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 accurately predict the observed variable at any location inside the data given.

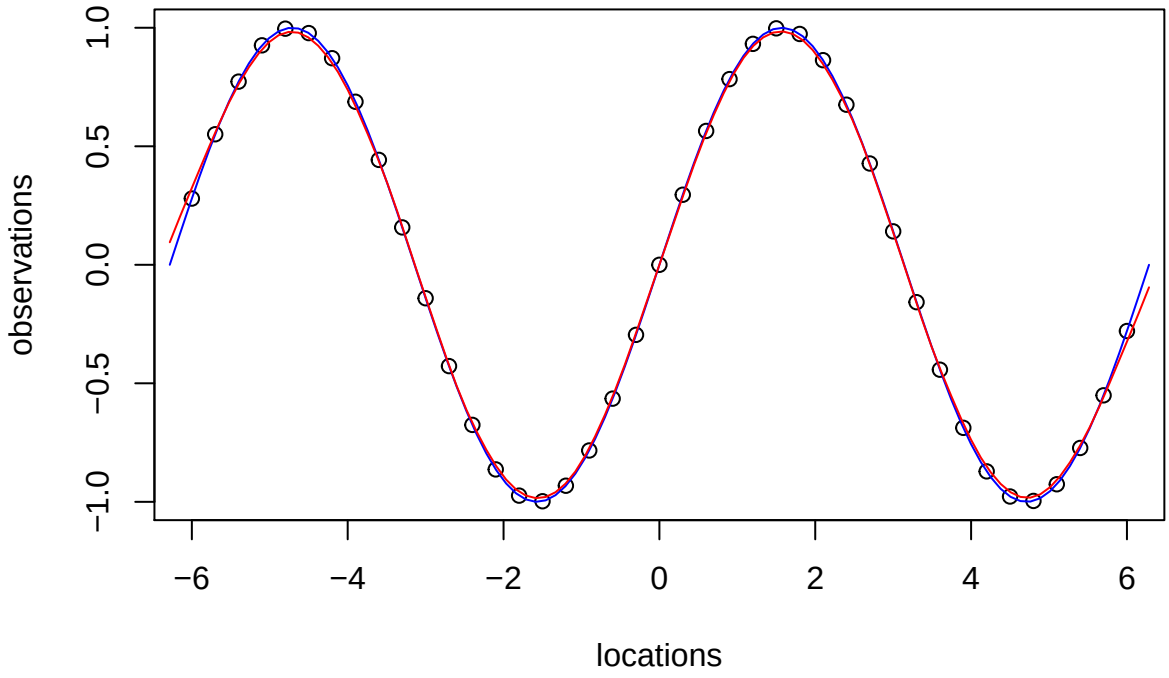
1.2 Glossary of Important Functions

- `LKrig`: Fits a kriging estimate to the given data.
- `LatticeKrig`: Calls LKrig, passing in default values and estimates for the needed parameters.
- `LKInfo`: Creates an object to store the parameters to use for a `LatticeKrig` / `LKrig` call; especially useful for examining the effect of changing one parameter on the fit.
- `surface`: Plots a fitted surface in 2D space as a color plot and adds contour lines.
- `image.plot`: Plots a dataset or fitted surface in 2D space as a color plot without contour lines.
- `predictSurface`: Computes the values from a Kriging fit and makes a surface, but doesn't plot it.

2 Quick Start Guide

In this section, we will lay out the bare essentials of the package to make the central features as easily accessible as possible. To fit a surface and interpolate data using `LatticeKrig`, the only required arguments are the measurement locations and measurement values. Calling the `LatticeKrig` function and passing in the locations and values will produce an `LKrig` object that contains all the information needed to calculate the model at any location. For a simple, 1-dimensional example, we will take our variable measurements to be the values of a simple function.

```
# We will make our measurements at integers from -6 to 6, inclusive
locations <- seq(-6,6,0.3)

# at each location, we measure the variable we're trying to make a model for
# for the sake of example, our variable will equal the sine of the location
observations <- sin(locations)

# use LatticeKrig to estimate a fit for these data points
# note that if there are any missing (NA) values, LatticeKrig removes them with a warning
kFit <- LatticeKrig(locations, observations)

#create a grid of 100 x-values from -2pi to 2pi to compare the fit and true function on
xGrid <- seq(-2*pi, 2*pi, len=100)

#draw the data points
plot(locations, observations)

#draw the true function for comparison
lines(xGrid, sin(xGrid), col='blue')

#draw the LatticeKrig estimate over the whole interval
lines(xGrid, predict(kFit, xGrid), col='red')
```

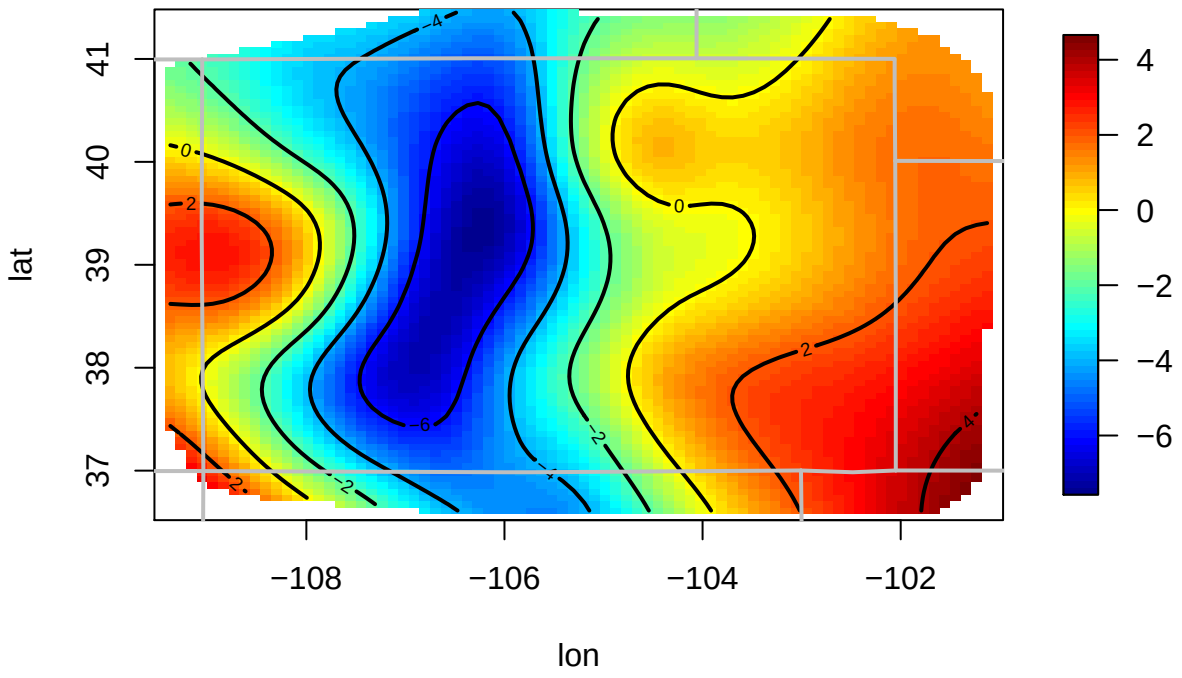


We can see that `LatticeKrig` takes in the data points (shown above in black) and produces a prediction over the whole interval (in red) that matches the true function (in blue) rather closely. For another, more practical example, we will predict the average spring temperature for locations throughout Colorado.

```
# load in the data
data(COmonthlyMet)

# getting the relevant data from the dataset
locations <- CO.loc
observations <- CO.tmean.MAM.climate

# use LatticeKrig to estimate a fit for these data points
kFit <- LatticeKrig(locations, observations)
```

```
## Warning in LatticeKrig(locations, observations): NAs removed
```

```
# plot the predicted surface over the given latitude/longitude window
surface(kFit)

# draw the USA state lines in gray, 2 pixels wide, to show where Colorado is
US(add=TRUE, col='gray', lwd=2)
```



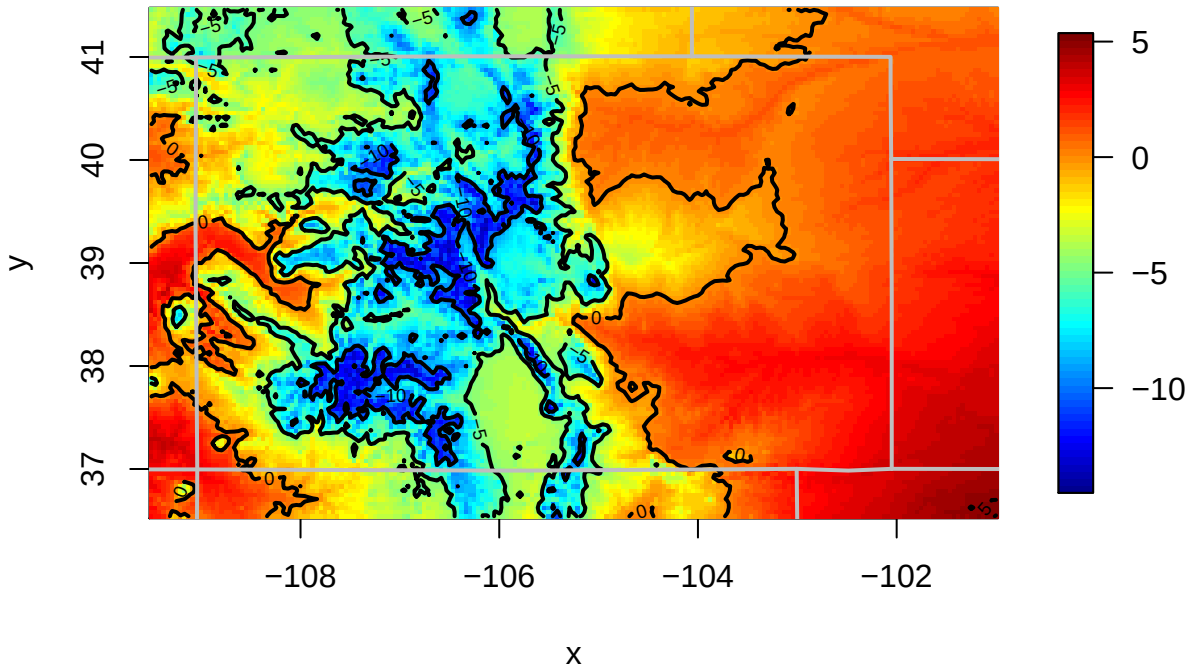
This plot is nice, but we can do better. We can see that the coldest temperatures are in the Rocky Mountains, which is unsurprising. Thus, we might expect that we will get a more accurate fit by having `LatticeKrig` account for the elevation at each location as well. Another way we can improve the plot is by increasing its resolution - the current plot is somewhat pixelated. We can tell the `surface` function to evaluate the surface at more points by using the `nx` and `ny` arguments, which will take longer to compute but produces a nicer looking, more detailed plot. Finally, we can also have `surface` extend the computation all the way to the corners of the window by using the `extrap` argument; by default it doesn't extrapolate outside of the existing data, since the `LatticeKrig` fitting method isn't designed to extrapolate and so the expected error increases dramatically when predicting outside of the given data. However, extending the plot to the corners will make it look nicer.

```
# load in the data
data(COmonthlyMet)

# getting the relevant data from the dataset
locations <- CO.loc
observations <- CO.tmean.MAM.climate

# get the elevations to include in our model
elevations <- CO.elev

# use LatticeKrig to estimate a fit for these data points
# this time, we include the elevations in Z, which stores
# variables other than the location for the model to use
kFit <- LatticeKrig(locations, observations, Z=cbind(elevations))
```

```
## Warning in LatticeKrig(locations, observations, Z = cbind(elevations)): NAs
```

```
## removed
```

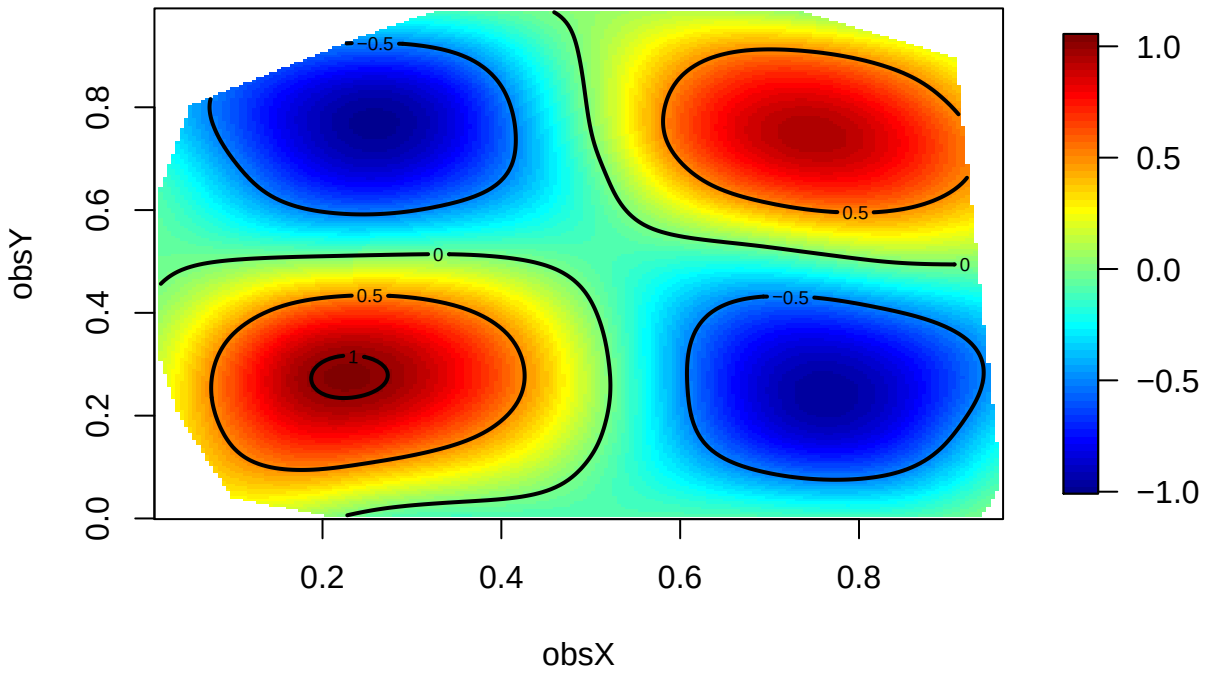
```
# plot the predicted surface over the given latitude/longitude window
# this time, we make predictions on a grid 200 wide and 150 tall,
# instead of 80 wide and 80 tall by default, and extrapolate to the corners.
prediction <- predictSurface(kFit, grid.list = CO.Grid, ZGrid = CO.elevGrid,
                             nx = 200, ny = 150, extrap = TRUE)

# draw the predicted surface
surface(prediction)

# draw the USA state lines in gray, 2 pixels wide, to show where Colorado is
US(add=TRUE, col='gray', lwd=2)
```



3 The LatticeKrig Function

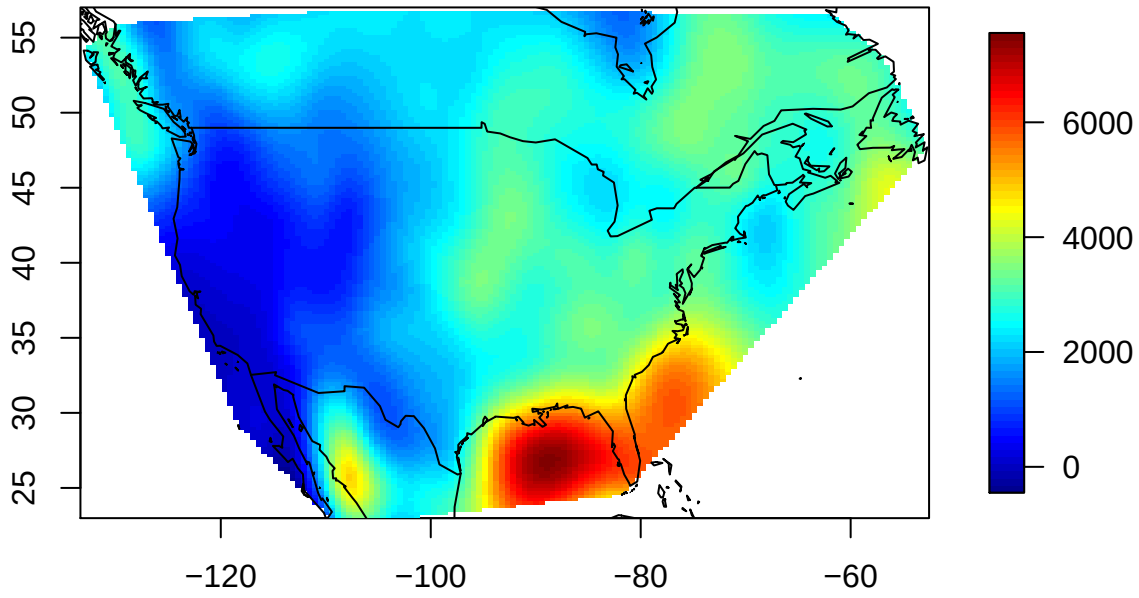
The LatticeKrig function takes in a set of locations and variable observations and returns a LKrig object that can be used to estimate the variable at other locations. It also has many optional parameters that can be used to adjust the model. In the following toy example, we take 100 samples of $\sin(2\pi x)\sin(2\pi y)$ 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.

```
# 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)
```



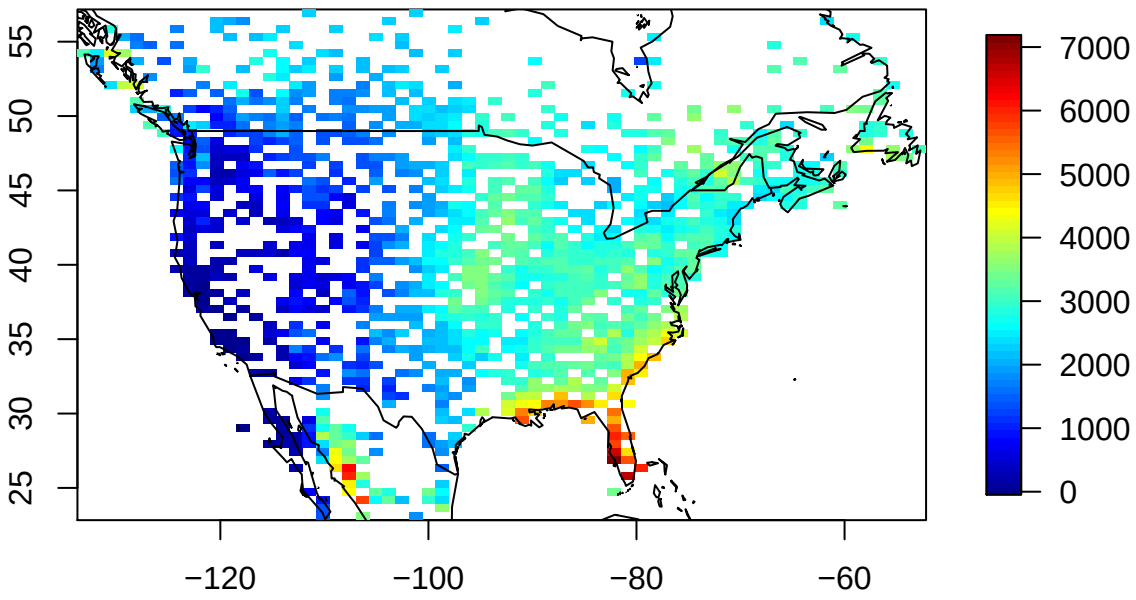
For another example, we will fit a surface to the NorthAmericanRainfall dataset included in the fields package.

```
library(fields)
data(NorthAmericanRainfall)
obsX <- cbind(NorthAmericanRainfall$longitude, NorthAmericanRainfall$latitude)
obsY <- NorthAmericanRainfall$precip
kFit <- LatticeKrig(obsX, obsY)
prediction <- predictSurface(kFit, nx = 150, ny = 150)
image.plot(prediction)
world(add=TRUE)
```



Note that all of the data in this dataset is taken on land (as shown below) and this model does not account for the difference between land and ocean, so the rainfall predictions this model makes over the Atlantic Ocean and Gulf of Mexico may not be accurate.

```
quilt.plot(obsX, obsY)
world(add=TRUE)
```



3.1 Fitting models with additional covariates

4 Frequently Asked Questions

4.1 The predicted values from my Kriging fit are nowhere near the data; what's wrong?

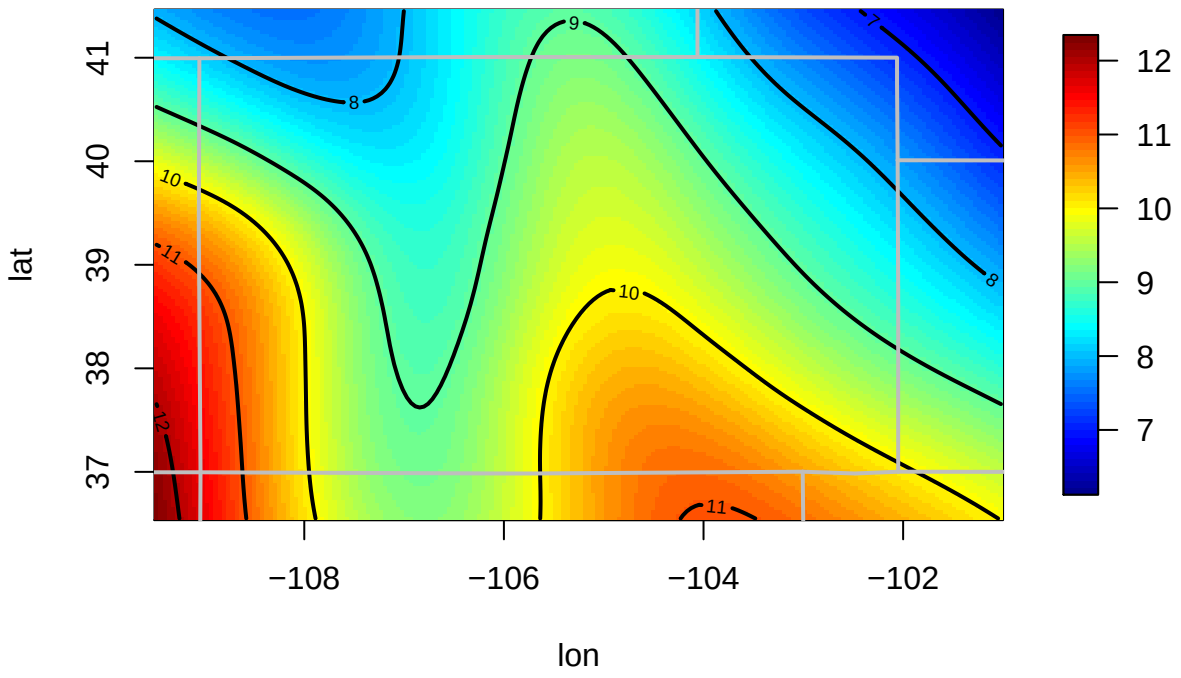
If your model includes covariates (the `Z` parameter of `LatticeKrig` and `LKrig`), your plot may not have included the effect of the covariate. The following code demonstrates this issue using the Colorado temperature data and how to fix it; first, we will set up the model.

```
# load in the data
data(COmonthlyMet)

# getting the relevant data from the dataset
locations <- CO.loc
observations <- CO.tmean.MAM.climate

# get the elevations to include in our model
elevations <- CO.elev

# use LatticeKrig to estimate a fit for these data points
kFit <- LatticeKrig(locations, observations, Z=cbind(elevations))
```

```
## Warning in LatticeKrig(locations, observations, Z = cbind(elevations)): NAs
## removed
```

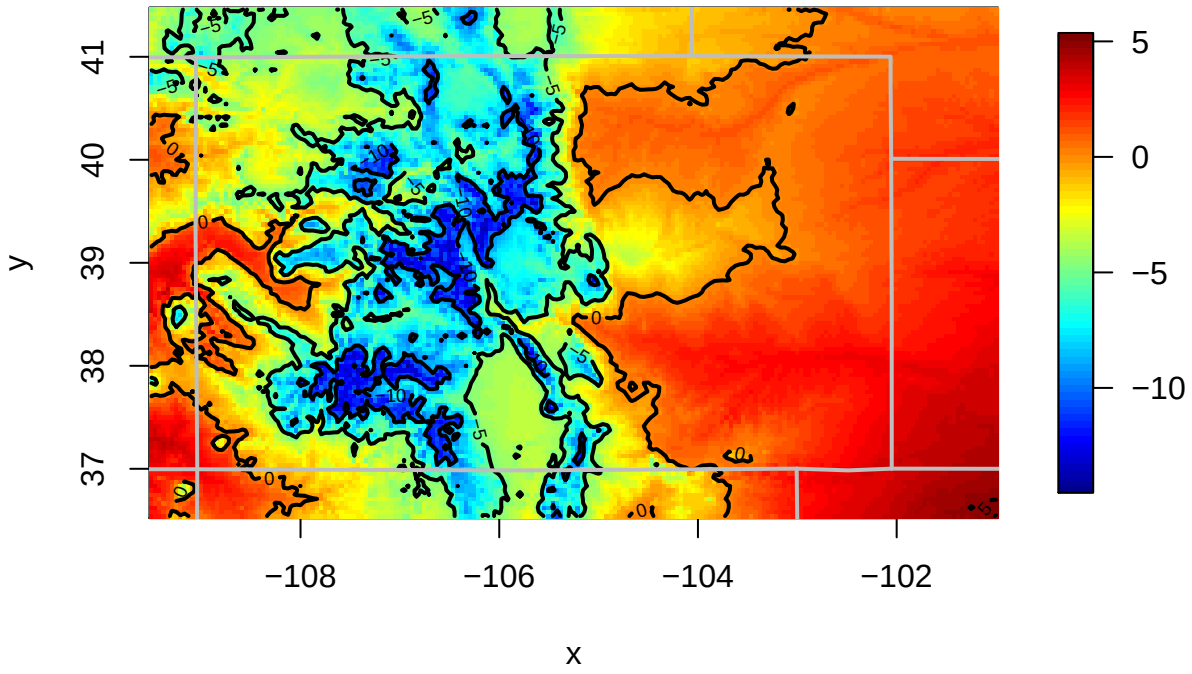
Using the `surface` function will leave out the covariate, resulting in a plot that doesn't match the original data and is smoother than we might expect.

```
# plot the predicted surface over the given latitude/longitude window
surface(kFit, nx = 200, ny = 150, extrap = TRUE)

#draw the USA state lines in gray, 2 pixels wide, to show where Colorado is
US(add=TRUE, col='gray', lwd=2)
```



To fix this, call `surface` on a `predictSurface` object instead of on an `LKrig` object, and make sure to pass in the `grid.list` and `ZGrid` parameters to the `predictSurface` call.

```
# predict the surface over the given latitude/longitude window, including the covariate
prediction <- predictSurface(kFit, grid.list = CO.Grid, ZGrid = CO.elevGrid, nx = 200, ny = 150, extrap

#draw the predicted surface
surface(prediction)

#draw the USA state lines in gray, 2 pixels wide, to show where Colorado is
US(add=TRUE, col='gray', lwd=2)
```


y
41
40
39
38
37
−108
−106
−104
−102
x
5
0
−5
−10

5 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 Z containing the coordinates of the locations and possibly other related information. 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 has a global maximum at 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 $\Sigma = S^T K S$, where K is a matrix of fixed size, independent of the number of observations. This form of Σ also makes 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 radial 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 $\mathbf{y}_2$.