ISTI Technical Reports

SpaghettiData and SpaghettiPlot: two Python classes for analysing and visualising SST trends)

Oscar Papini, ISTI-CNR, Pisa, Italy

ISTI-TR-2022/001

SpaghettiData and SpaghettiPlot: two Python classes for analysing and visualising SST trends

Papini O.

ISTI-TR-2022/001

This document describes the formalization of a "spaghetti plot" (i.e. a graph that captures the sea surface temperature trends in a target area) as a Python object, for which we defined two custom classes (SpaghettiData and SpaghettiPlot). In particular, we list the attributes and methods of these classes, together with the utilities that we use to create objects belonging to them.

Keywords: Sea surface temperature, Image analysis.

Citation
Papini O., *SpaghettiData and SpaghettiPlot: two Python classes for analysing and visualising SST trends)*. ISTI Technical Reports 2022/001. DOI: 10.32079/ISTI-TR-2022/001.

`SpaghettiData` and `SpaghettiPlot`: two Python classes for analysing and visualising SST trends

Oscar Papini

24th January 2022

In a previous technical report [1] we described a tool which is able to produce a so-called *spaghetti plot*, i.e. a graph that captures the sea surface temperature (SST) trends in a chosen period of time in a target area (see Figure 1). We used the spaghetti plots to study the problem of detecting and classifying mesoscale events patterns in an upwelling ecosystem, obtaining some preliminary but promising results [2; 3].

This report describes the formalization of a spaghetti plot as a Python 3 object, for which we defined two custom classes (`SpaghettiData` and `SpaghettiPlot`). We list the attributes and methods of these classes, together with the utilities that we use to create objects belonging to them.

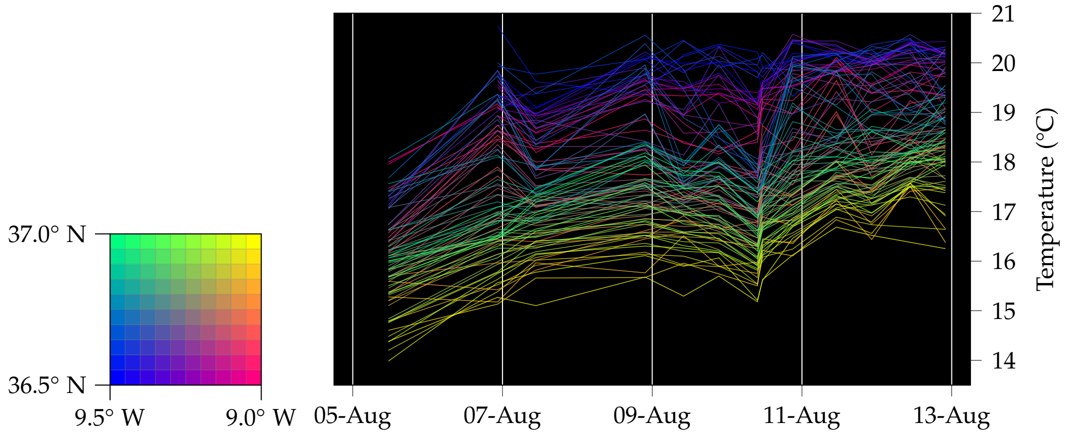

Figure 1: Example of a spaghetti plot. Each graph in the plot on the right describes the trend of the SST in the point with the same colour in the reference grid on the left. (Figure taken from [1].)

1 The `SpaghettiData` and `SpaghettiPlot` classes

1.1 `SpaghettiData`

The `SpaghettiData` class is used to describe a single plot in the global spaghetti plot.

The spatial references of a `SpaghettiData` object are contained in its attributes `latitude`, `longitude`, and `resolution`. These are float numbers representing the point which the SST values refer to; more precisely, if a `SpaghettiData` object has `latitude` $= \ell$, `longitude` $= m$ and `resolution` $= r$, then it is assumed that any SST value in it is computed by averaging some values in the square[▸1] $(\ell; \ell + r) \times (m; m + r)$. Latitude and longitude are expressed in degrees, with positive values corresponding to latitude N and longitude E, and negative ones corresponding to latitude S and longitude W; the resolution is also expressed in degrees, with typical values ranging between 0.05 and 0.25.

The SST values themselves (in degrees Celsius) are contained in the `data` attribute. This is an $n \times 2$ Numpy array where each of the n rows represents a value of the SST in a moment in time. In particular the two entries of a row are, in order, a `datetime.datetime` object and a float for the SST value.

A `SpaghettiData` object is initialised by

```
>>> sdata=SpaghettiData(lat,lon,res,temperatures)
```

where `lat`, `lon` and `res` are floats representing the latitude, longitude and resolution, and `temperatures` is a list of pairs of the form $(\text{time}, \text{SST})$ as in the previous paragraph. The class automatically constructs and populates the Numpy array; moreover, if `temperatures=[]`, then the `data` attribute of the `SpaghettiData` object is a 0×2 array.

1.2 `SpaghettiPlot`

An object of type `SpaghettiPlot` is essentially a collection of `SpaghettiData` objects, all belonging to a delimited target area. In particular, given the extension of this area in terms of the minimal and maximal latitude and longitude (let those numbers be lat, LAT and lon, LON respectively), and a resolution r, the area is divided in a grid, whose points have coordinates (ℓ, m) with

$$\begin{aligned} \ell &\in \{\text{lat} + ir \mid i \in \mathbb{N}\} \cap [\text{lat}; \text{LAT}), \\ m &\in \{\text{lon} + jr \mid j \in \mathbb{N}\} \cap [\text{lon}; \text{LON}). \end{aligned} \tag{1.1}$$

[▸1] If $a, b \in \mathbb{R}$, then $(a; b)$ is the open interval $\{x \in \mathbb{R} \mid a < x < b\}$ and $[a; b]$ is the closed interval $\{x \in \mathbb{R} \mid a \leqslant x \leqslant b\}$. Mixed notation, such as $[a; b)$, is admissible. Notice that $(a; b)$ (with a semicolon) is *different* from (a, b) (with a comma), which denotes the (ordered) pair whose elements are a and b. In any case, in this document the context should provide enough information to avoid confusion.

In other words, if k and h are the cardinalities of the two sets defined in Equation (1.1), we have

$$\ell \in \{\mathrm{lat}, \mathrm{lat}+r, \mathrm{lat}+2r, \dots, \mathrm{lat}+(k-1)r\},$$
$$m \in \{\mathrm{lon}, \mathrm{lon}+r, \mathrm{lon}+2r, \dots, \mathrm{lon}+(h-1)r\}.$$

A `SpaghettiData` object with `latitude` $= \ell$, `longitude` $= m$ and `resolution` $= r$ is then assigned to each point in the grid with coordinates (ℓ, m).

The attributes `min_lat`, `min_lon`, `max_lat`, `max_lon` and `resolution` contain the geographical information of a `SpaghettiPlot` object, with the same conventions for latitudes, longitudes and resolution as those stated in Subsection 1.1.

For simplicity of use in the algorithms, a point of the grid in a spaghetti plot is *not* identified with its geographical coordinates (ℓ, m), but with a pair of integers (i, j) with $i \in \{0, \dots, k-1\}$ and $j \in \{0, \dots, h-1\}$, where k and h are the same as above. The obvious relations between (ℓ, m) and (i, j) are

$$\begin{cases} \ell = \mathrm{lat} + ir \\ m = \mathrm{lon} + jr. \end{cases}$$

The attributes `latitude` and `longitude` contain $k \times h$ Numpy arrays that are used to recover the geographical information of a point, i.e. given (i, j) as above we have

$$\texttt{latitude[i,j]} = \ell,$$
$$\texttt{longitude[i,j]} = m.$$

The attribute `color` contains a $k \times h \times 3$ Numpy array such that `color[i,j]` is the RGB triple (normalized in $[0;1]$) of the colour corresponding to the point (i, j) in the grid (see Figure 1). The colour components are defined as

$$\mathrm{red} = \frac{j}{h-1}, \qquad \mathrm{green} = \frac{i}{k-1}, \qquad \mathrm{blue} = 1 - \frac{\mathrm{red} + \mathrm{green}}{2}$$

so that points near to each other have similar colours.

Finally, the `spaghetti` attribute contains the actual data: it is a Python dictionary indexed on (i, j) such that `spaghetti[i,j]` is a $n \times 2$ Numpy array equal to the `data` attribute of a `SpaghettiData` object whose `latitude` and `longitude` attributes match the `latitude[i,j]` and `longitude[i,j]` values of the `SpaghettiPlot` object. Please note that the `resolution` attributes of the `SpaghettiPlot` and the `SpaghettiData` objects must have the same value.

To initialise a `SpaghettiPlot` object it is sufficient to provide the geographical information of the target area:

```
>>> splot=SpaghettiPlot(min_lat,max_lat,min_lon,max_lon,resolution)
```

Initially all the entries in the spaghetti dictionary are empty arrays; they can be populated with the method

```
>>> splot.add_plot_data(spdata)
```

where spdata is a SpaghettiData object with the same resolution of splot and such that its latitude and longitude exist in the grid of splot.

The remaining two methods of a SpaghettiPlot object are used to plot its contents. The plot method produces the actual plot, as seen in the right side of Figure 1. It has two optional keyword arguments:

- `time_range=[time_min,time_max]` sets the temporal range of the plot (where both `time_min` and `time_max` are `datetime.datetime` objects);
- `temperature_range=[sst_min,sst_max]` sets the range of the SST values to be displayed in the plot (where both `sst_min` and `sst_max` are floats representing the SST expressed in degrees Celsius).

The `plot_reference_grid` method produces the reference grid of the plot, as seen in the left side of Figure 1. An optional keyword argument can be used in this method, `geomap=[mlat,Mlat,mlon,Mlon]`, which, if set, produces the plot of the reference grid inside a geographical map whose boundaries are defined by the values of `mlat`, `Mlat`, `mlon` and `Mlon` (see Figure 2).

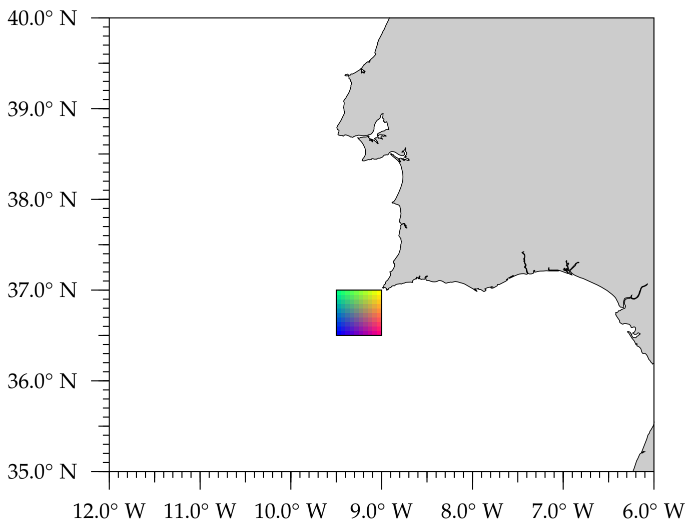

Figure 2: Plot of the reference grid of Figure 1 obtained from `plot_reference_grid` with the key `geomap=[35,40,-12,-6]`.

2 Creating a `SpaghettiData` dictionary

Since the information about SST has to be extracted from a series of NetCDF files (see [1] for more details), and that process may take some time depending on the number of files, the length of the time period, and the resolution chosen, we developed a small utility that reads the NetCDF files and stores the information in a suitable Python dictionary, which can be used in subsequent analyses and also saved to a file (as a pickled Python object) if needed at a later time.

Like a `SpaghettiPlot` object, the dictionary requires a target area in terms of maximal and minimal latitude and longitude, as well as a resolution: it has an entry for each point of geographical coordinates (ℓ, m) with ℓ and m as in Equation (1.1). The value of the entry is a `SpaghettiData` object with `latitude` $= \ell$, `longitude` $= m$ and the provided resolution; however, using the pair of *floats* (ℓ, m) as the corresponding key gives inconsistent and unreliable results, so we used as key the pair of (formatted) *strings*

$$(\texttt{f"\{}\ell\texttt{:.}x\texttt{f\}"}, \texttt{f"\{}m\texttt{:.}x\texttt{f\}"})$$

where x is the maximum number of decimal places between the representations of `min_lat`, `max_lat`, `min_lon`, `max_lon` and `resolution` (see the arguments below).

The utility is called by the command

```
>>> spdata_dict=create_spaghetti_data(<arguments>)
```

where `<arguments>` are the following:[2]

- `filedirs`: list of strings, each representing the path of a directory containing the NetCDF files with the SST information;
- `start_time`, `end_time`: `datetime.datetime` objects representing the beginning and the end of the considered time period;
- `min_lat`, `max_lat`, `min_lon`, `max_lon`, `resolution`: floats representing the spatial area covered by the dictionary, as defined above;
- `annual_trend=None`: if not `None`, this is a triple of floats (A, φ, μ) representing the coefficients of the annual trend function

  $$T(t) = A \cdot \sin(\omega t + \varphi) + \mu$$

  where t is the time expressed in days and $\omega = 365.256363004$ is the duration of a sidereal year; if this parameter is set, the annual trend of the SST (which is assumed to be sinusoidal) is taken into account when computing the SST

[2] We list positional arguments first, then keyword arguments—the latter are identified by the fact that they are followed by the "=" symbol and their default value.

values—in particular, if $\mathrm{SST}(t)$ is the temperature registered at a time t, then the `SpaghettiData` contains the pairs $(t, \mathrm{SST}'(t))$ instead of $(t, \mathrm{SST}(t))$, with

$$\mathrm{SST}'(t) = \mathrm{SST}(t) - \big(T(t) - T(t_0)\big)$$

where t_0 is the earliest time contained in the `SpaghettiData`;

- `lower_weight=None`: if not `None`, this is a float representing the weight to be given to lower quality data in the computation of the SST for a given square of the grid (see [1, Appendix A] for a brief description of the quality levels);
- `discard_threshold=None`: if not `None`, this is a pair (q, N), where q is a float and N is an integer, representing the minimum quantity of data that a NetCDF file should have in a square of the grid in order to not be discarded—in fact, we found that files with too few detected SST values usually produce plots with a lot of noise, preventing us from performing a proper analysis of the data; in particular we expect our NetCDF files to contain about one temperature value every $0.01°$ in latitude/longitude, so about $(100r)^2$ values in a square with resolution r: if `discard_threshold` $= (q, N)$, then a NetCDF file is discarded for a point (ℓ, m) of the grid if it contains less than $\max\{q(100r)^2, N\}$ values in the square $(\ell; \ell + r) \times (m; m + r)$; if `discard_threshold` is `None`, it defaults to $q = 0.0$ and $N = 1$, i.e. a file is discarded only if it has no data in the square;
- `save_data=False`: if `True`, the method produces in the current directory a pickled Python object called `SpaghettiData_YYYYmmdd_HHMMSS.pickle` (with `YYYY`, `mm`, `dd`, `HH`, `MM`, `SS` being the current year, month, day, hour, minute and second) containing the dictionary, together with a file called `SpaghettiData_YYYYmmdd_HHMMSS.txt` describing the values of the parameters used to produce it.

3 Creating a `SpaghettiPlot` object

A `SpaghettiPlot` object can be initialised directly and then populated with data from several `SpaghettiData` objects, as we described in Subsection 1.2; however it is more convenient to have a dedicated utility that does the work for us.

We wrote a function that returns a `SpaghettiPlot` object already populated with data that come either from a series of NetCDF files, or from a `SpaghettiData` dictionary produced as in Section 2—in fact, in the former case this function actually creates one by calling `create_spaghetti_data`.

This utility is called by the command

```
>>> splot=create_spaghetti_plot(<arguments>)
```

where `<arguments>` are the same as `create_spaghetti_data` together with one additional keyword argument, `load_data`, whose value can be either `None` (default) or a string representing the path of a pickled Python object containing a dictionary of `SpaghettiData`, as produced by the utility `create_spaghetti_data` called with `save_data=True`.

The behaviour of `create_spaghetti_plot` depends on the value of `load_data`:

- if `load_data` is `None`, then all the other parameters are passed to the function `create_spaghetti_data` and the resulting `SpaghettiData` dictionary (which may be saved with `save_data=True`) is used to populate a `SpaghettiPlot` object;
- if `load_data` is not `None`, then the corresponding `SpaghettiData` dictionary is loaded; in this case
  - the values of `file_dirs`, `annual_trend`, `lower_weight`, `discard_threshold`, and `save_data` are ignored;
  - the values of `min_lat`, `max_lat`, `min_lon`, `max_lon`, and `resolution` are used to define the target area of the `SpaghettiPlot`;[3]
  - the values of `start_time` and `end_time` are used to define the time range of the `SpaghettiPlot` object, so that SST values in the `SpaghettiData` dictionary whose time is outside this interval are not considered.

4 Future development

A `SpaghettiPlot` object is useful to visualise SST trends in a target area; however, depending on the dimensions and resolution of that area, the resulting plot can be hard to read. From our experience, it is very difficult to identify meaningful patterns when more than 100–120 plots are superimposed in a spaghetti plot.

On the other hand, a `SpaghettiData` dictionary can be a powerful instrument for the analysis of SST trends, eventually culminating in a pattern recognition algorithm for mesoscale events in an upwelling ecosystem. At the moment we are developing scripts that extract and process *statistics* (e.g. mean, standard deviation) for the SST from a `SpaghettiData` dictionary, in an attempt to recognise peculiar behaviours and associate them with specific events.

References

[1] Oscar Papini. *A tool for the temporal analysis of sea surface temperature maps*. ISTI Technical Reports 2021/011. ISTI-CNR, 2021. DOI: `10.32079/ISTI-TR-2021/011`.

[3]Notice that this function *does not check* consistency between the values of these parameters and the corresponding properties of the `SpaghettiData` dictionary—in particular, if the two values of the resolution disagree, the resulting plots may be inaccurate.

[2] Oscar Papini, Marco Reggiannini and Gabriele Pieri. 'SST Image Processing for Mesoscale Patterns Identification'. In: *Engineering Proceedings* 8, 5 (2021). Presented at the 16th International Workshop on Advanced Infrared Technology and Applications — AITA. DOI: `10.3390/engproc2021008005`.

[3] Marco Reggiannini et al. 'Mesoscale Patterns Identification Through SST Image Processing'. In: *Proceedings of the 2nd International Conference on Robotics, Computer Vision and Intelligent Systems — ROBOVIS*. SciTePress, 2021, pp. 165–172. DOI: `10.5220/0010714600003061`.