LatticeKrig Vignette

Matthew Iverson

5/21/2019

Contents

1 Introduction

In this vignette, we will briefly explain what kriging is, explore the functions in the LatticeKrig package, and show examples of how they can be used to solve problems.

1.1 What is Kriging?

Kriging (named for South African satistician Danie Krige) is a method for making predictions from a data set. It is designed to be used on spatial data – that is, our data contains the observed variable and the location it was observed at, and pairs of observations taken close together have similar values. As such, it can be applied to a variety of physical data sets, from geological data to atmospheric data.

The goal of kriging is to create a model, based on data from some locations, that can predict the observed variable at any location.

1.2 Glossary of Important Functions

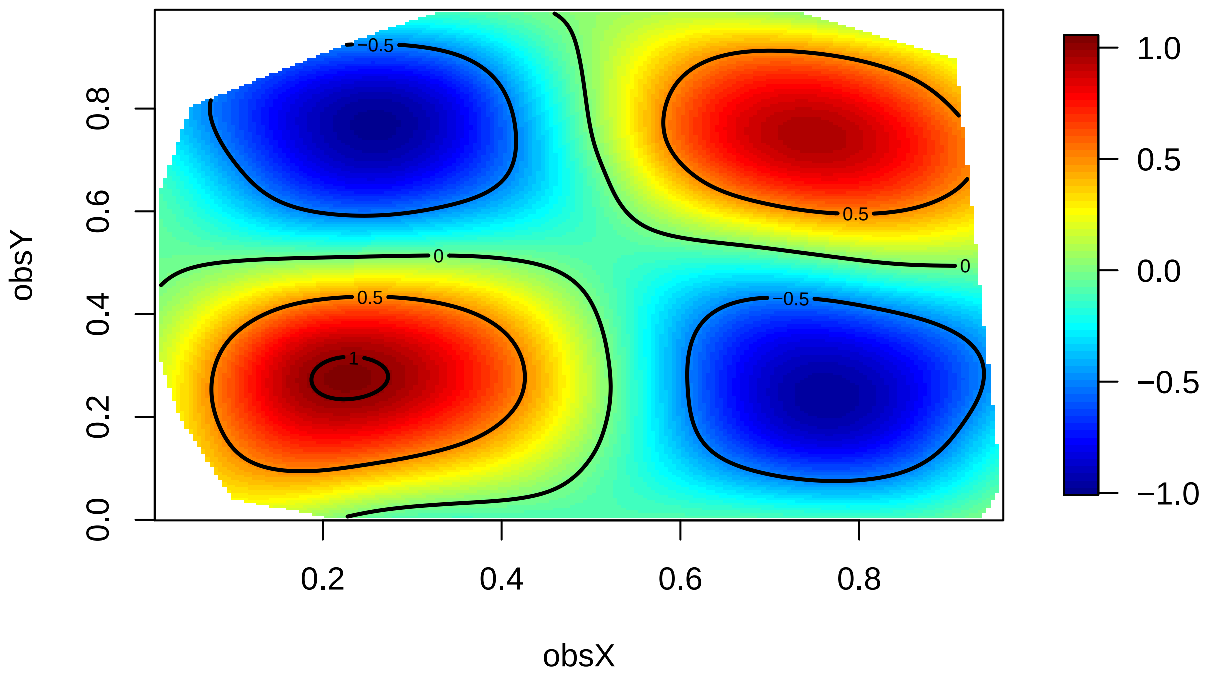
- `LKrig`: Fits a kriging estimate to the given data.
- `LatticeKrig`: Calls LKrig, passing in default values and estimates for the needed parameters.
- `surface`: Plots a fitted surface in 2D space as a color plot and adds contour lines.

- hi

2 The `LatticeKrig` Function

The `LatticeKrig` function takes in a set of locations and variable observations and returns a LKrig object that can be used to estimate the variable at other locations. It also has many optional parameters that can be used to adjust the model. In the following toy example, we take 100 samples of $\sin(2\pi x)\sin(2\pi y)$ at random locations on the unit square and add some noise onto the samples. We then use `LatticeKrig` to recover the shape of the function and predict it over the unit square. Notice that, by default, the `surface` function won't make predictions outside the boundaries of the original data, and so some parts near the corners are left blank.

```r
# set the seed of the random number generator so we always get the same output
set.seed(31734)
# make a group of 100 random (x,y) points in the unit square
obsX <- runif(100);
obsY <- runif(100);
obsXY <- cbind(obsX, obsY);
# evaluate sin(2 pi x) * sin(2 pi y) at each location, then add some noise
obsZ <- sin(2*pi*obsX) * sin(2*pi*obsY) + 0.1*rnorm(100);
# use kriging to find an approximate 2D-surface fit
kFit <- LatticeKrig(obsXY, obsZ)
# plot the kriging fit's predicted values on a 200x200 grid in the unit square
surface(kFit, nx=200, ny=200)
```



3 Appendix A: The Linear Algebra of Kriging

Suppose we have a vector$\mathbf{y}$ of observations, where each observation y_i is taken at location $\mathbf{s}_i$, and a covariate matrix X. Assuming that the observations are a linear combination of the covariates with a Gaussian process of mean 0, we have

$$\mathbf{y} = X\mathbf{d} + \epsilon$$

where $\epsilon \sim MN(\mathbf{0}, \Sigma)$ for some covariance matrix Σ. We can then make assumptions to determine the form of Σ: Assuming the process is stationary, σ_{ij} will only depend on the vector $\mathbf{s}_i - \mathbf{s}_j$; assuming the process is isotropic, σ_{ij} will only depend on the scalar $||\mathbf{s}_i - \mathbf{s}_j||$, which also means that Σ will be symmetric. This then allows us to establish a covariance function, c, such that $\sigma_{ij} = c(||\mathbf{s}_i - \mathbf{s}_j||)$. The covariance function describes how strongly correlated observations at varying distances are; as such, we would expect that $c(0) = 1$ and $\lim_{x\to\infty} c(x) = 0$. We can make further assumptions about the covariance function to make computations easier. In LatticeKrig, we assume the covariance function is a Wendland function, which has compact support on $[0, 1]$. This compact support will lead to a sparse Σ, which makes computing with Σ significantly faster and allows us to compute kriging estimates on very large data sets in a reasonable amount of time. Alternatively, in fixed-rank kriging, it is assumed that $\Sigma = S^TKS$, where each column of S is a set of basis functions evaluated at a location of an observation. This form of Σ also makes some computations easier, making it another technique for kriging on large data sets.

In LatticeKrig, we assume that $\epsilon = \Phi\mathbf{c} + \mathbf{e}$, where Φ is a matrix of radial basis functions (so ϕ_{ij} is the j^{th} basis function evaluated at the i^{th} point) and each basis function is the same except for a shift in location, $\mathbf{c}$ is the vector of coefficents that each basis function is weighted by, and $\mathbf{e}$ is the vector of measurement errors, distributed $N(0, \sigma^2 I)$. Thus, our total model is $\mathbf{y} = X\mathbf{d} + \Phi\mathbf{c} + \mathbf{e}$. We can't predict measurement error, so instead we focus on predicting $X\mathbf{d} + \Phi\mathbf{c}$ at new locations. The matrix of covariates X and the matrix of basis functions Φ are both determined from the points we choose to predict at: the unknowns we need to estimate are $\mathbf{c}$ and $\mathbf{d}$. We estimate $\mathbf{d}$ by using the generalized least squares estimate: $\mathbf{d} = (X^T\Sigma^{-1}X)^{-1}X^T\Sigma^{-1}$. Estimating $\mathbf{c}$ is more involved. First, we partition X and $\mathbf{y}$ into two parts: the parts corresponding to the known data, X_1 and $\mathbf{y}_1$, and the parts corresponding to the data we want to predict, X_2 and $\mathbf{y}_2$. Since we assume that y follows a Gaussian process, we can write

$$\begin{pmatrix}\mathbf{y}_1\\ \mathbf{y}_2\end{pmatrix} \sim N\left(\begin{pmatrix}\mu_1\\ \mu_2\end{pmatrix}, \begin{bmatrix}\Sigma_{11} & \Sigma_{12}\\ \Sigma_{21} & \Sigma_{22}\end{bmatrix}\right).$$

It is known from multivariate probability theory that

$$E[\mathbf{y}_2|\mathbf{y}_1] = \mu_2 + \Sigma_{21}\Sigma_{11}^{-1}(\mathbf{y}_1 - \mu_1).$$

Where μ_1 and μ_2 are the means of $\mathbf{y}_1$ and $\mathbf{y}_2$, respectively. Since $\epsilon = \Phi\mathbf{c} + \mathbf{e}$ has mean 0, the mean must come from the $X\mathbf{d}$ term: that is, $\mu_1 = X_1\mathbf{d}$ and $\mu_2 = X_2\mathbf{d}$. Since $E[\mathbf{y}_2|\mathbf{y}_1]$ is the best estimator of the values of $\mathbf{y}_2$, we want to find a value of $\mathbf{c}$ that makes our model reproduce this estimator, so we set $E[\mathbf{y}_2|\mathbf{y}_1] = X_2\mathbf{d} + \Phi_2\mathbf{c}$, where Φ_2 is the matrix of all basis functions evaluated at the points where we're trying to predict y. This gives us the equation

$$X_2\mathbf{d} + \Phi_2\mathbf{c} = X_2\mathbf{d} + \Sigma_{21}\Sigma_{11}^{-1}(\mathbf{y}_1 - \mu_1).$$

Now, consider what happens if we make the covariance function and basis function match. Each entry in Σ_{21} is the covariance function of the distance between the j^{th} data point and the i^{th} prediction point, which would be equal to the basis function of the distance between the j^{th} data point and the i^{th} prediction point, which is each entry in Φ_2. This means we can substitute $\Phi_2 = \Sigma_{21}$ into our equation, giving us:

$$\begin{aligned}
X_2\mathbf{d} + \Phi_2\mathbf{c} &= X_2\mathbf{d} + \Sigma_{21}\Sigma_{11}^{-1}(\mathbf{y}_1 - \mu_1)\\
\Phi_2\mathbf{c} &= \Sigma_{21}\Sigma_{11}^{-1}(\mathbf{y}_1 - \mu_1)\\
\Phi_2\mathbf{c} &= \Phi_2\Sigma_{11}^{-1}(\mathbf{y}_1 - \mu_1)\\
\mathbf{c} &= \Sigma_{11}^{-1}(\mathbf{y}_1 - \mu_1)
\end{aligned}$$

and so we arrive at a formula for $\mathbf{c}$ which, when the basis function equals the covariance function, gives us the best estimator for $\mathbf{y}_2$.