Week 7 — Resampling, Smoothing splines and GAM

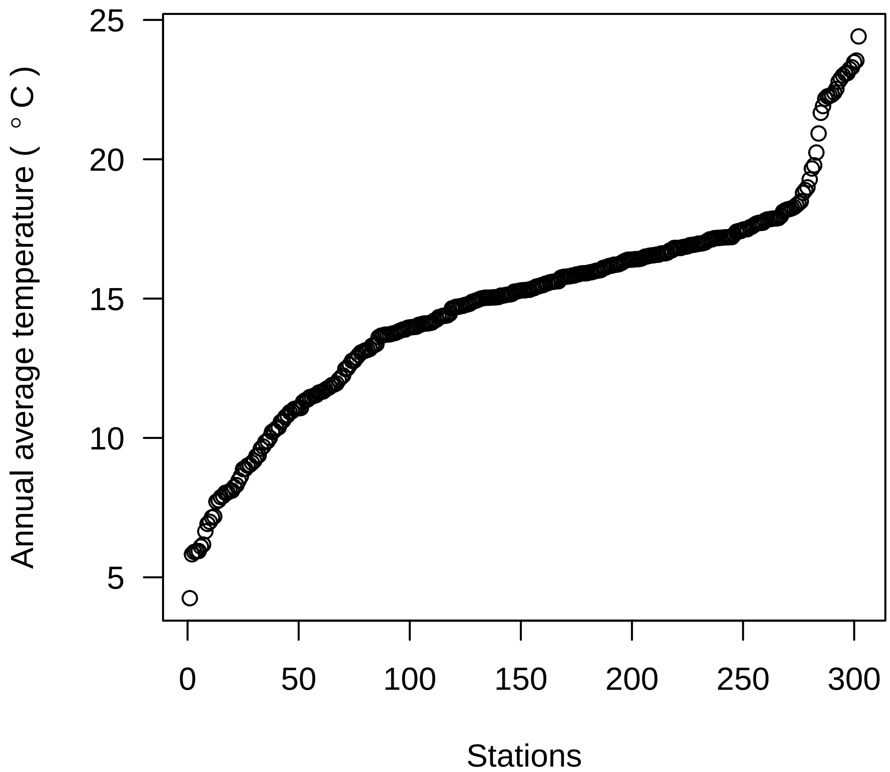
ISLR Chapter 5 (Monday)

Question 1: *Please answer Question 5 (The first of the "Applied questions) on page 198 of ISLR*

ISLR Chapter 7 (Wednesday)

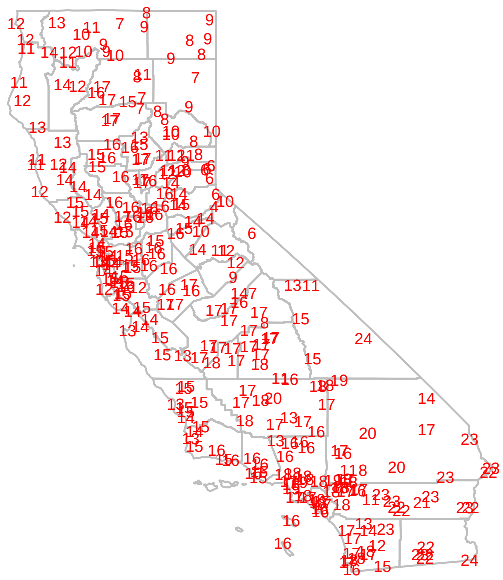
We will be working with temperature data for California. Here is some climate data from California weather stations.

```
d <- read.csv("temperature.csv")
head(d)
##      ID             NAME     LONG    LAT  ALT  JAN  FEB  MAR  APR  MAY
## 1 19303 PARKER-RESERVOIR -114.166 34.283  225 11.9 14.8 17.8 21.9 26.8
## 2 19335    IRON-MOUNTAIN -115.133 34.133  281 12.2 14.8 17.5 21.6 26.4
## 3 19343   EAGLE-MOUNTAIN -115.450 33.800  297 12.6 15.2 17.7 21.6 26.1
## 4 19347 MITCHELL-CAVERNS -115.533 34.933 1326  7.8  9.4 10.9 14.8 19.6
## 5 19348    MOUNTAIN-PASS -115.533 35.466 1442  3.9  5.8  8.1 11.9 16.7
## 6 19349  EL-CENTRO-2-SSW -115.566 32.766   -9 12.6 14.9 17.2 20.3 24.7
##    JUN  JUL  AUG  SEP  OCT  NOV  DEC
## 1 32.1 35.2 34.3 30.8 24.6 17.2 12.2
## 2 31.7 34.9 33.8 30.0 23.9 17.0 12.3
## 3 31.4 34.2 33.4 30.0 24.2 17.5 13.0
## 4 25.1 28.2 27.0 23.6 18.4 12.0  8.1
## 5 22.6 26.3 25.0 21.1 15.0  8.7  4.2
## 6 29.5 32.9 32.7 29.3 23.6 16.9 12.5
d$temp <- rowMeans(d[, c(6:17)])
plot(sort(d$temp), ylab=expression('Annual average temperature ( '~degree~'C )'),
     las=1, xlab='Stations')
```



We can make a simple (and illegible) map of the values

```
library(raster)
CA <- shapefile("counties_2000.shp")
plot(CA, border='gray')
text(d[,3:4], labels=round(d$temp), cex=.5, col='red')
```



Or map this via spplot, for easy color-coding. In this case we need to first make a SpatialPointsDataFrame. To do so, we can either use the formula notation, or use the `SpatialPoints` function. I use the `SpatialPoints` function here.

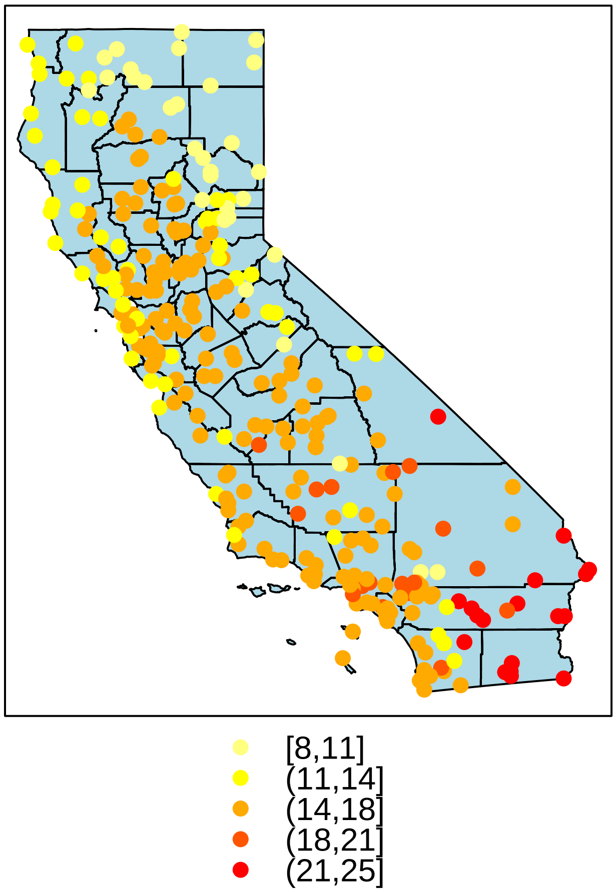
```
dsp <- SpatialPoints(d[,3:4], proj4string=CRS("+proj=longlat +datum=NAD83"))
dsp
## class       : SpatialPoints
## features    : 302
## extent      : -124.233, -114.166, 32.57, 41.966  (xmin, xmax, ymin, ymax)
## crs         : +proj=longlat +datum=NAD83 +ellps=GRS80 +towgs84=0,0,0
```

Combine the `SpatialPoints` with its `data.frame`.

```
dsp <- SpatialPointsDataFrame(dsp, d)
dsp
## class       : SpatialPointsDataFrame
## features    : 302
## extent      : -124.233, -114.166, 32.57, 41.966  (xmin, xmax, ymin, ymax)
## crs         : +proj=longlat +datum=NAD83 +ellps=GRS80 +towgs84=0,0,0
## variables   : 18
## names       :    ID,                 NAME,     LONG,    LAT,  ALT,  JAN,  FEB,  MAR, APR,  MAY,  JUN,
## min values  : 19303, ADIN-RANGER-STATION, -124.233,  32.57,  -59, -3.8, -2.5, -1.9, 0.9,  5.1,  9.9,
## max values  : 24549,        YUMA ARIZONA, -114.166, 41.966, 2438, 14.7,   16, 19.3,  24, 29.3, 34.6,
```

And now plot

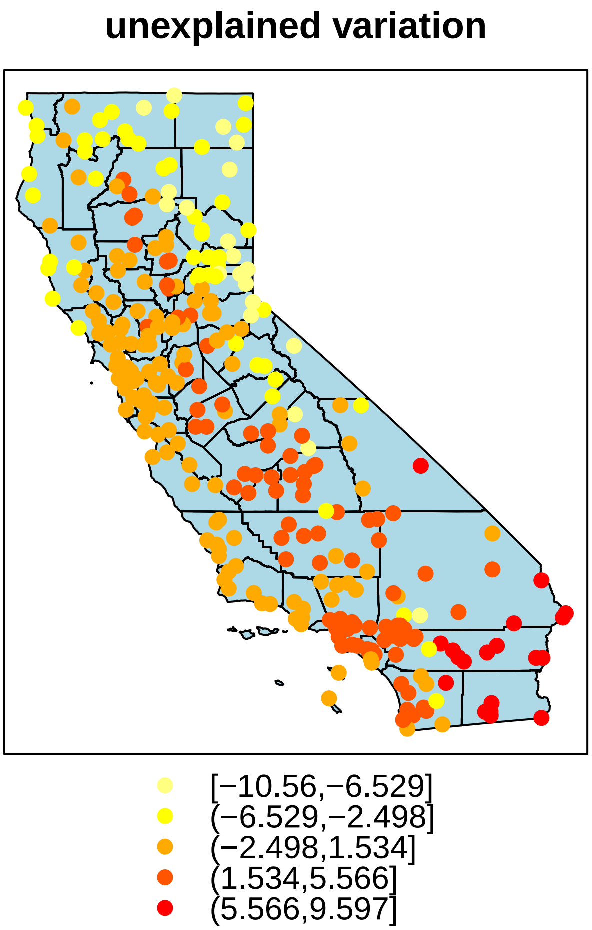
```r
cuts <- c(8,11,14,18,21,25)
pols <- list("sp.polygons", CA, fill = "lightblue")
print(spplot(dsp, 'temp', cuts=cuts, sp.layout=pols,
    col.regions=rev(heat.colors(5))))
```



NULL model

A *null model* to explain the variation in temperature data could take the mean annual temperature across the stations would be a good estimator of temperature at any location. I first compute that value, and then compute the difference between that value and the observed values. And I make a map of the differences.

```r
summary(d$temp)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    4.25   12.91   15.30   14.81   16.92   24.41
tavg <- mean(d$temp)
tavg
## [1] 14.81087
dsp$diff <- dsp$temp - tavg
print(spplot(dsp, 'diff', col.regions=rev(heat.colors(5)),
    sp.layout=pols, main = "unexplained variation" ))
```



This does not look very good. There are large differences, and they are strongly spatially structured (autocorrelated). Let's define an RMSE function and compute RMSE for the null-model (on the training data in this case).

```
RMSE <- function(observed, predicted) {
  sqrt(mean((predicted - observed)^2, na.rm=TRUE))
}

RMSE(tavg, dsp$temp)
## [1] 3.722373
```

As the NULL-model does not look good, let's see if we can improve upon it.

Transform and sample

Transformation

First, I transform the data to a planar CRS (Teale Albers in this case) to assure that the computations are OK. That is, we want to avoid interpreting angles as if they were planar coordinates.

```
TA <- CRS(" +proj=aea +lat_1=34 +lat_2=40.5 +lat_0=0 +lon_0=-120 +x_0=0
          +y_0=-4000000 +datum=NAD83 +units=m +no_defs +ellps=GRS80 +towgs84=0,0,0")
library(rgdal)
dta <- spTransform(dsp, TA)
cata <- spTransform(CA, TA)
```

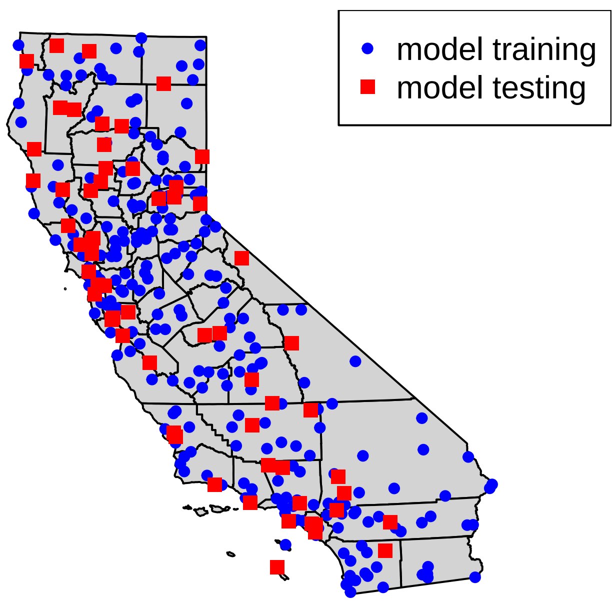
Sampling for cross-validation

I assign the data to five bins to do five fold cross-validation.

```
# to always have the same random result, I set the seed.
set.seed(5162016)
library(dismo)
k <- kfold(dta)
table(k)
## k
##  1  2  3  4  5
## 60 61 60 61 60
```

To illustrate what I have done for one 'fold':

```
test <- dta[k==1, ]
train <- dta[k!=1, ]
plot(cata, col='light gray')
points(train, pch=20, col='blue')
points(test, pch=15, col='red')
legend('topright', c('model training', 'model testing'), pch=c(20, 15), col=c('blue', 'red'))
```



Linear model

First, fit a simple linear model. This is sometimes referred to as a 'trend surface' (see Chapter 9 in oSU). I get the coordinates (in Teale Albers) of the train data set and I combine these with the values of interest (temp).

```
df <- data.frame(coordinates(train), temp=train$temp)
colnames(df)[1:2] = c('x', 'y')
```

Then I fit a linear (regression) model to the data

```
m <- glm(temp ~ x+y, data=df)
summary(m)
##
## Call:
## glm(formula = temp ~ x + y, data = df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -9.5810  -1.5613   0.1474   1.8181   8.8619
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.390e+01  2.135e-01  65.073  < 2e-16 ***
## x            7.109e-07  1.521e-06   0.467    0.641
## y           -8.675e-06  1.100e-06  -7.887  1.1e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 8.236277)
##
##     Null deviance: 3579.8  on 241  degrees of freedom
## Residual deviance: 1968.5  on 239  degrees of freedom
## AIC: 1202
##
## Number of Fisher Scoring iterations: 2
```

Question 2: *Describe (in statistical terms) and explain (in physical terms) the results shown by* `summary(m)`

Question 3: *According to this model. How much does the temperature in California change if you travel 500 miles to the north (show the R code to compute that)*

We can now estimate temperature values at any location with the predict function. For example for our hold-out (test) sample.

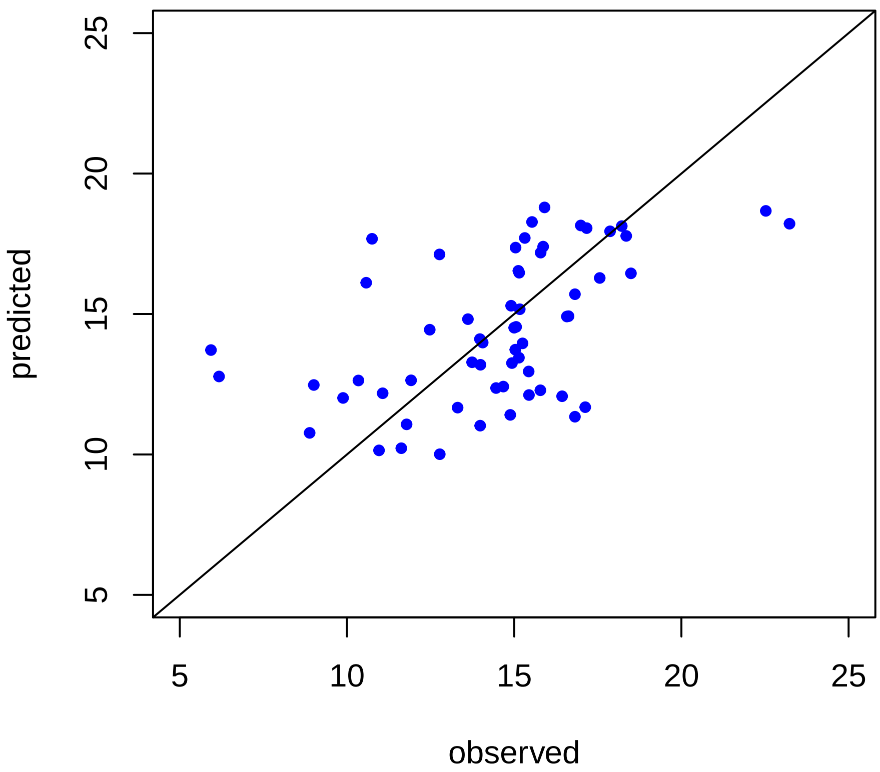
```
v <- data.frame(coordinates(test))
colnames(v)[1:2] = c('x', 'y')
p <- predict(m, v)
head(p)
##        1        2        3        4        5        6
## 18.67154 17.67720 17.39930 17.94395 16.28229 18.12864
```

And we can evaluate the results by comparing them with the known values for these locations.

```
# first the null model
RMSE(mean(train$temp), test$temp)
## [1] 3.179886
# now the linear model
RMSE(p, test$temp)
```

```
## [1] 2.84847

plot(test$temp, p, xlim=c(5,25), ylim=c(5,25), pch=20,
    col='blue', xlab='observed', ylab='predicted')
# for reference: y=x
abline(0,1)
```



OK, now the same thing but k-(5)-fold:

```
r <- rep(NA,5)
for (i in 1:5) {
  test <- dta[k==i, ]
  train <- dta[k!=i, ]
  df <- data.frame(coordinates(train), temp=train$temp)
  m <- glm(temp ~ ., data=df)
  v <- data.frame(coordinates(test))
  p <- predict(m, v)
  r[i] <- RMSE(p, test$temp)
}
r
## [1] 2.848470 3.227967 2.675434 2.819683 2.757730
mean(r)
## [1] 2.865857
```

Question 4: *Was it important to do 5-fold cross-validation, instead of a single 20-80 split?*

The model is not great, but it did capture something, and it is better than the null model. We can also predict values for grid cells. I first create a raster with the extent of California, and with 10 km resolution (square) grid cells.

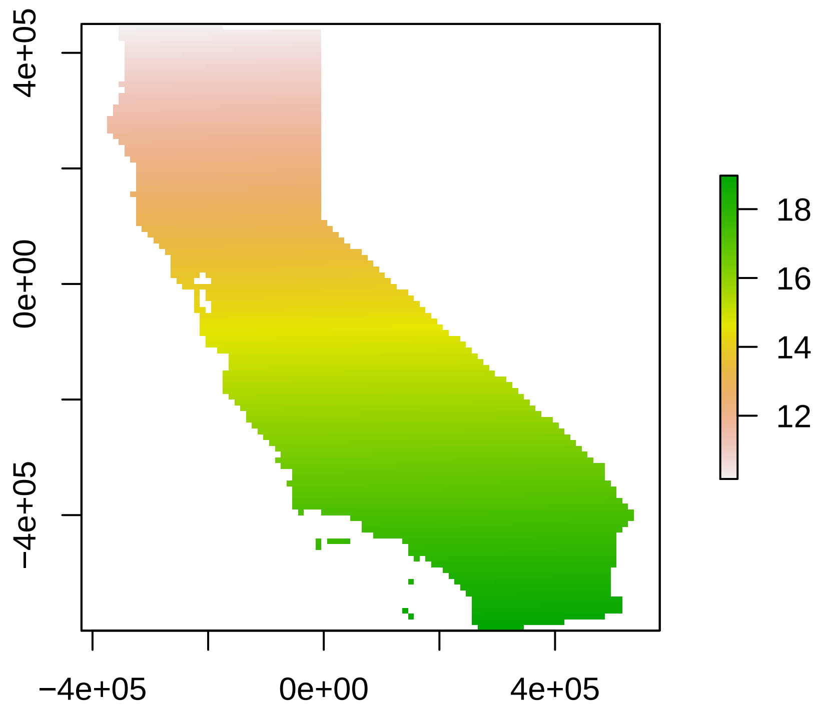
```
r <- raster(round(extent(cata)), res=10000, crs=TA)
```

I will show two ways to estimate the values for this raster. The 'hard' way:

```
# get the x coordinates
x <- init(r, v='x')
# set areas outside of CA to NA
x <- mask(x, cata)
# get the y coordinates
y <- init(r, v='y')
# combine the two variables (RasterLayers)
s <- stack(x,y)
names(s) <- c('x', 'y')
```

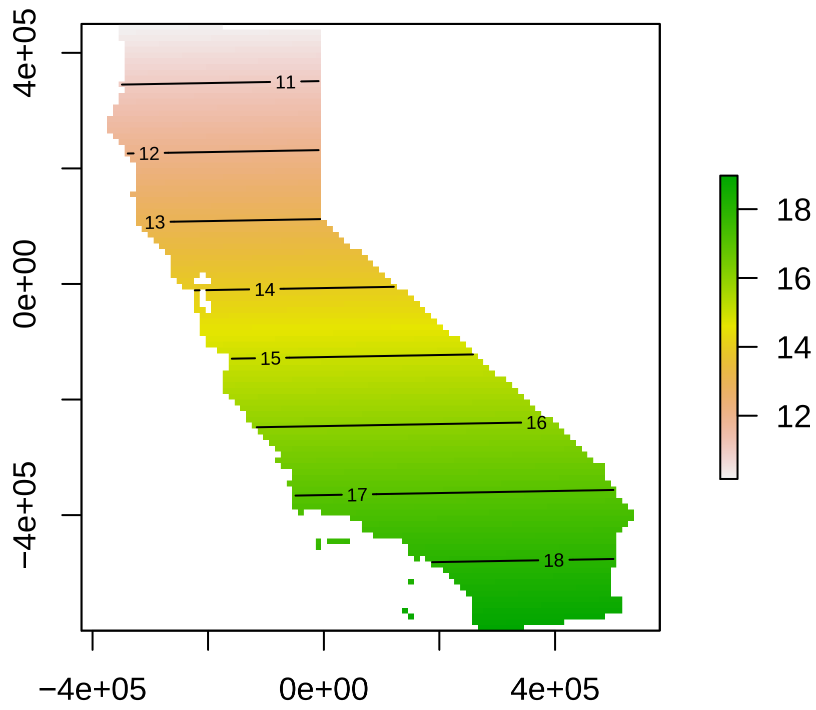
Now make a model with all data (no splits).

```
df <- data.frame(coordinates(dta), temp=dta$temp)
colnames(df)[1:2] = c('x', 'y')
m <- glm(temp ~ ., data=df)
# predict
trend <- predict(s, m)
plot(trend)
```



Instead of predict, we can use the `interpolate` function. That is a simpler approach as `interpolate` “knows” about needing to use coordinates.

```
z <- interpolate(r, m)
mask <- mask(z, cata)
zm <- mask(z, mask)
plot(zm)
contour(zm, add=TRUE)
```

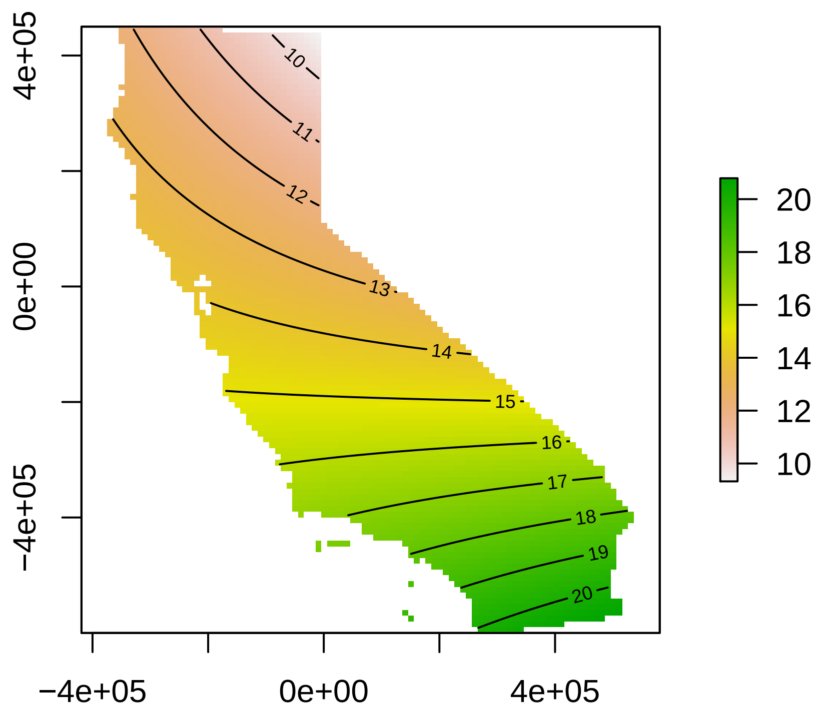


Here is an alternative models with interaction terms. I am onlry using a single split (no k-fold) to not clutter the example too much.

```
df <- data.frame(coordinates(dta), temp=dta$temp)
colnames(df)[1:2] = c('x', 'y')
test <- df[k==1, ]
train <- df[k!=1, ]

m <- glm(temp ~ x*y, data=train)
summary(m)
##
## Call:
## glm(formula = temp ~ x * y, data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -9.6890  -1.6651   0.0988   2.0840   9.7998
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.325e+01  2.864e-01  46.275  < 2e-16 ***
## x           -2.845e-06  1.841e-06  -1.545  0.12371
## y           -9.210e-06  1.090e-06  -8.448 2.97e-15 ***
## x:y         -1.306e-11  3.969e-12  -3.290  0.00115 **
```

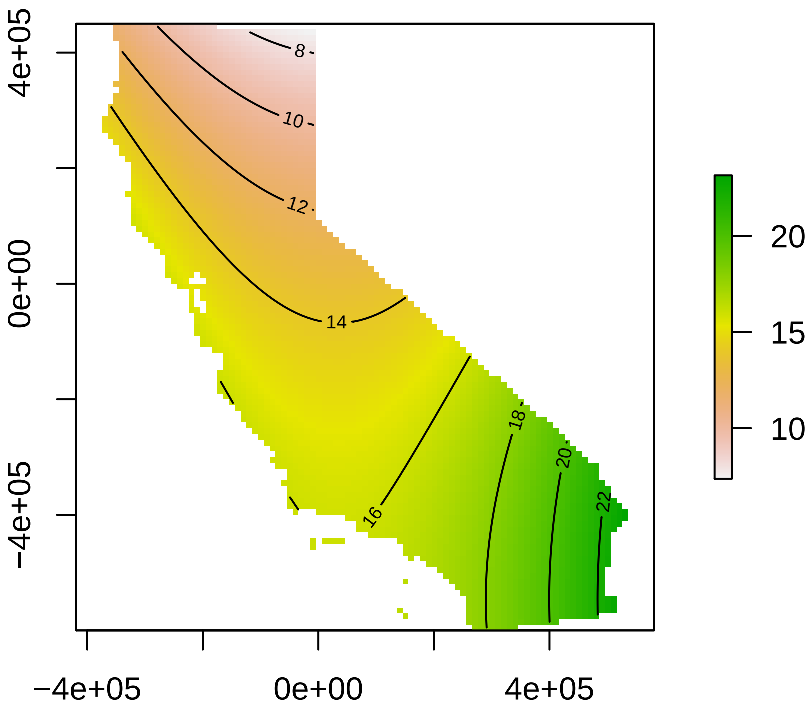
```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 7.91116)
##
##     Null deviance: 3579.8  on 241  degrees of freedom
## Residual deviance: 1882.9  on 238  degrees of freedom
## AIC: 1193.3
##
## Number of Fisher Scoring iterations: 2
AIC(m)
## [1] 1193.255
RMSE(predict(m, test), test$temp)
## [1] 2.591301
z <- interpolate(r, m)
zm <- mask(z, mask)
plot(zm)
contour(zm, add=TRUE)
```



A model with polynomial terms

```
m <- glm(temp ~ x + y + I(x^2) + I(y^2), data=df)
summary(m)
```

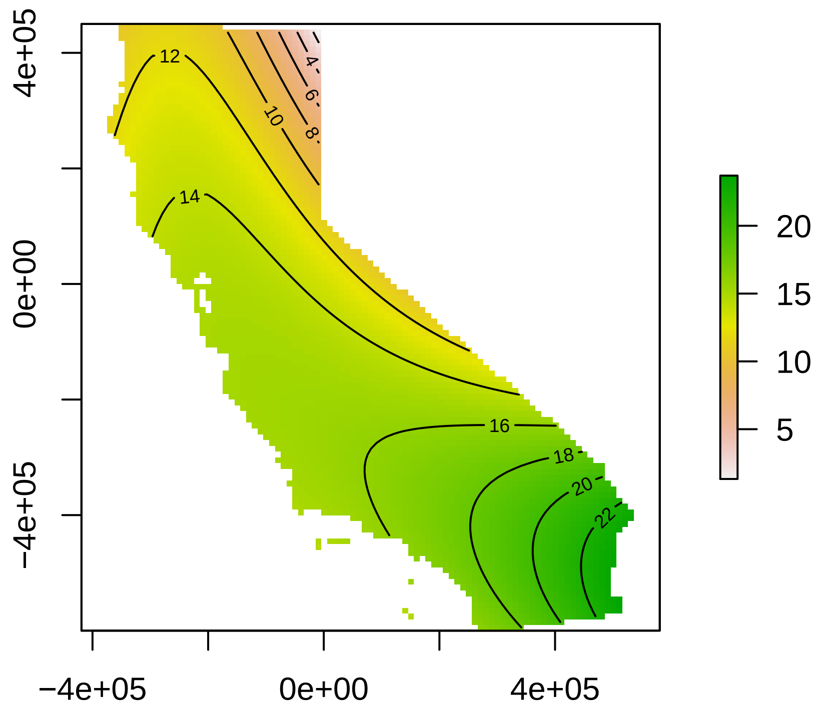
```
##
## Call:
## glm(formula = temp ~ x + y + I(x^2) + I(y^2), data = df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -9.6891  -1.0553   0.0012   1.7994   7.8593
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.340e+01  2.681e-01  49.986  < 2e-16 ***
## x           -2.003e-06  1.389e-06  -1.443     0.15
## y           -9.938e-06  9.116e-07 -10.903  < 2e-16 ***
## I(x^2)       2.935e-11  3.480e-12   8.433 1.49e-15 ***
## I(y^2)      -9.097e-12  2.221e-12  -4.095 5.44e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 6.592786)
##
##     Null deviance: 4184.5  on 301  degrees of freedom
## Residual deviance: 1958.1  on 297  degrees of freedom
## AIC: 1433.6
##
## Number of Fisher Scoring iterations: 2
AIC(m)
## [1] 1433.562
RMSE(predict(m, test), test$temp)
## [1] 2.57128
z <- interpolate(r, m)
zm <- mask(z, mask)
plot(zm)
contour(zm, add=TRUE)
```



Second-order polynomials and interactions

```
m <- glm(temp ~ poly(x, 2, raw=TRUE) * poly(y, 2, raw=TRUE), data=df)
summary(m)
##
## Call:
## glm(formula = temp ~ poly(x, 2, raw = TRUE) * poly(y, 2, raw = TRUE),
##     data = df)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -10.4185   -0.9676    0.1926    1.4591   10.8233
##
## Coefficients:
##                                                  Estimate Std. Error
## (Intercept)                                     1.338e+01  3.014e-01
## poly(x, 2, raw = TRUE)1                        -1.595e-05  2.719e-06
## poly(x, 2, raw = TRUE)2                        -4.384e-11  1.162e-11
## poly(y, 2, raw = TRUE)1                        -1.636e-05  1.353e-06
## poly(y, 2, raw = TRUE)2                        -2.987e-11  4.383e-12
## poly(x, 2, raw = TRUE)1:poly(y, 2, raw = TRUE)1 -1.032e-10  1.596e-11
## poly(x, 2, raw = TRUE)2:poly(y, 2, raw = TRUE)1 -1.899e-16  3.735e-17
## poly(x, 2, raw = TRUE)1:poly(y, 2, raw = TRUE)2 -1.087e-16  2.224e-17
## poly(x, 2, raw = TRUE)2:poly(y, 2, raw = TRUE)2 -1.212e-22  5.460e-23
```

```r
##                                                t value Pr(>|t|)
## (Intercept)                                     44.398  < 2e-16 ***
## poly(x, 2, raw = TRUE)1                         -5.867 1.20e-08 ***
## poly(x, 2, raw = TRUE)2                         -3.773 0.000195 ***
## poly(y, 2, raw = TRUE)1                        -12.097  < 2e-16 ***
## poly(y, 2, raw = TRUE)2                         -6.816 5.33e-11 ***
## poly(x, 2, raw = TRUE)1:poly(y, 2, raw = TRUE)1  -6.468 4.14e-10 ***
## poly(x, 2, raw = TRUE)2:poly(y, 2, raw = TRUE)1  -5.084 6.62e-07 ***
## poly(x, 2, raw = TRUE)1:poly(y, 2, raw = TRUE)2  -4.888 1.68e-06 ***
## poly(x, 2, raw = TRUE)2:poly(y, 2, raw = TRUE)2  -2.220 0.027215 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 5.78023)
##
##     Null deviance: 4184.5  on 301  degrees of freedom
## Residual deviance: 1693.6  on 293  degrees of freedom
## AIC: 1397.7
##
## Number of Fisher Scoring iterations: 2
AIC(m)
## [1] 1397.744
RMSE(predict(m, test), test$temp)
## [1] 2.22646
z <- interpolate(r, m)
zm <- mask(z, mask)
plot(zm)
contour(zm, add=TRUE)
```



Question 5: *What is the best model sofar? Why?*

Question 6: *Rerun the last model using (a) ridge regression, and (b) lasso regression. Show the changes in coefficients for three values of lambda; by finishing the code below*

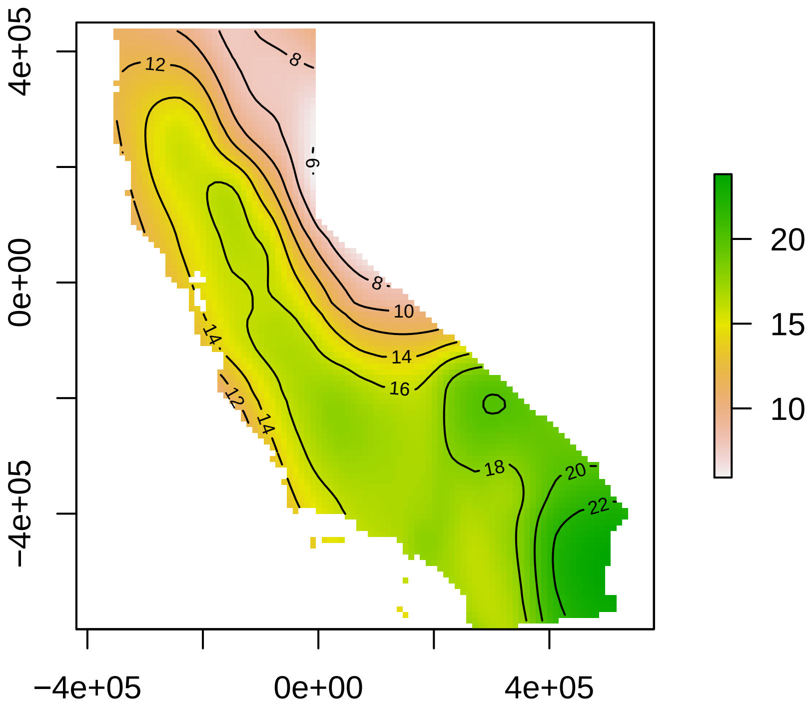
```
f <- temp ~ poly(x, 2, raw=TRUE) * poly(y, 2, raw=TRUE)
x <- model.matrix(f, df)
library(glmnet)
## Loading required package: Matrix
##
## Attaching package: 'Matrix'
## The following object is masked from 'package:spam':
##
##     det
## Loading required package: foreach
## Loaded glmnet 2.0-16
g <- glmnet(x, df$temp)
```

Local regression

We can use the `loess` function, or the `locfit` library (And see the lab on GWR!)

```
m <- loess(temp ~ x + y, span=.2, data=train)
z <- interpolate(r, m)
```

```
zm <- mask(z, mask)
plot(zm)
contour(zm, add=TRUE)
```




```
RMSE(predict(m, test), test$temp)
## [1] 2.053495
```

Question 7: *What does the the "span" argument represent?*

Thin plate splines

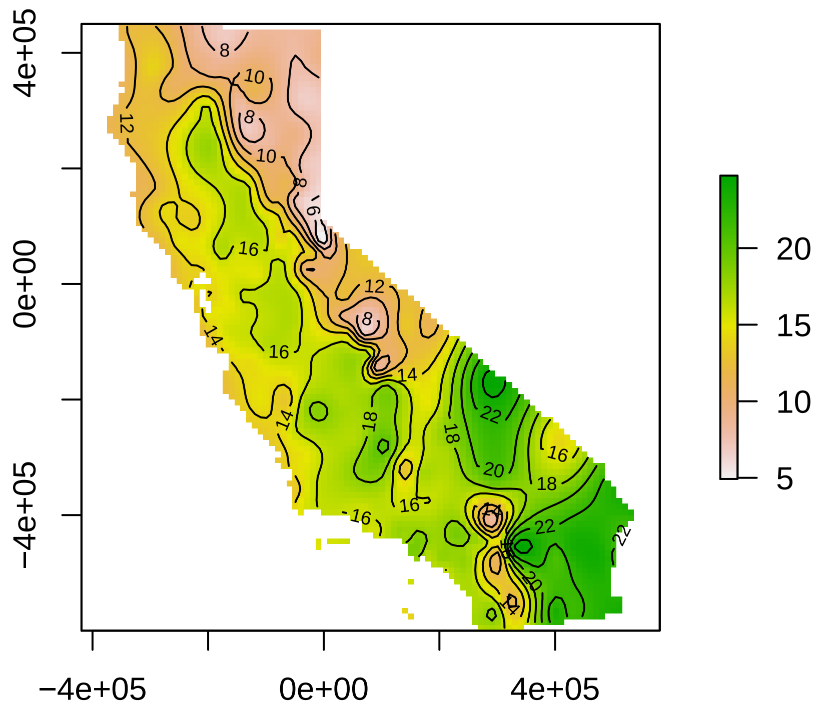
Now to thin-plate splines (~ n-dimensioal smooting splines). First fit the model.

```
library(fields)
tps <- Tps(train[, c('x', 'y')],  train$temp)
## Warning:
## Grid searches over lambda (nugget and sill variances) with  minima at the endpoints:
##    (GCV) Generalized Cross-Validation
##     minimum at  right endpoint  lambda  =  9.606714e-07 (eff. df= 229.9 )
tps
## Call:
## Tps(x = train[, c("x", "y")], Y = train$temp)
##
##  Number of Observations:                 242
```

```
##  Number of parameters in the null space 3
##  Parameters for fixed spatial drift     3
##  Model degrees of freedom:              229.9
##  Residual degrees of freedom:           12.1
##  GCV estimate for sigma:                0.2831
##  MLE for sigma:                         0.2578
##  MLE for rho:                           69200
##  lambda                                 9.6e-07
##  User supplied rho                      NA
##  User supplied sigma^2                  NA
## Summary of estimates:
##                   lambda       trA      GCV      shat -lnLike Prof converge
## GCV        9.606714e-07 229.90000 1.602401 0.2830549     536.8624       NA
## GCV.model            NA        NA       NA        NA           NA       NA
## GCV.one    9.606714e-07 229.90000 1.602401 0.2830549           NA       NA
## RMSE                 NA        NA       NA        NA           NA       NA
## pure error           NA        NA       NA        NA           NA       NA
## REML       2.973892e-04  69.19357 3.146969 1.4990581     507.1963        6
```

Now make a prediction for the raster cells.

```
ptps <- interpolate(r, tps)
ptps <- mask(ptps, mask)
plot(ptps)
contour(ptps, add=TRUE)
```

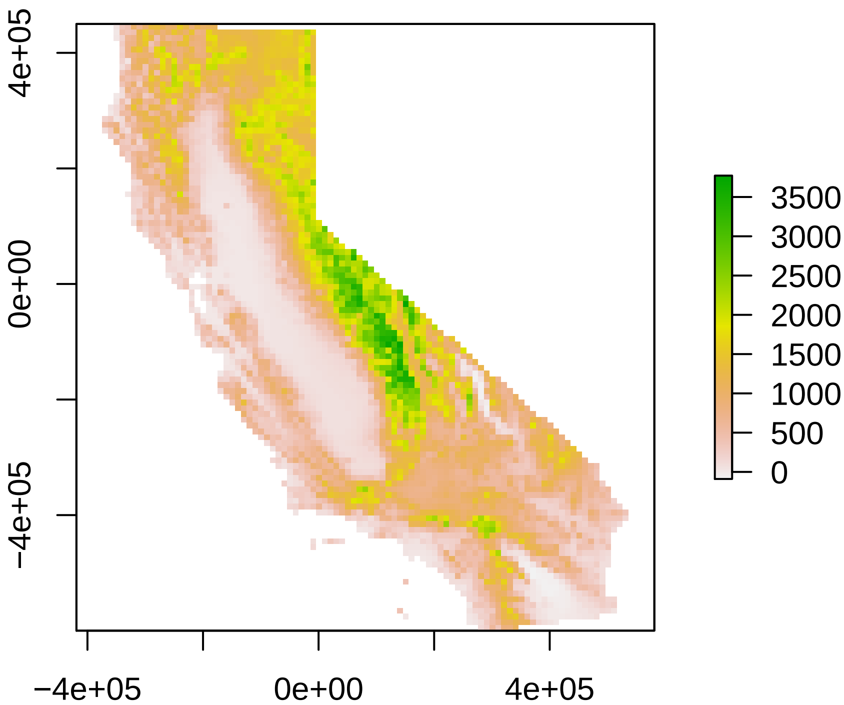


And predict to the test points.

```r
pt <- predict(tps, test[, c('x', 'y')])
RMSE(pt, test$temp)
## [1] 2.167317
```

Now let's bring in elevation as a co-variable. First get and prepare the elevation data.

```r
elv1 <- raster::getData('worldclim', res=0.5, var='alt', lon=-122, lat= 37)
elv2 <- raster::getData('worldclim', res=0.5, var='alt', lon=-120, lat= 37)
elv <- merge(elv1, elv2, overlap=FALSE)
telv <- projectRaster(elv, r)
celv <- mask(telv, mask)
names(celv) <- 'elevation'
plot(celv)
```



Extract elevation values for test and train points

```
train$elevation <- extract(celv, train[, 1:2])
test$elevation <- extract(celv, test[, 1:2])
```

There are a few points just outside this raster, that have `NA` values for elevation. I remove these for now.

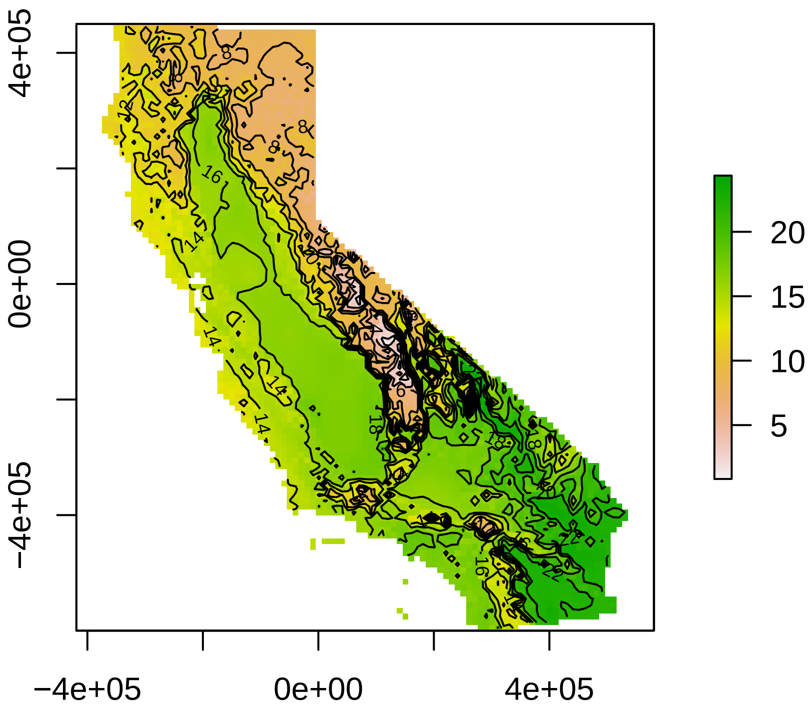
```
train <- train[!is.na(train$elevation), ]
test <- test[!is.na(test$elevation), ]
```

Fit the model, now with an additional variable.

```
tps2 <- Tps(train[, c('x', 'y', 'elevation')], train$temp)
## Warning:
## Grid searches over lambda (nugget and sill variances) with  minima at the endpoints:
##   (GCV) Generalized Cross-Validation
##    minimum at  right endpoint  lambda  =  7.244989e-05 (eff. df= 221.35 )
```

And predict to grid cells.

```
ptps2 <- interpolate(celv, tps2, xyOnly=FALSE)
plot(ptps2)
contour(ptps2, add=TRUE)
```



Evaluate

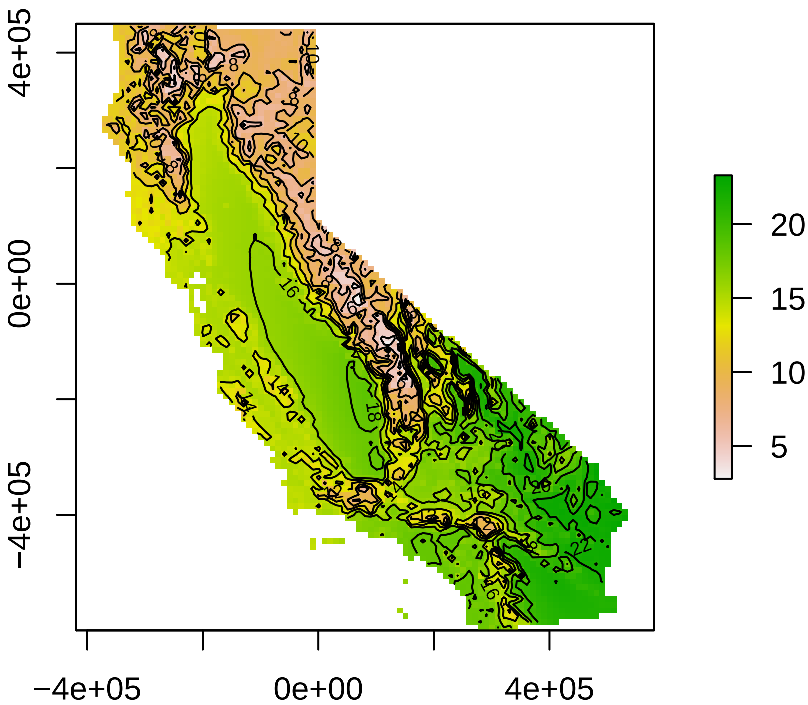
```
pt2 <- predict(tps2, test[,  c('x', 'y', 'elevation')])
RMSE(test$temp, pt2)
## [1] 1.762481
```

Question 8: *What is a main reason that this the best prediction sofar?*

General additive models

Here is a quick GAM example, using the mgcv package.

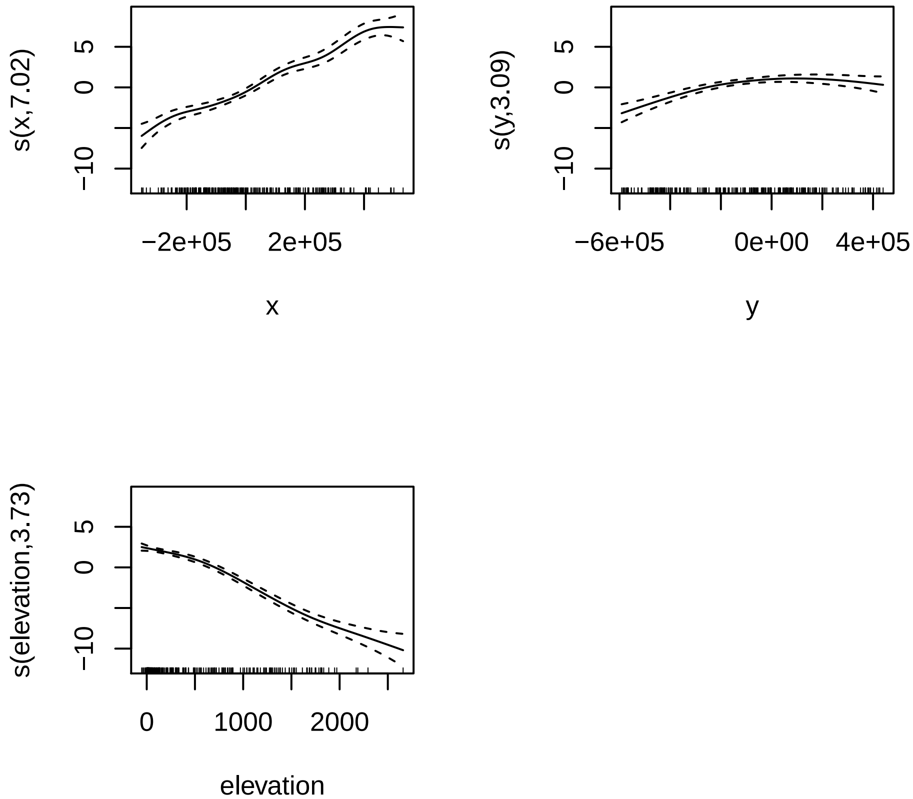
```
library(mgcv)
ga <- gam(temp ~ s(x) + s(y) + s(elevation), data=train)
x <- interpolate(celv, ga, xyOnly=FALSE)
plot(x)
contour(x, add=TRUE)
```




```r
pg <- predict(ga, test)
RMSE(test$temp, pg)
## [1] 1.699617
```

To see the responses to the variables do

```r
plot.gam(ga, pages=1)
```



Fitting gams is an art. Below is one example of what you might do for spatial interpolation.

Question 9: *Use the help files to exaplin the model below. What do you think of it? Is it better or worse than the gam we did above?*

```
ga2 <- gam(temp ~ te(x, y, k=12, bs='ts') + s(elevation, bs='ts'), data=train)
```