Poission Smoothing Example

Doug Nychka

2025-02-18

```
library( statmod)
suppressMessages(library( fields) )
suppressMessages( library( statmod))
setwd("~/Dropbox/Home/Desktop2/PhDProjects/densityOlga")
```

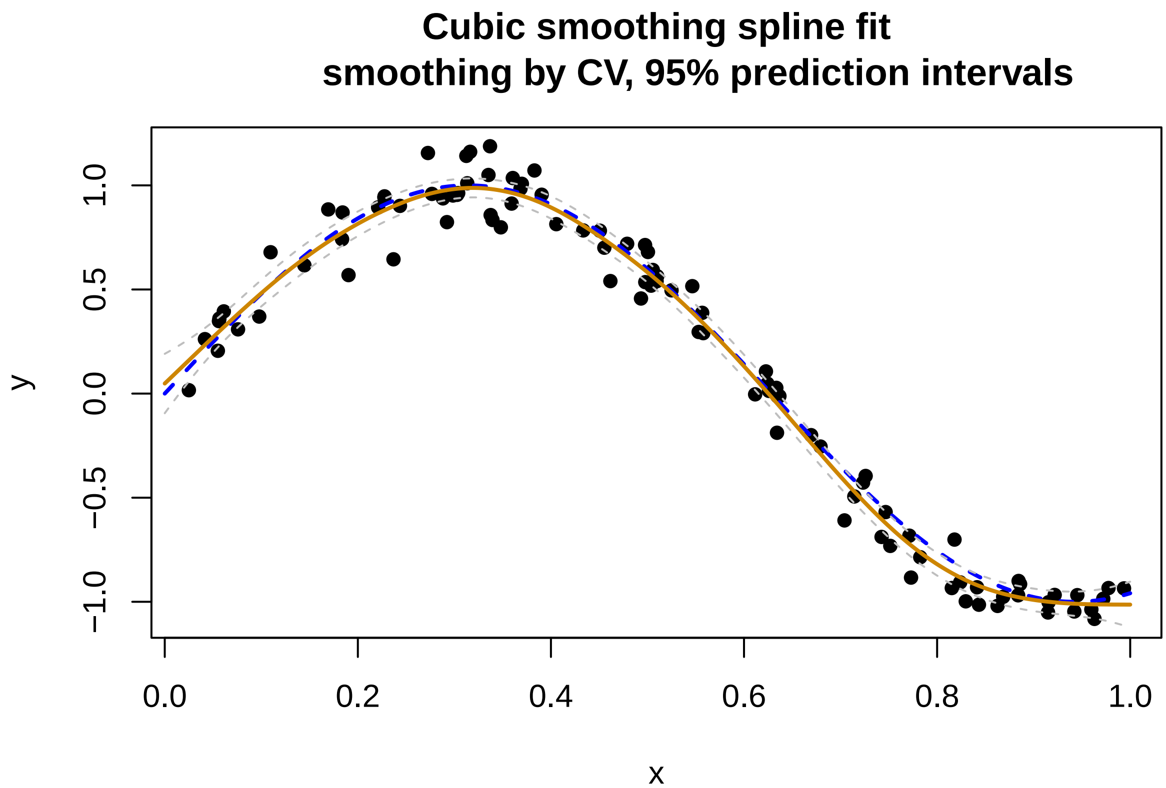
Some background

This example is an introduction to using a common curve fitting method called a spline. The function **Tps** implements this in the fields package and it is a likelihood based method to fit curves and surfaces. For the 1-D case in problems 1 and 2 the default method is know as a cubic smoothing spline although this is not important to do this HW.

Using the function Tps in R.

Here is a quick 1 D example how to use **Tps**. In looking at this code observe that the Tps function does not take the "formula" syntax. Just give it the locations and the values for the data in that order.

```
#some simulated data
set.seed(113)
n<- 100
# random points between [0,1]
x<-  sort(runif( 100))
# test function
y<- sin(5*x) + .1*rnorm( n)
# true curve we are trying to estimate
xGrid<- seq( 0,1,length.out= 300)
fTrue<- sin(5*xGrid)

# fit a spline
obj<-Tps( x, y) # amount of smoothing  found by cross-validation
# take a look
  plot( x,y, pch=16)
  lines( xGrid, fTrue, col="blue", lty=2, lwd=2)
  # predicted curve
  fhat<- predict( obj,xGrid)
  lines( xGrid, fhat, lwd=2,, col="orange3")
  # standard errors of prediction
  SE<- predictSE( obj, xGrid)
  # 95% confidence envelope
  lines( xGrid, fhat + 1.96* SE, col="grey", lty=2)
  lines( xGrid, fhat - 1.96*SE,  col="grey", lty=2)
  title("Cubic smoothing spline fit
        smoothing by CV, 95% prediction intervals")
```



More about Tps

The **Tps** object has quite a bit of information about the fit.

```r
print( obj)
```

```
## Call:
## Tps(x = x, Y = y)
##
##  Number of Observations:                100
##  Number of parameters in the null space 2
##  Parameters for fixed spatial drift     2
##  Model degrees of freedom:              7.6
##  Residual degrees of freedom:           92.4
##  GCV estimate for tau:                  0.0975
##  MLE for tau:                           0.1045
##  MLE for sigma:                         14.04
##  lambda                                 0.00078
##  User supplied sigma                    NA
##  User supplied tau^2                    NA
## Summary of estimates:
##                   lambda      trA        GCV     tauHat -lnLike Prof converge
## GCV         0.0007782923 7.608059 0.01028970 0.09750312    -73.55961        1
## GCV.model             NA       NA         NA         NA           NA       NA
## GCV.one     0.0007782923 7.608059 0.01028970 0.09750312           NA        1
## RMSE                  NA       NA         NA         NA           NA       NA
```

```
## pure error           NA       NA         NA         NA           NA       NA
## REML       0.0002227867 9.967793 0.01051223 0.09728513    -77.38011        9
```

The error variance is estimated by "tau squared" and the estimate reported here is close to the true error standard deviation.

The

```
Model degrees of freedom:  7.6
```

Indicates that about 8 "effective" parameters are needed to represent this curve. (The number of effective parameters does not have to be a whole number.) This value, by default, is found from the data by cross-validation. But to fix it at a certain amount, for example for 3.5 degrees of freedom or 15 you can just add the argument `df`

```
obj3_5<-Tps( x, y, df= 3.5)
obj15<-Tps( x, y, df= 15)

gHat3_5<-  predict(obj3_5,xGrid)
gHat15 <- predict(obj15,xGrid)
```

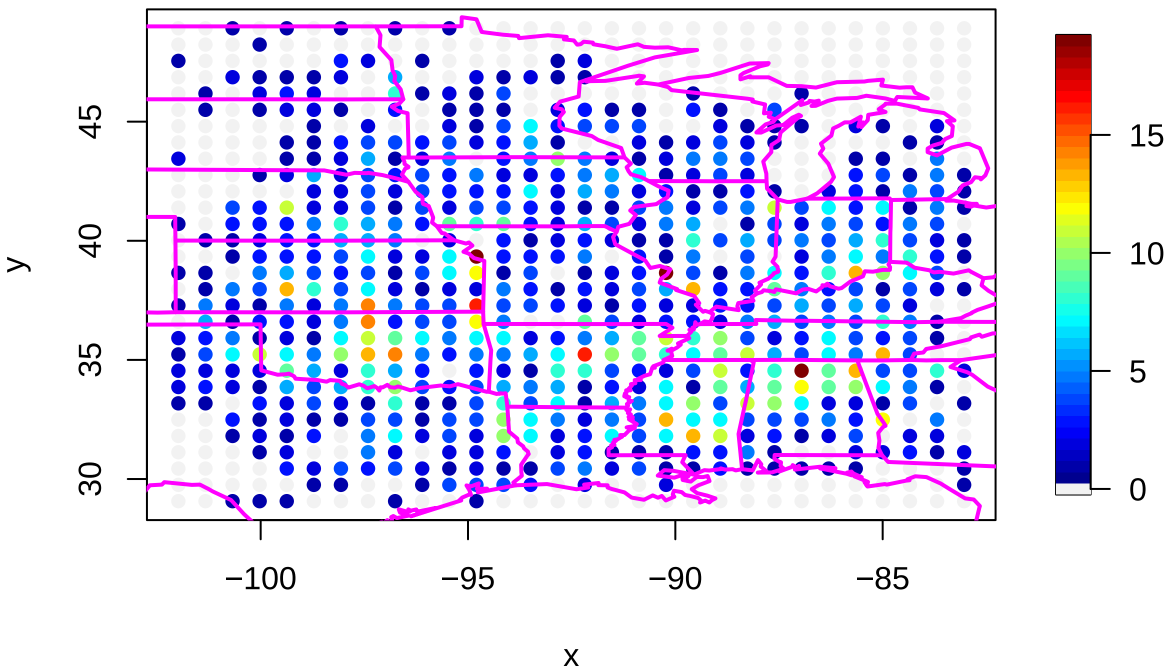
The returned object also includes the predicted values at the data points and the residuals using the usual format. E.g. for the fit above these are `obj$fitted.values` and `obj$residuals`.

Note that the `predict` function for the Tps object only returns the predicted curve, use `predictSE` to get the standard errors.

Tornado occurence over space.

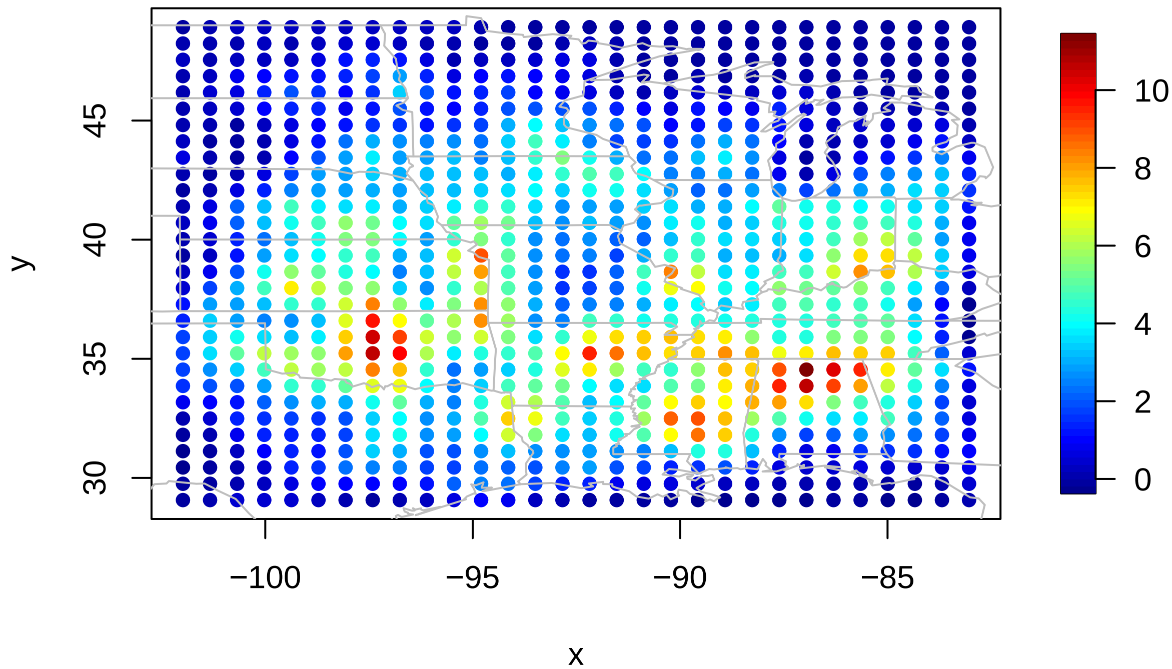
The code below reads in and then plots a version of the tornado data aggregated over space. Here the Midwest US is divided up into grid boxes (30X30) and the number of tornados are counted over the period 1950-2020 for each grid box. The data set is **loc**, the grid box centers, in longitude and latitude and **y** the counts in each grid box. The bubblePlot below gives a useful plot of these data.

```
load("tornadoCountsSpace.rda")

# set zero counts to grey to make plot
# easier to interpret and also just use a small number of colors
ctab<- c("grey95", tim.colors(40))

bubblePlot( loc, y,col=ctab,
            highlight=FALSE
            )
US( add=TRUE, col="magenta", lwd=2)
```

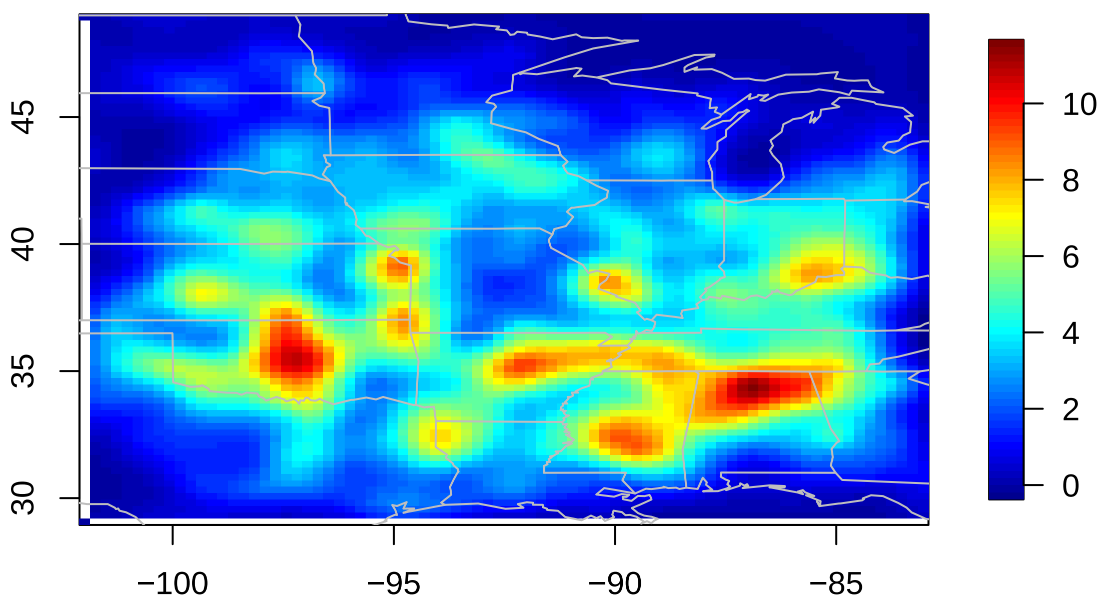


Here is a naive surface fit to these counts using Tps. It is not right because it does not adjust for Poisson data and also because it is not constrained to be positive. But this is an example of how to fit a spatial data set and get a quick plot the surface using **bubblePlot**. Since there are 900 locations this takes more time to fit using **Tps** than the analysis in Problems 1 and 2. The first surface plot is straight forward using the predicted values. The second evaluates on a fine grid (with the surface values in the `out$z` component.)

```
TpsObj<- Tps( loc,y)
bubblePlot(loc, TpsObj$fitted.values,
           col=tim.colors())
US( add=TRUE, col="grey")
```




```r
# a more detailed plot evaluating the fit on a finer grid

out<- predictSurface( TpsObj)
imagePlot( out,  col=tim.colors())

US( add=TRUE, col="grey")
```



Poission regression

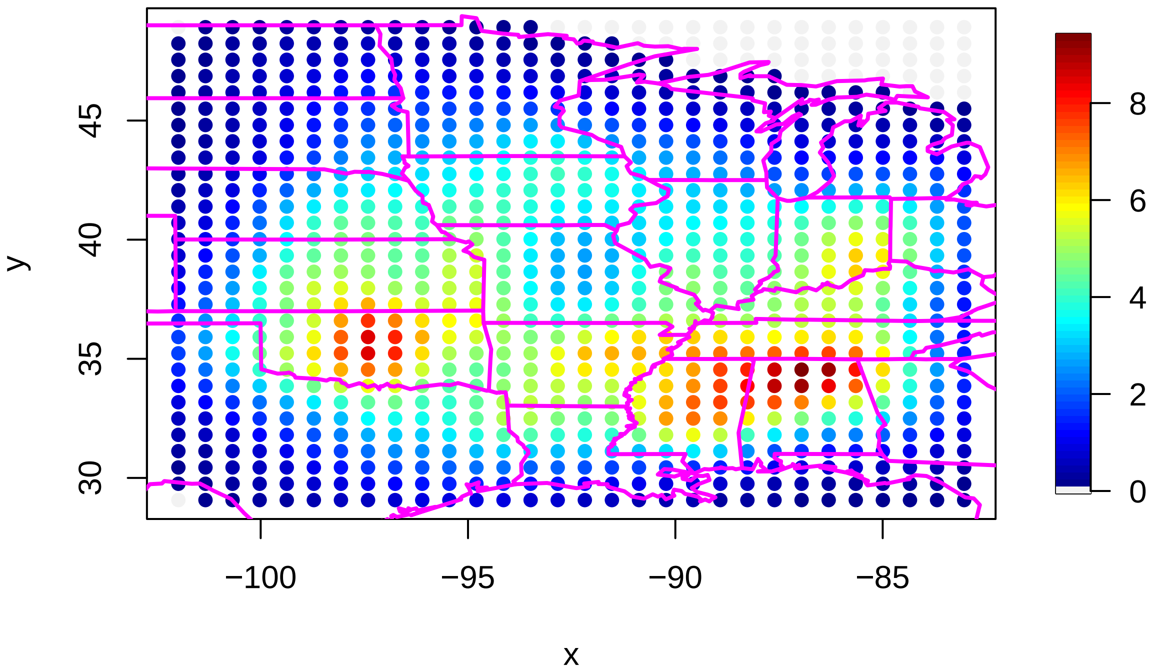
One can refit this surface fitting using a Poisson model for the counts and fit using the IRW algorithm.

The model is the number of tornados in grid box at location s is Poisson with expected value exp(g(s)) ad g is assumed to be a smooth surface given by a Gaussian process – in this case a thin plate spline. Note that in the fitting the effective degrees of freedom for the is specified and varying this will effect the smoothness of the surface.

```
nuOld<- rep( log( mean(y)), length(y))

for( k in 1:10){
  muI<- exp( nuOld)
  W<- c(muI)
  z<- nuOld  + (1/muI)*( y- muI)
  # in place of WLS -- a smoothing/curve fitting step
  # note that smooting found by default method ( CV)
  tempObj<- Tps( loc,z,
                weights=W,
                df=60, give.warnings=FALSE)
  nuNew <- tempObj$fitted.values
  test<- sqrt( mean( (nuNew- nuOld)^2))
  cat( k, test, fill=TRUE)
  nuOld<- nuNew
}
```

```
## 1 0.7455175
## 2 0.4084566
## 3 0.2705066
## 4 0.1499811
## 5 0.03667542
## 6 0.001998531
## 7 1.179648e-05
## 8 3.166767e-05
## 9 7.953344e-07
## 10 4.08871e-13
```

```
ctab<- c("grey95", tim.colors( 64))
bubblePlot(
     loc,exp(tempObj$fitted.values),
     col=ctab,
     highlight=FALSE)

     US( add=TRUE, col="magenta", lwd=2)
```




```
# a surface plot
objSur<- predictSurface( tempObj, nx=150, ny=300)
imagePlot(objSur$x, objSur$y, exp(objSur$z), col=ctab )
US( add=TRUE, col="magenta", lwd=2)
```