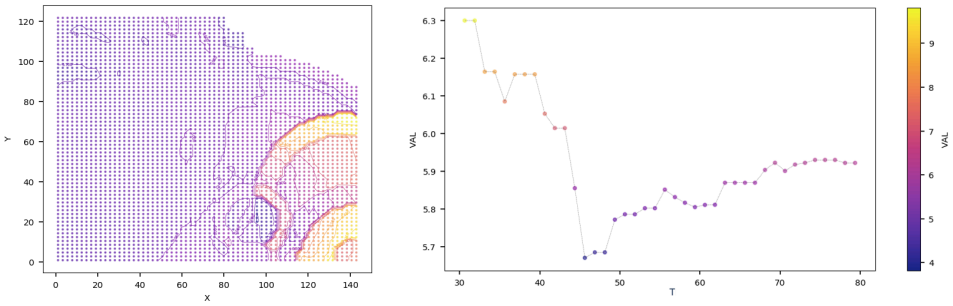


Figure 18: The output of `plot_sheet` (left) and `plot_core` (right).

- `target_show` – this function uses Plotly [20] to show a three-dimensional plot of the value, accuracy and neighborhood of the target. The interactive output is usually shown in the default user's web browser (figure 19).

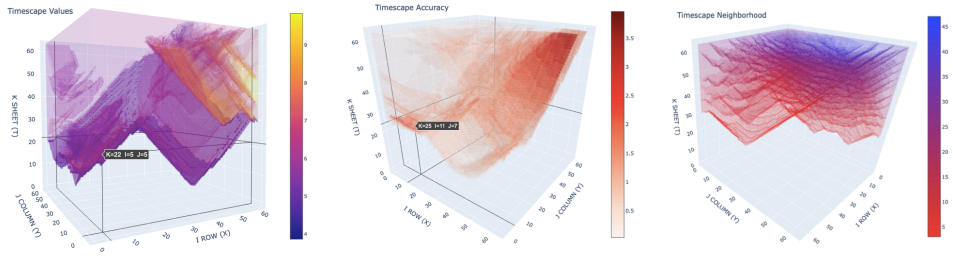


Figure 19: The three possible outputs of `target_show` – left: `FIELD_VAL`: values, center: `FIELD_ACC`: accuracy, right: `FIELD_NUM`: neighborhood.

The syntax is `target_show(my_model, field)` or `my_model.show(field)`, where `field` is a field chosen from `FIELD_VAL` for the values (float), `FIELD_ACC` for the accuracies (float) and `FIELD_NUM` for the neighbors (int). The resulting graphics is CPU and GPU demanding. The color scale of the values is the same of the other plot functions, the accuracy is shown in a white-red scale and the neighbors are colored according to a red-blue ramp.

A few constants in the `timescape` module can be edited to change the appearance of the graphics, they affect the output of all the functions: `PLOT_COLORS`, default `'plasma'`, is the color ramp name; `PLOT_ALPHA`, default `0.5`, is the transparency of most objects; `PLOT_DPI`, default `150` dots per inch, is the resolution of the matplotlib output, the latter does not affect `target_show`.

4.5 Module Objects

The code style is mostly object-oriented. The main stream is just the repetition, event-by-event, of the same interpolator. The data are stored in objects that mimic the logical structures of the Timescape algorithm, as described in sections 2 and 3. In the following are described the `Model` object, the data containers and the exceptions:

Model – this object is the container of a Timescape model. The output of `build_model` is an instance of `Model`, it contains the parameters, the source and the target events. `Model` has a lot of methods, most have been described in the previously (sections 4.3 and 4.4). The remainig methods are:

- `time` – this method returns the time corresponding to a given index `k`. The syntax is `my_model.time(k)`.
- `place` – this method returns the (x, y) spatial coordinates corresponding to the given indices `i` and `j`. The syntax is `my_model.place(i, j)`, the method outputs two floats.
- `coordinates` – this method returns the (t, x, y) spatiotemporal coordinates corresponding to the given indices `k`, `i` and `j`. The method syntax is `my_model.coordinates(k, i, j)`, it outputs three floats.
- `voxel` – this method returns the `Voxel` object (an element of the model target) which contains the spatiotemporal coordinates (t, x, y). The syntax is `my_model.voxel(t, x, y)`, the arguments are floats. This method raises an exception if (t, x, y) does not belong to the target bulk.

Event An `Event` is the representation of the event $\boldsymbol{p} = (t, x, y\,;\, v)$, its value v cannot be `null`. The source events $\boldsymbol{S}$ are stored in the model as an array of events – `[Event]`.

Voxel A `Voxel` is an altenative representation of an event that is part of a lattice. In lieu of the coordinates it uses integer indices: $\boldsymbol{p} = (k, i, j\,;\, v, s, n)$, where s is the accuracy of the interpolated value and n the number of neighbors, i.e. the number of events in S_p. Both v and s can be `null`, of course if $v =$ `null` the s cannot be evaluated. The target events $\boldsymbol{T}$ are stored in the

model as a triple array of voxels – `[[[Voxel]]]`, nested according to a sheet - row - column order.

EventWrapper This object is used to store the event-distance pairs S_p (equation 21) to be used to interpolate the value of each $\boldsymbol{p} \in \boldsymbol{T}$. In fact, an `EventWrapper` is just an `(Event, float)` pair. This utility class is used to feed the actual interpolator with the appropriate input events for each $\boldsymbol{p}$: the S_P is passed to the interpolator as an array of event wrappers – `[EventWrapper]`.

Exceptions The `timescape` module defines two extensions of `Exception`:

- `TFEx` – The Timescape Fatal Exception. It is used internally for blocking errors and it is also raised by some functions, e.g. to signal out of range arguments or i/o errors.
- `TEvalEx` – The Timescape Evaluation Exception. It is used only internally to manage the interpolation errors without stopping the model workflow. This exception is never raised outside the module.

4.6 Interpolators

The interpolators are implemented as methods of the `TimescapeEvaluator` class. This class is structured to be easily customised, adding new methods. The statement $v_p, s_p \leftarrow \mathsf{algorithm}\left(S_p\right)$ of algorithm 3 works this way[44]:

```
voxel_evaluator = TimescapeEvaluator(causes, voxel)
voxel.val, voxel.stdev = getattr(voxel_evaluator,
                                 Model.params['ALGORITHM'])()
```

First, an instance of `TimescapeEvaluator` is created for the `voxel` (i.e. the target event) being interpolated; than the appropriate method is called by its name `Model.params['ALGORITHM']`: this is literally the `algorithm` parameter to be configured in the input file. The naming convention for these methods commands all-caps.

The structure of the interpolators has to be as follows:

```
def MEHTOD(self):
    try:
         # evaluate value and stdev...
```

[44] Function `eval_voxel` of `timescape.py`.

```
        return value, stdev
    except Exception as ex: raise TEvalEx(ex)
```

The outer try-except construct has the purpose of intercepting and managing any error that could have occurred during the event's value/stdev evaluation; such non-blocking errors are recorded in the log file and the voxel `bad` flag is raised[45]. The method returns two floats: the voxel value v_p and the accuracy s_p. The value v_p can be either `None` (if there is not enough information in S_p) or not. The s_p can be calculated directly only with statistical interpolators [17], e.g. the default method `KRIG`, otherwise it is `None`.

Each instance of `TimescapeEvaluator` includes two variables: `vox` is the target event $\boldsymbol{p} = (i, j, k\,;\, \mathsf{null}, \mathsf{null}, n)$ ready to be interpolated, `ewx` is S_p as an array of `EventWrapper` objects. It is easy to cycle in S_p to scan each $\boldsymbol{q} \in S_p$ using a for-cycle like `for ew in self.ews`, the loop variable `ew` takes the role of each element of S_p. To access the values encapsulated in `vox` and `ew` one has to call the attributes detailed in table 3.

variable		`vox` (target $\boldsymbol{p}$)	`ew` (source $\boldsymbol{q}$)
time index	k	`vox.k`	—
time in time units	t	`vox.t`	`ew.t`
time in spatial units	ct	`vox.ct`	`ew.ct`
x index	i	`vox.i`	—
x coordinate	x	`vox.x`	`ew.x`
y index	j	`vox.j`	—
y coordinate	y	`vox.y`	`ew.y`
event value	v	—	`ew.val`

Table 3: Extracting the information encapsulated in `vox` and `ew` (the elements of `ews`).

The Timescape Python distribution comes with three interpolators, plus a dummy method to be used as a starting point for the implementation of new user-defined interpolators. The following list is also meant to be a coding guide for the Timescape customization:

- `IDW` – It is the sharp version of the inverse distance weighting algorithm (section 3.2.1). It is the simplest and fastest method of `TimescapeEvaluator`, corresponding to the weighted average $v_p = \left[\sum_{\boldsymbol{q}\in S_p} \frac{v_q}{D(\boldsymbol{p},\boldsymbol{q})}\right] \Big/ \left[\sum_{\boldsymbol{q}\in S_p} \frac{1}{D(\boldsymbol{p},\boldsymbol{q})}\right]$, of equations (34) and (35). The accuracy cannot be estimated.
  The code consists in a simple cycle on S_p:

[45] In this case the voxel label will end with `-BAD`

```
norm, value = 0.0, 0.0
for ew in self.ews:
   weight = 1.0 / ew.dist
   norm += weight
   value += weight * ew.val()
return (value / norm), None
```

- `SIDW` – It is the smooth version of the inverse distance weighting algorithm (equation 36). The the m^2 parameter can be set in the configuration file as the user-defined parameter `MYPAR_SIDW_SQMASS`. To pass its value to the interpolator, the configuration file must include a line like

  ```
  MYPAR_SIDW_SQMASS = 123.456
  ```

  this value is read in the `SIDW` method in this way:

  ```
  sq_mass = Model.params.get('MYPAR_SIDW_SQMASS', 1.0)
  ```

  the default value being 1.0. This is the general schema to be followed in order to pass a user parameter to a custom interpolator.

- `KRIG` – It is the three-dimensional universal Kriging (section 3.2.2), this is the default interpolator. It is based on the PyKrige module [21], this package offers ordinary- and universal-3d, other than 2d Kriging. In this case all the coordinate informations have to be retrieved from S_p and passed to `UniversalKriging3D`. The core part of the code is

  ```
  ... xx = [ew.x for ew in self.ews] ...
  UK3D = PyKrige.UniversalKriging3D(xx, yy, ctt, vv)
  values, variances = UK3D.execute('points', x, y, ct)
  return values[0], math.sqrt(variances[0])
  ```

  where `xx`, `yy`, `ctt`, `vv` are the arrays of the x, y, t (in space units) coordinates and values of the elements of S_p. The Kriging interpolator `UK3D` is called in `points` mode, with a single event to be evaluated. The output of `UK3D.execute` is a couple of single-element arrays, containing the estimated values and variances $[v_p]$ and $[s_p]$, whose first (and only) element is returned in the last line.

- `MYINTERP` – This mock interpolator serves as a *hello world* method, a starting point for the inexperienced user to start from. It also illustrates the Timescape logging feature: to log a `text_to_log` string just use[46]

  ```
  logline('{} {}'.format(self.vox, 'text_to_log'))
  ```

  that writes one line in the `timescape.log` file containing a timestamp, the voxel's fields values and the `text_to_log` string.

We designed the interpolators methods with a novice/average python user as a target, not particularly skilled in object-oriented programming. The code inside the interpolators' methods is typically entirely procedural.

[46] Any exception raised during the evaluation is automatically recorded in the log file.

5 Fine-tuning the Results

In the following, few improvements of the Timescape algorithm are suggested. Section 5.1 is devoted to the interpolation parameters tuning, no coding required, while section 5.2 deals with possible modifications of the distance function itself, requiring some intervention on the code.

5.1 Interpolation Parameters

The Timescape algorithm adds further intricacy to the already complex machinery of geostatistical interpolation. To exploit the potential benefits of this time-aware interpolation technique one has to tune the parameters to best suit the source events' spatiotemporal distribution. To do so, it is advisable to conduct a detailed statistical analysis of the source events by means of the `source_dist`, `source_footprint` and `source_ensemble` analysis functions (section 4.2). The Timescape model parameters are divided in two groups and discussed separately: the algorithm-related (table 1) and the geometry-related ones (table 2).

Algorithm-related parameters The algorithm-related parameters are those presented in table 1. Some of them are related to the physical nature of the input dataset (the metric, the period T – if any, and the radius of the sphere for the S^2 metric). The choice of the optimal spatial interpolator depends on the nature of the data: as a general rule, Kriging is preferred over the deterministic interpolators for its capability to estimate the accuracy of the values. On the other hand, Kriging needs some prerequisites that are not always matched [17] in the spatial and, even worse, spatiotemporal distribution of the source events.

The critical choice is the time-to-space conversion factor c and the causal parameter k. In section 2.7 it is discussed a heuristic approach for choosing their values, a detailed analysis can be obtained running the `source_ensemble` function (section 4.2, figure 16, left), to assist this involved task. It is also advisable to estimate the running times, starting from the measured throughput as a slight underestimate. The best performing c and k can be found visually, looking for a stable minimum of the average residual, a "valley" of more or less equivalent minima to be further examined in detail, recursively. It is also important to keep the number of `null` target events at a fair

share of the total – the same tool (figure 16, centre) also estimates the null events as a function of (c, k); of course, the less the null voxels, the best the interpolation.

The choice of the spatial interpolation algorithm gives each Timescape model its distinctive character. In section 3.2.1 and 3.2.2 are described in detail the deterministic and statistical interpolators. It is worth emphasising, here, that the deterministic interpolators (`IDW` and `SIDW`) are very fast to compute and all the CPU time is practically due to the evaluation of the distances. On the other hand, for statistical interpolators (`KRIG`) the evaluation of spacetime distances is only an "appetiser" in terms of complexity and running times. In this case, it is advisable to put a threshold on the number of source events to be used for the evaluation of the target events. This is precisely the role of the n_{max} parameter.

Geometry-related parameters The *geometry-related parameters* are the boundaries of the Timescape model bulk and the number of target voxels. These parameters should be treated likewise any conventional geostatistical tool. The target spatial extent, in principle, should be contained into the convex hull of the spatial distribution of the source events, otherwise what the Timescape algorithm outputs is –technically– an extrapolation, subject to the caveats of the case. On the temporal side, however, the matter is a bit more complicated; as a matter of principle, it is possible to forecast the target values well after the source events' greatest time. The forecast, however, cannot be prolonged indefinitely; as a rule of thumb, it is advisable a maximum forecast of the same duration of the sample events' time span.

Operationally, one can start evaluating the target voxels with the same space extent of the source events and a time interval such that t_{min} is within the source time span interval $\mathsf{t}_{\mathsf{min}} \in \left[\min\{t_q\}, \max\{t_q\}\right]$, $\boldsymbol{q} \in \boldsymbol{S}$ and t_{max} is between the source maximum time and this maximum time plus the entire source time span, i.e. $\mathsf{t}_{\mathsf{max}} \leq 2\max\{t_q\} - \min\{t_q\}$.

It is advisable to start the interpolation with a downscaled model consisting of just a few thousand voxels (taking I, J and K in the range 10–20, see eq. 19), checking for interpolation errors, before running the full model, with a consistently bigger bulk.

5.2 Loosening Causality

The Timescape algorithm can be modified in various ways. A few adjustments caould be suggested to be explored; some can be implemented using the `timescape` module as-is, some other require a modification of the code.

Weak Causality It consists in letting $k \to \infty$ in equation 9: $\mathcal{C}_p^{k-} \to \mathcal{P}_p$, so that the causal cone of $\boldsymbol{p}$ coincides with its whole past $\mathcal{P}_p$. This is easily achieved, just setting a very large k value. It is worth noting that this does not affect the value of the spacetime distances (equation 32).

Backwards Action It is possible to take into account both forward- and backward-propagation getting rid of the $t_p \geq t_q$ constraint in the definition of the causal cone, modifying the definition of S_p (equation 21) as

$$S_p = \left\{ (\boldsymbol{q}_a, d_a),\ \boldsymbol{q}_a \in \boldsymbol{S} \cap \left(\mathcal{C}_p^- \cup \mathcal{C}_p^+ \right) \right\} \tag{43}$$

It allows retrodiction: the values of past events can be inferred from future ones. To implement backwards action, one has to alter the code of the `_distance` function substituting the line `if deltat < 0.0: return None` with the instruction `deltat = math.fabs(deltat)` that flips $\mathcal{C}_p^-$ onto $\mathcal{C}_p^+$.

The extreme version of this algorithm alteration consists in combining it with weak causality, thus extending each S_p to the whole set $\boldsymbol{S}$.

Metric modification In principle, it is possible to define new distance functions[47], but it requires major interventions on the code, not only on the `distance` function but also on the input parsing. To avoid this, the Timescape Python module already comes with three plane metrics, and the radius of the spherical metric can be defined by the users[48].

If a particular custom distance is needed, it is advisable to modify the spatial coordinates accordingly, presenting such altered `X` and `Y` coordinates in the source file, choosing the `DIAMOND` metric function that just sums the coordinates differences.

[47] As long as these functions satisfy the conditions of non-negativity, symmetry, reflexivity and subadditivity.

[48] Perhaps this is a bit pretentious, but if one is interested in other planets...

References

[1] A. Ashtekar and V. Petkov, editor. *Springer Handbook of Spacetime*. Springer Berlin Heidelberg, 2014. doi:10.1007/978-3-642-41992-8.

[2] S. Banerjee. On geodetic distance computations in spatial modeling. *Biometrics*, 61(2):617–625, 2005. doi:10.1111/j.1541-0420.2005.00320.x.

[3] Neil D. Bennett, Barry F.W. Croke, Giorgio Guariso, Joseph H.A. Guillaume, Serena H. Hamilton, Anthony J. Jakeman, Stefano Marsili-Libelli, Lachlan T.H. Newham, John P. Norton, Charles Perrin, Suzanne A. Pierce, Barbara Robson, Ralf Seppelt, Alexey A. Voinov, Brian D. Fath, and Vazken Andreassian. Characterising performance of environmental models. *Environmental Modelling & Software*, 40:1–20, 2013. doi:10.1016/j.envsoft.2012.09.011.

[4] G. Christakos. *Spatiotemporal Random Fields*. Elsevier, 2017. doi:10.1016/b978-0-12-803012-7.01001-1.

[5] E. E. Mamajek et al. IAU 2015 Resolution B3 on Recommended Nominal Conversion Constants for Selected Solar and Planetary Properties, 2015. arXiv:1510.07674.

[6] B. Efron and C. Stein. The jackknife estimate of variance. *The Annals of Statistics*, 9(3):586–596, 1981. doi:10.1214/aos/1176345462.

[7] B. Efron and R. Tibshirani. Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statistical Science*, 1(1):54–75, 1986. doi:10.1214/ss/1177013815.

[8] G. Schay. *Introduction to Probability with Statistical Applications*. Springer International Publishing, 2016. doi:10.1007/978-3-319-30620-9.

[9] J. Jost. *Geometry and Physics*. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-00541-1.

[10] J. Jost. *Riemannian Geometry and Geometric Analysis*. Springer Berlin Heidelberg, 2011. doi:10.1007/978-3-642-21298-7.

[11] Keith Knight. *Mathematical Statistics*. Chapman & Hall CRC, 2000.

[12] M. Ciolfi. TimescapeGlobal Java application, GNU-GPL 3 license. URL: https://sourceforge.net/projects/timescapeglobal.

[13] M. Ciolfi. TimescapeLocal Java application, GNU-GPL 3 license. URL: https://sourceforge.net/projects/timescapelocal.

[14] M. Ciolfi, F. Chiocchini, M. Mattioni, M. Lauteri. Timescape local spacetime interpolation tool: projected coordinates java standalone application. *Smart eLab*, 10:20–39, 2017. doi:10.30441/smart-elab.v10i0.201.

[15] M. Ciolfi, F. Chiocchini, M. Mattioni, M. Lauteri. Timescape global spacetime interpolation tool: geographic coordinates java standalone application. *Smart eLab*, 11:1–51, 2018. doi:10.30441/smart-elab.v11i0.202.

[16] Avery McIntosh. The jackknife estimation method, 2016. arXiv:1606.00497.

[17] N. Cressie. *Statistics for Spatial Data*. John Wiley & Sons, Inc., 1993. doi:10.1002/9781119115151.

[18] Open Geospatial Consortium. OGC GeoTIFF Standard. URL: https://www.opengeospatial.org/standards/geotiff.

[19] Pickle. Python object serialization. URL: https://docs.python.org/3/library/pickle.html.

[20] Plotly. Python data visualization. URL: https://plotly.com.

[21] PyKrige developers. PyKrige, 2D and 3D Python Kriging implementation. URL: https://pykrige.readthedocs.io.

[22] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, 2018. URL: https://www.R-project.org/.

[23] V. Schroeder. Quasi-metric and metric spaces. *Conformal Geometry and Dynamics of the American Mathematical Society*, 10(18):355–361, 2006. doi:10.1090/s1088-4173-06-00155-x.

[24] T. Frankel. *The Geometry of Physics*. Cambridge University Press, 2009. doi:10.1017/cbo9781139061377.

[25] The matplotlib developers. matplotlib python plotting library. URL: https://matplotlib.org.

[26] The StatsModels developers. StatsModels statistics in python. URL: https://www.statsmodels.org/stable.

[27] V. Beneš, M. Prokešová, K. Staňková Helisová and M. Zikmundová. Space-time models in stochastic geometry. In *Stochastic Geometry, Spatial Statistics and Random Fields*, pages 205–232. Springer International Publishing, 2014. doi:10.1007/978-3-319-10064-7_7.

Index