Project Kernel Density Estimation: [ need title]

**Motivation for and Objective of Project**

The objective of this project is to create a top level module to centralize the implementation of kernel density estimation (KDE) methods within the PySal library of spatial analysis tools. Currently, kernel density estimation methods are implemented in a variety ways, both internally to PySal and additionally within other software programs associated with the The GeoDa Center for Geospatial Analysis and Computation, such as STARS. Centralizing KDE methods into one module will facilitate its implementation and is a practical objective, given that KDE is utilized for many purposes, such as exploratory point data analysis, point data smoothing and hot spot detection [de Smith, Goodchild, Longley, 2006], within a wide range of disciplines, including archeology, banking, climatology, hydrology, economics, genetics and physiology (Sheather, 2004).

Kernel density estimation is a way to estimate the density of data points using kernel methods (Sheather, 2004), and can be thought of as a smoothly curved surface that is fitted over each point in a data set. This curved surface is an estimation of the distribution of the data as a probability density function, and is non-parametrically derived from the data. ‘Kernel’ simply refers to the form of probability estimation function used to perform the estimation. The density at each data point is the sum of these estimated values derived from the kernel function at each point.

Kernel density estimation is more easily understood if related to a histogram, perhaps a more familiar method of data classification. In a histogram, data points are placed into discrete bins. However, such discrete placement of the data can mask some of its characteristics due to overgeneralization.

In contrast to histogram classification, kernel estimation reveals smoother transitions between data points. In kernel estimation, each data point is treated as if spread over a range, or neighborhood, where it is the center of its own neighborhood. These neighborhoods can be thought of as being analogous to histogram bins. The kernel estimator counts and sums the number of data points found within all the overlapping neighborhood layers to arrive at a cumulative probability estimation, where more area under the curve corresponds to higher probabilities, similar to the way height in a histogram bin corresponds to more values within the bin. The difference here is that in effect, each data point is weighted by how far it is from the center of all the other overlapping neighborhoods and so are not restricted by the discrete values found at the bounded edges of a histogram bin. Values are highest where neighborhoods overlap the most but smoothly diminish at points where there is less overlap. These cumulative sums as estimated by the kernel are then rescaled by the range used to construct the neighborhoods so that the area under all curves sums to one, to assure standardized view of the relationships between curves. [sensu de Smith, Goodchild, Longley, 2006; figure out how to cite this http://www.math.caltech.edu/~alberts/talks/KernelEstimation.pdf] The result is an interpolation over distances between points, yielding something similar to a risk surface.

**Considerations for implementation**

General form for kernel density estimation is:

\begin{displaymath}
\hat{{f}}(x)=\frac{1}{n}\sum_{i=1}^{n}K\left(\frac{x-x(i)}{h}\right)\end{displaymath}

K in the equation refers to the form of the kernel, or form of probability function. Kernels used are Gaussian [or the standard normal distribution], Quartic, Exponential, Triangular, Uniform and Epanechnikov . [LIST FXNS HERE OR SOMEWHERE NEAR?]. The shape of these functions determines, in part, the contribution each point will make to the total probability estimation. This contribution is also determined by the numerator, whereby x – x(i) defines the range, or spread of influence of the data point x(i). In the denominator, the value for h, or bandwidth, is used to modify the range for x(i).

The kernel estimate depends less on the shape chosen for K than on the value of its bandwidth h (de Smith, Goodchild, Longley , 2006; Sheather, 2004 ). Larger values for h decrease the difference between x and x(i), smoothing more because kernel will be estimated over smaller distances. Large values for h over-smooth the estimation, while too small of values for h under-smooth.

**Implementation in code:**

2) Kernel Equations

Explanation of xi, mu, sig

Gaussian, Triangular, Uniform - we worked on Gaussian bc is most often used—put in docstring which ones currently available, or in other documentation

3) Code Workflow

Flowchart

Functions:

Preparing Lists

Calculation Function

Output Grid

Output Point

**Future work:**

This module represents the first step in centralizing kernel density estimation calculations. Gaussian, triangular, and uniform kernel functions are available to the user at the moment, but Epanichekov, Quadratic, and Quartic are not. The blueprint is made for these to be implemented, though, and extra conditional statements in the current ‘calculate()’ function, coupled with similar functions to the ‘Gaussian()’, and ‘Gaussian\_Point()’, etc. functions, will allow these to assimilate into the existing code with ease.

In its current form, the code allows the user to view kernel output values at a grid defined by the maximum and minimum x,y values (with bandwith buffer added), divided into equally sized grid cells by a resolution value. In addition, a user can query a specific location’s kernel height. Creating a variable-sized output matrix would be a useful addition, along with the ability to query multiple locations within the area of interest. Another useful variation on the output grid functionality is the matrix output form of [x1,y1, z, x2,y2,z….x2,y2,z]. We chose not to overwhelm the user with a matrix of this size in the case of a large coverage area for the sake of our project, but this could certainly be implemented. This would be particularly useful in the case of a raster dataset creation for geovisualization.

Though the kernel module is not designed for direct visualization use, it would be helpful to see these outputs in an applied geographically-relevant projects’ graphic interface. Finally, this module represents the knowledge of our group’s efforts to understand and automate kernel density estimation calculations. Code optimization by those more experienced in both Python and kernel density estimation calculations could improve the performance on this model. We tested this code on relatively small datasets. Its performance on a very large input dataset is yet to be determined.

|  |
| --- |
| Outputs need to be tested in some visualization application. |
| ? HTML link to documentation on choices of bandwidth, and kernel selection?  Gaussian is most often used but is most computationally intensive [i think because the edges are asymptotic? i.e. http://www.spatialanalysisonline.com/output/images/image410.gif where as triangular e.g. is bounded- http://www.spatialanalysisonline.com/output/images/image416.gif??? Not sure ] |

Computational optimization was outside the scope of experience of these collaborators.

**references**

|  |
| --- |
|  |

Geospatial Analysis - a comprehensive guide. 3rd edition © 2006-2011 de Smith, Goodchild, Longley <http://www.spatialanalysisonline.com/output/>

Density Estimation

Simon J. Sheather

Statistical Science , Vol. 19, No. 4 (Nov., 2004), pp. 588-597

Published by: [Institute of Mathematical Statistics](http://www.jstor.org.ezproxy1.lib.asu.edu/action/showPublisher?publisherCode=ims)

Article Stable URL: http://www.jstor.org/stable/4144429