FAST CALCULATION OF RIPLEY'S K AND L FUNCTIONS ON A SPHERE.

Tobias Ståhl
School of Natural Sciences, University of Tasmania
Hobart, TAS 7005 Australia
tobias.staal@utas.edu.au

July 20, 2022

ABSTRACT

In this brief paper, I show how Ripley's K functions can be calculated efficiently for the surface of a sphere using a distance matrix. The immediate use is to analyse global point distributions on Earth; however, the approach is expandable for more complex surfaces.

***K*eywords** Ripley's K · Spatial descriptive statistics · Point pattern

1 Introduction

Ripley's K-function reveals clustering and regularisation in point distributions. Further, the K-function can be centred or compensated for variance (L-function). The metrics have applications in a large range of analysis [1, 2]. The framework for general spaces is given by Ripley (1977) [3], and for space-sphere point processes by Møller et al (2019)[4]. Integrating distance and area over non-Euclidean surfaces can be computationally expensive. In this brief contribution, I provide an applied method for calculating the K functions for points on the surface of a sphere using a distance matrix. The approach and code were developed to analyse the global distribution of geophysical measurements [5].

The empirical K-function, without edge correction, is given as [2]:

$$\widehat{K}(t) = \frac{|W|}{n(n-1)} \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \mathbf{I}\{d_{ij} \leq t\}, \tag{1}$$

where $|W|$ is the area of the observation window, n is the number of points, and $n-1$ is hence the maximum number of neighbours from each point. d_{ij} are the pairwise distances between points. t is the distance. I is the indicator function, returning 1 if true. The function is typically calculated for multiple distances, t, to investigate at what scale clustering or inhibition occurs.

On a sphere, the distance between points, d, can be calculated with the haversine formula; similarly, the area of the observation window, W, might be estimated as a spherical cap for a spherical planet. However, refined distance functions or numerical calculations from triangular irregular networks are required in some situations. Such calculation of arc distances and area integration can be computationally costly.

Here, I propose an efficient method for such calculations:

1. Organise all coordinate pairs between points in an array, only include each pair once and exclude the distance to each point itself.
2. Define the t-values to include in the K-function, and store them to array T.
3. Calculate the distances for all pairs, stored in a sorted 1-D array D. This process can be vectorized, and potentially parallelized.

4. For each distance t in T; count how many elements in $D < t$. This is effectively the indicator function, I. This process can be vectorized.
5. Calculate $\widehat{K}(T)$, centred $\widehat{K}(T)$, and $\widehat{L}(T)$.

2 Application

Calculate the distances for all pairs. Using numpy [6] this can be done as:

```
def make_dist_list(lats, lons):
    '''
    lats and lons are numpy arrays with coordinates
    returns : 1D array with all distances
    '''

    assert np.shape(lats) == np.shape(lons), 'lons and lats must have same shape'
    n = np.shape(lats)[0]

    lats = np.meshgrid(lats, lats)
    lons = np.meshgrid(lons, lons)

    # mask the triangular matrix, k = 1 excludes the diagonal
    mask = np.tril(np.ones((n,n)), k=-1).astype('bool')

    return distance(lats[0][mask].flatten(), lons[0][mask].flatten(),
                lats[1][mask].flatten(), lons[1][mask].flatten())
```

Define the t-values. With the here proposed method, it is possible to efficiently calculate the function for a large number of distances:

```
T = np.linspace(0, 180, 1001) * 111.1111111 # Make kilometres from degrees, if on Earth.
```

The observational area, $|W|$, is a cap for the case of sphere in km. Note that the distances are converted back to degrees from km:

```
def obs_area(t, radius=6_371, t_factor = 111.1111111):
    '''
    Spherical cap surface area
    t = arc distance
    radius of sphere
    '''
    return 2* np.pi * radius**2 * (1 - np.cos(np.deg2rad(t)/t_factor))
```

Finally, the functions and reference area is calculated:

```
dists = make_dist_list(lats, lons)
K = A /(n*n) * 2 * np.sum(dists[:, None] < T, axis=0)
R = obs_area(T)
```

Reshaping the arrays allows for vectorization.

Some examples of centered K functions are shown in Fig. 1. Note in particular the final example, where samples are drawn from a uniform distribution of parallels, failing to consider that meridians converge towards the poles.

With this paper, I only briefly introduce the methods. Ongoing work aim to integrate second-order analysis into existing frameworks for geophysocal data processing.

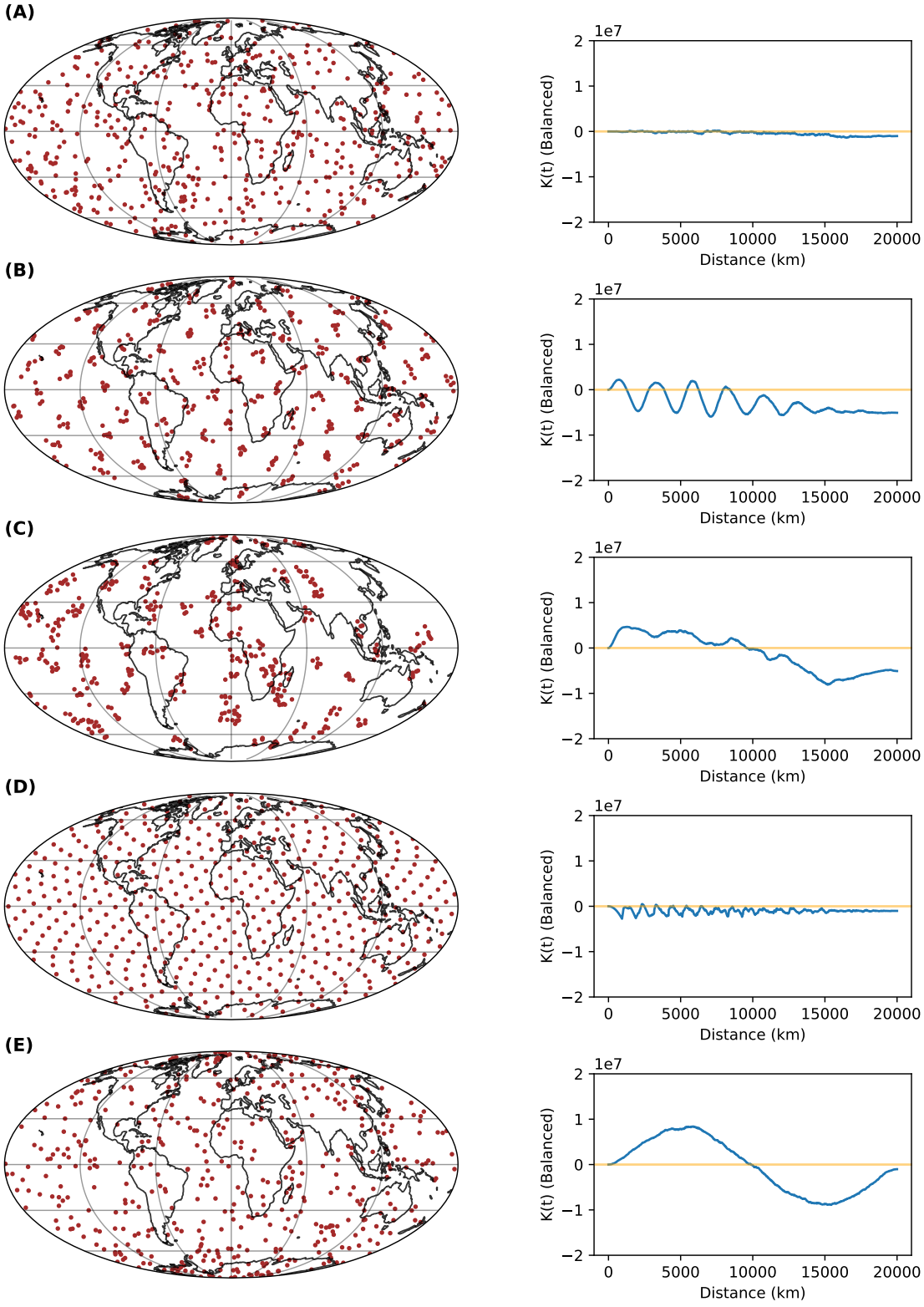

Figure 1: Five examples for global point data. Calculated for 250 points; (a) A random distribution, shown as a centered K function for all distances. (b) Clusters at regular distances, shown as clustering and inhibitions. (c) Clusters at random distances, shown as clustering at short distances and inhibition at longer distances (d) Fibonacci lattice, with similar separation between points, and inhibition at all scales. (e) Random sampled lat lon, this example shows the clustering of points selected from a uniform distribution sof -90 to 90 latetude, and -180 to 180 longitude. Due to the convergence of the meridioans near the poles, the distribution is clustered up to a hemisphere area, and inhibited on a global scale.

Additional figures and Python code is available at `https://github.com/TobbeTripitaka/ripley`.

References

[1] Philip M. Dixon. Ripley's K Function. *Wiley StatsRef: Statistics Reference Online*, 3:1796–1803, 2014.

[2] Adrian Baddeley, Ege Rubak, and Rolf Turner. *Spatial Point Patterns*. CRC press, 2015.

[3] B. D. Ripley. Modelling Spatial Patterns. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(2):172–192, 1977.

[4] Jesper Møller, Heidi S Christensen, Francisco Cuevas-Pacheco, and Andreas D Christoffersen. Structured Space-Sphere Point Processes and K-Functions. *Methodology and Computing in Applied Probability*, 23(2):569–591, jun 2021.

[5] Tobias Stål, Anya M. Reading, Sven Fuchs, Jacqueline A. Halpin, Mareen Lösing, and Ross J. Turner. Understanding Sampling Bias in the Global Heat Flow Compilation. *Frontiers in Earth Science, in review*, 2022.

[6] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020.