fieldmodel

Kristian M. Eschenburg

April 10, 2020

Contents

1 Introduction

1.1 Fundamentals

fieldmodel is a Python package for fitting densities to scalar fields. The package offers two flavors of fitting: on manifolds and on regular 3D grids. Given a scalar field assigning, let's say, elevation values to latitude-longitude coordinates centered around Mount Rainier, we could use **fieldmodel** to fit a Gaussian distribution to this elevation scalar map. The output of this fitting procedure would be 2 parameters: a `mean` parameter (let's say centered at the true summit latitude-longitude coordinates of Mount Rainier), and a `sigma` parameter, analogous to the standard deviation parameter from a univariate Gaussian, that is indicative of how quickly the elevation from the summit decays.

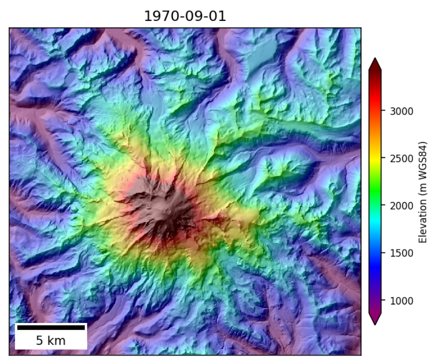

Figure 1: Rainier elevation as a function of decade.

Along with the fitting procedure, **fieldmodel** also offers some basic plotting functionality to visualize the fitted densities along with the data to which those densities were fit.

1.2 Software Requirements

In order to run the following analysis, we need to download some Python code and install these on our servers. Please review the `requirements.txt` file in the base directory. If you've installed your Python distribution using Anaconda, these packages should already be installed. In order to install **fieldmodel**, enter the following in to the Terminal:

```
git clone https://github.com/kristianeschenburg/fieldmodel
cd ./fieldmodel/
pip install -e .
```

This will install the package, and will enable you to import the modules.

2 GeodesicFieldModel

The primary class of interest (in our case) is **GeodesicFieldModel – GFM**) for short. We can invoke the **GFM** class as follows:

```
import numpy as np
from fieldmodel import GeodesicFieldModel as GFM

"""
The GFM.FieldModel is instantiated with 6 parameters:
r: int
  Initial sigma estimate
amplitude: bool
  Whether to include amplitude in fitting procedure
peak_size: int
  Minimum distance between local minima.  If two point are within peak_size distance of one another,
  the local minima which the smaller signal is discarded
hood_size: int
  Size of search space around each local maximum. hood_size defines the radius of a circle around the local
  maxima in which to search for the optimal mean location.
metric: string
  Cost function to use to optimization procedure.  Options include: 'pearson', 'kendall', 'spearman', 'L2', and 'L1'
"""

G = GFM.FieldModel(r=10, amplitude=False, peak_size=15,
        hood_size=20,  verbose=False, metric='pearson')
```

In order to fit the actual model, we need to provide the model with some arguments.

```
import numpy as np
from fieldmodel import GeodesicFieldModel as GFM

"""
We use the GFM.FieldModel.fit() method to fit our model, which takes 4 arguments:

distance: int, array
  Symmetric pairwise distance matrix for all data points in the domain
data: float, array
  Scalar field to which the fieldmodel will be fit.  Associates each datapoint with a numerical value.
x / y: float, array
  Optional. (x,y) coordinates of data points.  Used for plotting model results.
"""

# Create fake distance matrix between and fake scalar map
D = np.random.rand(10, 10)
D = np.abs(D + D.T)
scalar_map = np.random.rand(10)

# Crate fake x / y coordiantes
[x,y] = np.random.rand(10, 2)

G.fit(distance=D, data=scalar_map, x=x, y=y)
```