Spatial interpolators in Observable Plot

Philippe Rivière [1], Mike Bostock [2]

Abstract – We describe the spatial interpolators implemented in Observable Plot, that allow the user to create raster images and vector topographic contours from a set of spatially located sample values.

1. Observable Plot

Observable Plot is a free, open-source, JavaScript library for visualizing tabular data, focused on accelerating exploratory data analysis. It has a concise, memorable, yet expressive API, featuring scales and layered marks in the grammar of graphics style [1].

2. Spatial Interpolation

Given a set of samples with known locations on a plane and values (such as the measured temperature in the given location), we call *interpolation* any technique that estimates the value in any other location. Such a function may follow one or several additional requirements, such as continuity, differentiability, exactness (*i.e.* does the interpolator return the true value when evaluated on a sample location), etc.

The first law of geography, as Waldo Tobler famously put it, is that "everything is related to everything else, but near things are more related than distant things [2]." According to this law, we call *spatial interpolation* any such function where nearer data points have more influence on the interpolated value than farther ones.

Many such algorithms have been designed for quantitative data: Inverse Distance Weighted (IDW), Kernel Density Estimation (KDE), Kriging, Mean value coordinates, Natural neighbor interpolation, Bicubic interpolation, Stewart potentials, Thin plate spline…

In the following we describe Observable Plot's four built-in spatial interpolators.

Nearest. One obvious candidate is the *nearest sample* interpolation. For any (unknown) location, estimate its value as being equal to the value of the nearest sample (with some aggregation rule for samples taken at the same location). This results in Voronoi cells around each sample, with a uniform value equal to the sample value. This algorithm is exact, but not continuous (Fig. 1).

Barycentric. Constructs a Delaunay triangulation of the samples, and then for each pixel in the raster grid,

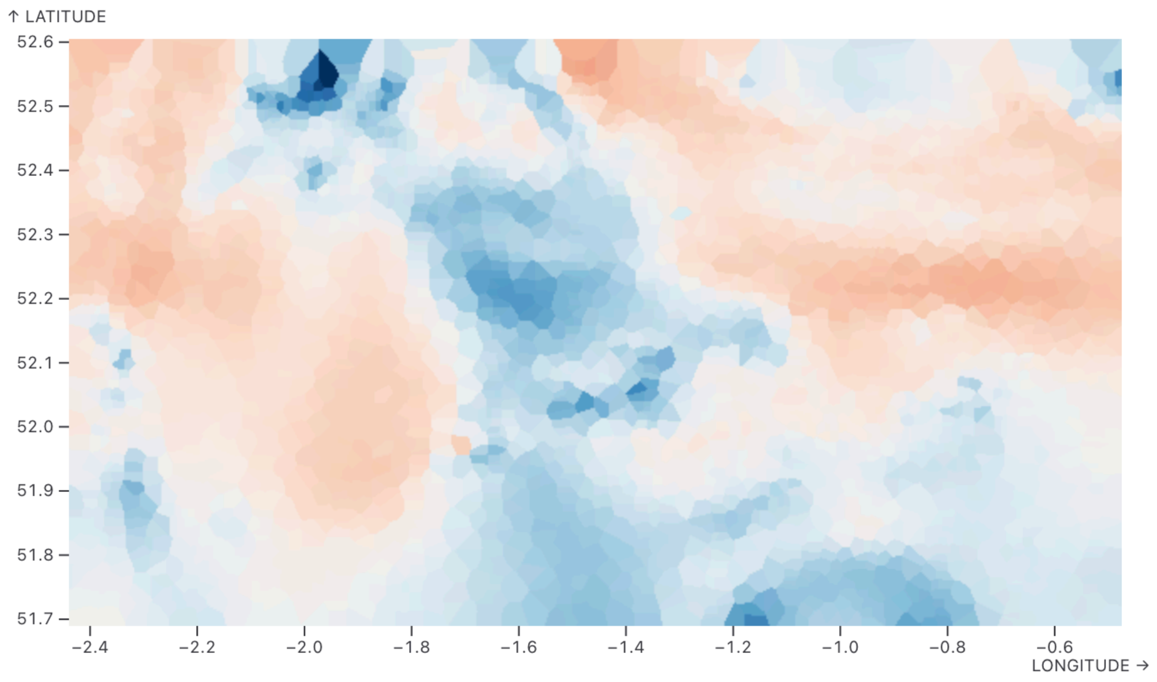

Figure 1. In 1955 the Great Britain aeromagnetic survey measured the Earth's magnetic field by plane. Each sample recorded the longitude and latitude alongside the strength of the international geomagnetic reference field (IGRF). Interpolated with *nearest.*

1. Visionscarto.net. 2. Observable, Inc.
Please send communications to: philippe.riviere@visionscarto.net

determines the triangle that covers it and mixes the values associated with the triangle's vertices using weighted barycentric coordinates (1 at any corner). Exact and continuous.

Random-walk. For each pixel in the raster grid, initiates a random walk, stopping when either the walk is within a given distance (minDistance) of a sample or the maximum allowable number of steps (maxSteps) have been taken, and then assigning the current pixel the closest sample's value. The random walk uses the "walk on spheres" algorithm in two dimensions described by Sawhney and Crane [3]. It is exact, but not continuous (though blurring can help make it continuous).

None. For the sake of completeness, let's also mention the *none* interpolator: project any given sample to the grid, and return *unknown* for points on the grid that have no associated sample.

3. Quantitative vs. Categorical

For the *nearest* and *random-walk* methods, the selected sample value is returned as is, which makes no difference whether categorical or quantitative. For the *barycentric* method, though, we need to define how mixing works. On quantitative values, the mixing takes a (linearly) weighted average of the values. For categorical values, one of the 3 selected samples is picked at random, with a probability distribution equal to the barycentric weights. This allows to create a map, say, of the expected species of an iris given its petal and sepal lengths, where the resulting color texture represents a spatially variable probability distribution. The image generated (fig. 2) can help build an intuition for a machine-learning model that would characterize the data in a more statistically controlled way.

4. Contours

With quantitative values, the generated rectangular matrix of interpolated values can serve as a base for the traditional marching squares algorithm, to derive contours of equal value (isolines, topographic contours). This in fact generalizes the classic d3-contour module [4] to non-gridded data. In that case any blurring —for smoother contours— happens on unscaled values.

5. Projections & Maps

Plot's spatial interpolators operate on projected coordinates in pixel space—not on the original data dimensions. As a consequence, they are compatible with

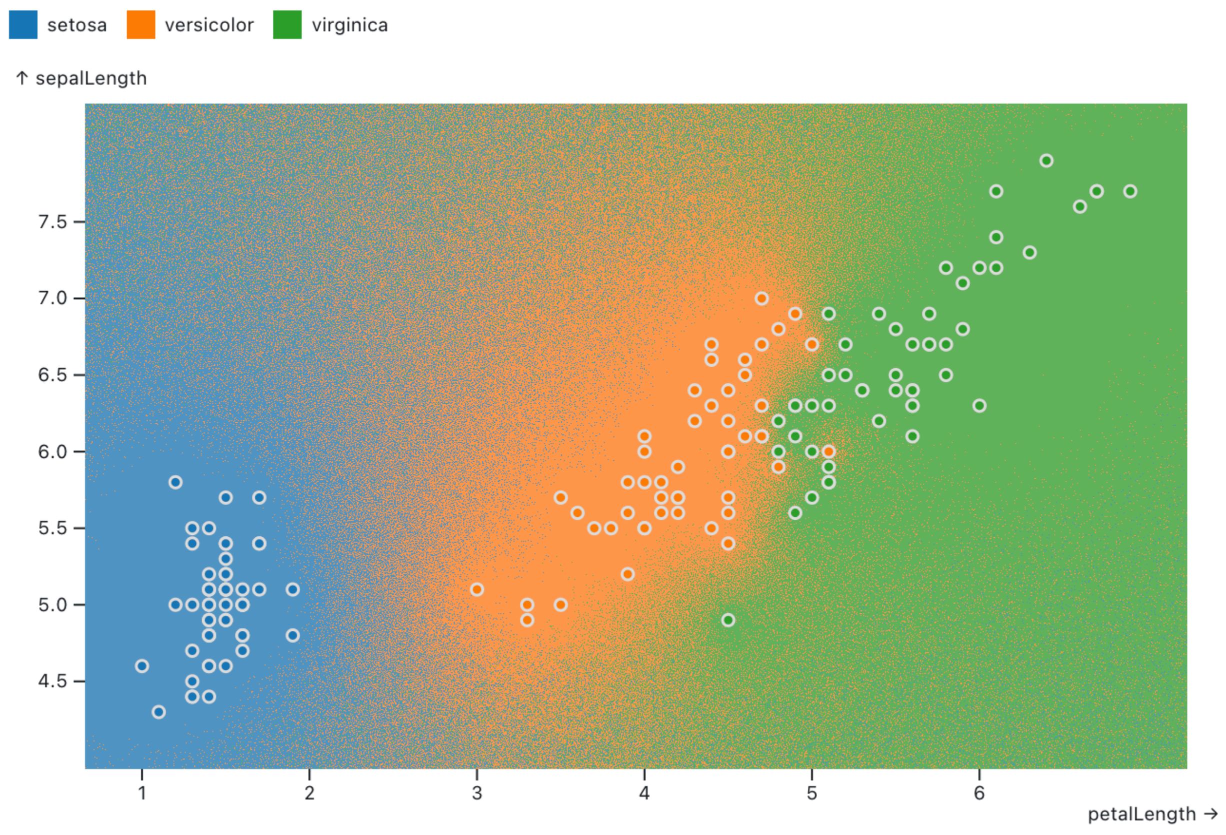

Figure 2. A scatterplot of Iris species. The texture, generated with the *random-walk* interpolator, reflects the growing uncertainty as we are farther from any sample location.

Plot's projection system, and can be used to derive world maps of quantitative or categorical data (fig. 3).

6. Extensibility

These methods are vanilla JavaScript functions that read in three channels (Value, and projected coordinates X and Y), a width and a height, and return as output a rectangular array of width × height interpolated values. They can be used outside of a *plot*, and indeed outside of Observable Plot, as generic transformations for custom data visualization or for statistical analysis. Conversely, Plot supports spatial interpolators specified as an arbitrary function, which allows anyone to implement any other spatial interpolation algorithm. For example, we are experimenting with a substitution of a blue noise function for pseudo-random generator, which results in smoother (and more statistically accurate) textures for the random-walk method. Plot's API makes it relatively easy to plug in a new implementation of an old algorithm, or your newly-designed algorithm.

7. Future Work & Suggestions

Implementing some of the classical algorithms described in section 3 would provide a wider range of options to interpolate quantitative data. New options to existing interpolators —such as a cutoff distance for the *random-walk*— could be provided to fit users' needs. Other types of spatially-defined functions, beyond interpolators, could also be implemented within the same API: a density estimator, a distance field, a Poisson potential flow…

8. Conclusion

Observable Plot's spatial interpolators are a recent contribution to the domain of data visualization and map making. They generalize tools such as d3-contour, by making them work on non-gridded data samples. Beyond data visualization, the *random-walk* interpolator can prove interesting to, *e.g.*, teach machine learning. The system is open-source and offers a public API that allows anyone to add their own implementation of any arbitrary spatial algorithm.

9. References

[1] https://observablehq.com/plot/

[2] Waldo Tobler, "A computer movie simulating urban growth in the Detroit region", *Economic Geography*, 46(Supplement): 234–240, 1970.

[3] Rohan Sawhney and Keenan Crane, "Monte Carlo Geometry Processing: A Grid-Free Approach to PDE-Based Methods on Volumetric Domains", SIGGRAPH 2020 / ACM Transactions on Graphics 2020.

[4] Michael Bostock, Vadim Ogievetsky and Jeffrey Heer, "D3: Data-Driven Documents", IEEE Transactions on Visualization and Computer Graphics, Vol. 17, Issue 12, 2011.

15 JUNE 2023.

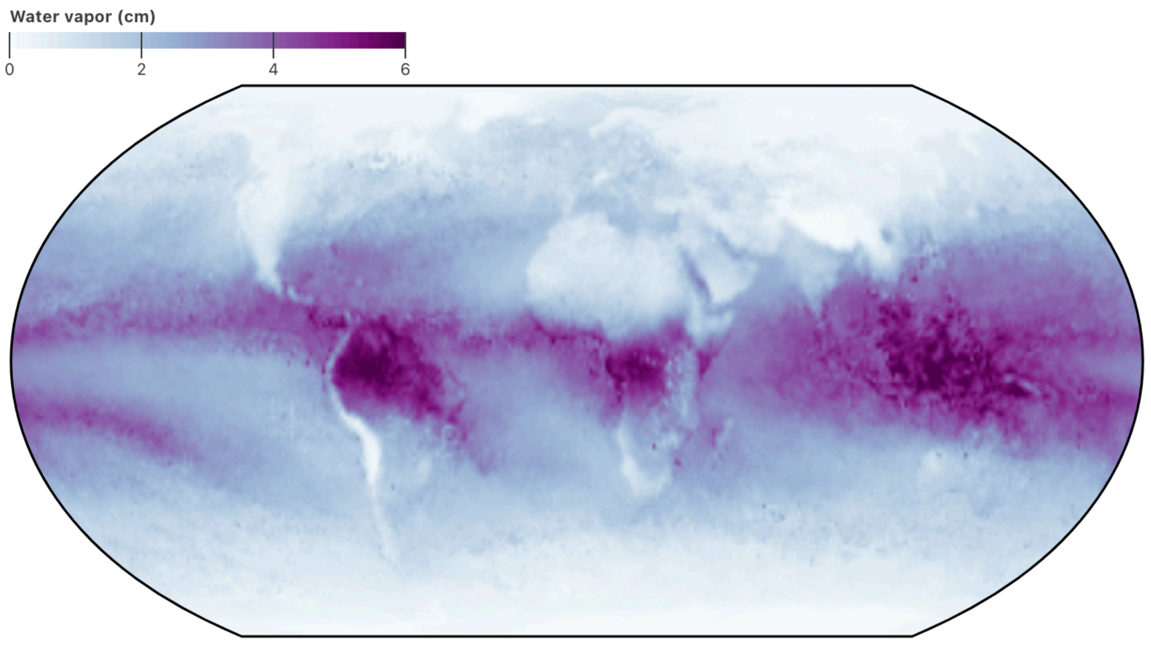

Figure 3. A map of global atmospheric water vapor measurements from NASA Earth Observations, projected with the Equal Earth projection.