Exercise: Thin plate spline interpolation

D G Rossiter
Cornell University; 中国科学院南京土壤所

March 3, 2016

Contents

1 Introduction **1**

2 1-D spline interpolation **1**
2.1 Non-smoothing splines 2
2.2 Smoothing splines 9

3 Thin-plate splines **12**
3.1 Theory 12
3.2 The `fields` R package 14
3.3 Fitting thin-plate splines 15
3.4 Evaluation 19

4 Comparison with a geostatistical approach **20**

5 Considerations for using thin-plate spline interpolation **24**

References **26**

Index of R concepts **28**

君子 上 达， 小人 下达
– 论文宪问23

"A person of noble character constantly improves, a person of low character continuously degenerates."
– The Analects, 14.23

1 Introduction

Spatial interpolation is the process of predicting values of a target variable or attribute at an unobserved location, from a set of values at known (observed) locations.

- In **ordinary kriging** (OK) interpolation we assume that the observations are the result of a locally spatially-correlated second-order stationary random process; we model that process, and use the model to predict.
- In **trend surface** (TS) interpolation we assume that the observations are the result of a regional process; we model that process with a single surface as a polynomial of the coördinates and predict from the equation of the surface.
- In **universal kriging** (UK) interpolation we assume that some of the variation is explained by regional processes, with some residual local variation explained by a locally spatially-correlated second-order stationary random process; we model the two processes simultaneously, and use the model to predict.

In this tutorial we introduce another method: fitting a surface (as in the trend surface) but adjusting to local observations (as in kriging) using **2-dimensional smoothing splines**. This method is not "geostatistical" in the sense that there is no assumed model, other than the requirement that the fitted surface be in some sense "smooth". Like UK, it fits a regional trend with local variations from the trend dependent on the observations. Unlike UK, the analyst does not perform a formal analysis of the spatial structure [4]; thus some authors consider splines to be suitable only for visualization [5]. This method of interpolation, under the name of "radial basis function" interpolation, is provided by the Geostatistical Analyst toolkit of ArcGIS [11] and of course is easily implemented in R.

In §5 we discuss the (dis)advantages of using smoothing splines for interpolation.

2 1-D spline interpolation

To illustrate how splines work, we begin with interpolation in one dimension by smoothing splines [9, §5.4] [18, §8.7]. This can be useful in its own right, shows how the mathematics work, and allows easy visualization. In §3 we then consider two-dimensional (surface) interpolation.

2.1 Non-smoothing splines

We first discuss non-smoothing splines, i.e., where some data values are respected.

A **spline** is a piecewise polynomial function (i.e., each piece is defined only over some range), with the pieces being joined at **knots**. These splines match the function value at each knot. Splines are a type of **basis expansion** Hastie et al. [9, §] , which allow linear methods to be used on non-linear relations. In particular, splines allow local fitting of piecewise polynomials to observations, with a defined degree of continuity between the pieces. An order-M spline is a piecewise polynomial function defined to have continuous derivatives up to order $(M - 2)$.

The most common order is the fourth, i.e., with two continuous derivatives, so that values, slopes and curvatures match at the knots. These look smooth to the human eye and also correspond to an intuitive concept of smoothness. A non-smoothing spline goes through each knot, i.e., the measured values are considered exact. When the spline is only defined in terms of the knots (including boundaries) it is called *basis* spline, often abbreviated to *B*-spline.

The coefficients of each piece are found from a system of linear equations with constraints of order $(M - 2)$; for the common fourth-order splines these are the first and second derivatives, as well as the data values, at the knots. So-called **natural** splines have another constraint: the function must be linear outside the boundary knots; this guards against high-order polynomial extrapolation outside the knots.

To compute a spline, the analyst must specify:

1. the **order** of the spline; most commonly cubic splines (4^{th} order);
2. the **number** of knots;
3. the **placement** of knots in the range of the variable.

The knots can be at the known data points, in which case the spline interpolation is exact at known points. A more common approach is to define equally-spaced knots through the range of the sequence, make a linear interpolation from adjacent known points, and fit the spline through the knots.

As an example, we use a dataset from a transect near Sandford-on-Thames (Oxfordshire, England), originally reported by Webster and Cuanalo [22], and also used by Davis [3, Example 4.12].[1] The soil was sampled at three depth intervals (5–6 cm thick, centred on 8, 30, 65 cm depth) by augering every 10 m along a regular transect, resulting in 321 sites.

TASK 1 : Load the example dataset and examine its structure. •

[1] This same example is used in the exercise on compositional variables.

This dataset was provided by Richard Webster to Murray Lark, who formatted it for easier processing. He in turn kindly provided it to us as file `sandford.txt`.

TASK 2 : Examine the text file structure in a text editor. •

```
> file.show("sandford.txt")
```

The first few lines are:

```
    Top             2               3
    Clay    Silt    Clay    Silt    Clay    Silt
1   70      15      85      10      84      14
2   65      20      75      15      85      10
3   65      20      75      10      70      20
```

Each row (after the headers) is an observation number in sequence along the transect, followed by the silt and clay in layers 1, 2 and 3 respectively. It's easiest to read this in without the header lines, and add our own column labels.

TASK 3 : Read the text file into a data frame and assign field names. •

Function `read.table` is the generic function to read text files; in this case the only non-default setting is the `skip` argument to skip the first two lines.

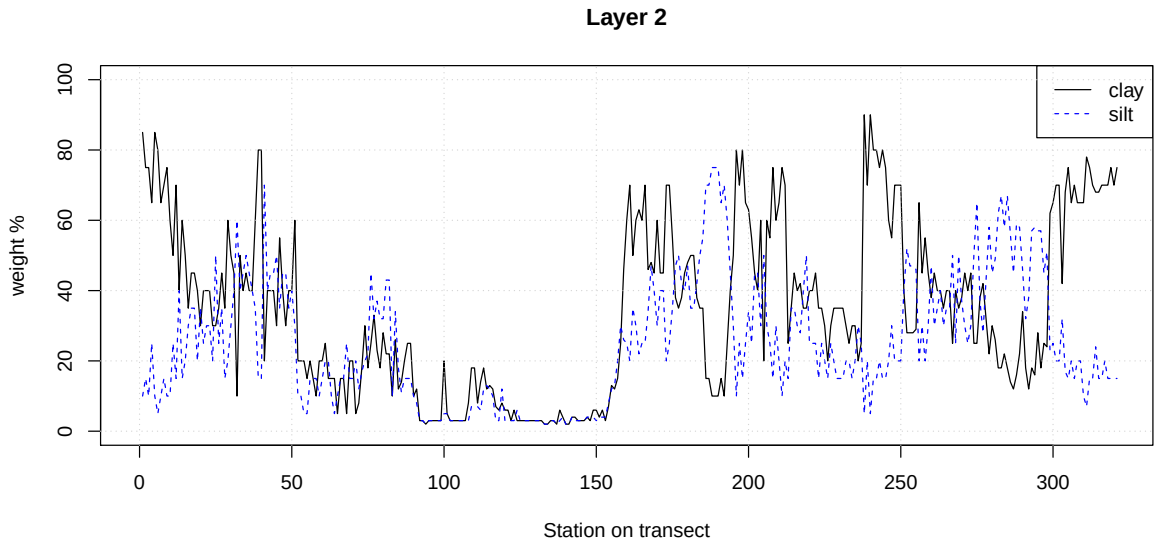
We read the file into an R data frame we name (unimaginatively) as `ds` (for 'dataset'):

```
> ds <- read.table("sandford.txt", skip=2)
> names(ds) <- c("seq","clay1","silt1","clay2","silt2","clay3","silt3")
```

```
'data.frame': 321 obs. of  7 variables:
 $ seq  : int  1 2 3 4 5 6 7 8 9 10 ...
 $ clay1: int  70 65 65 70 80 65 70 65 65 40 ...
 $ silt1: int  15 20 20 10 10 10 10 15 25 40 ...
 $ clay2: int  85 75 75 65 85 80 65 70 75 60 ...
 $ silt2: int  10 15 10 25 10 5 10 15 10 10 ...
 $ clay3: num  84 85 70 65 80 1 25 80 30 25 ...
 $ silt3: num  14 10 20 10 15 1 15 10 10 25 ...
```

TASK 4 : Display the clay and silt fractions along the transect for the second layer. •

```
> plot(ds$clay2, type="l", ylim=c(0,100), xlim=c(0,320), main="Layer 2", xlab="Station on transect", ylab="weight
> lines(ds$silt2, lty=2, col="blue")
> legend("topright", c("clay","silt"), lty=1:2, col=c("black","blue"))
> grid()
```



Clearly there is spatial dependence, and we could attempt to model this with a variogram and use the model to interpolate. However, we also see some abrupt changes, e.g., the transition from very low clay and silt from positions 120–150 to quite high clay and moderate silt by position 155.

TASK 5 : Fit a natural spline basis to the clay content of the second layer. •

The `ns` function of the `splines` package computes a B-spline basis matrix for a natural spline of order 4 (cubic). The analyst must decide on the number of knots and their placement; this determines how closely the spline matches the data values. For the placement, since the points were equally-spaced on the transect, it makes sense to also space the knots equally.[2] For the number of knots, we must decide on a smoothness: fewer knots gives fewer pieces and thus more smoothness. Looking at the plot, most of the short-range irregularity seems to be at about 10 stations; so we place a knot every 10 stations along the transect, with the `df` argument to `ns` function.

TASK 6 : Compute the spline basis. •

We first compute the spline **basis**, an $n \times k$ matrix (k knots at ech of

[2] We could also place the knots at what appear to be transitions between sections of the transect.

n observation) used when computing splines. These are weights that will be given to each observation when computing the coefficients of the splines.

```
> require(splines)

Loading required package:  splines

> (n <- length(ds$clay2))

[1] 321

> basis.10 <- ns(ds$seq, df=floor(n/10))
> str(basis.10)

 ns [1:321, 1:32] 0 0.000167 0.001333 0.0045 0.010667 ...
 - attr(*, "dimnames")=List of 2
  ..$ : NULL
  ..$ : chr [1:32] "1" "2" "3" "4" ...
 - attr(*, "degree")= int 3
 - attr(*, "knots")= Named num [1:31] 11 21 31 41 51 61 71 81 91 101 ...
  ..- attr(*, "names")= chr [1:31] "3.125%" "6.25%" "9.375%" "12.5%" ...
 - attr(*, "Boundary.knots")= int [1:2] 1 321
 - attr(*, "intercept")= logi FALSE
 - attr(*, "class")= chr [1:3] "ns" "basis" "matrix"
```

This spline basis has 32 cubic functions, each of which is defined at all 321 stations (data points), which will be used to determine the coefficients (see below). But most of the basis values at any station are zero.

TASK 7 : Display the non-zero basis values for stations 148 to 152. •

```
> for (i in 148:152) {
+     print(paste("Station",i))
+     print(round(basis.10[i,basis.10[i,]!=0],4))
+ }

[1] "Station 148"
    12     13     14     15
0.0045 0.3482 0.5902 0.0572
[1] "Station 149"
    12     13     14     15
0.0013 0.2827 0.6307 0.0853
[1] "Station 150"
    12     13     14     15
0.0002 0.2212 0.6572 0.1215
[1] "Station 151"
    13     14     15
0.1667 0.6667 0.1667
[1] "Station 152"
    13     14     15     16
0.1215 0.6572 0.2212 0.0002
```

For example, at station 150, four of the 32 basis functions have non-zero basis values, corresponding to polynomial pieces 12 . . . 15. Since station 151 is a knot, it only has values for three pieces.

TASK 8 : Compute the coefficients for each basis polynomial of the spline. •

Now we use this basis in the workhorse `lm` function to compute the coefficients for each basis polynomial, from the observed values of the

response variable, here clay percentage. The response variable is a vector of n observations, while the predictor is a $n \times k$ matrix; thus the coefficients are estimated by ordinary least squares:

$$\mathcal{Y} = XB + \varepsilon \longrightarrow \hat{B} = (X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}\mathcal{Y} \tag{1}$$

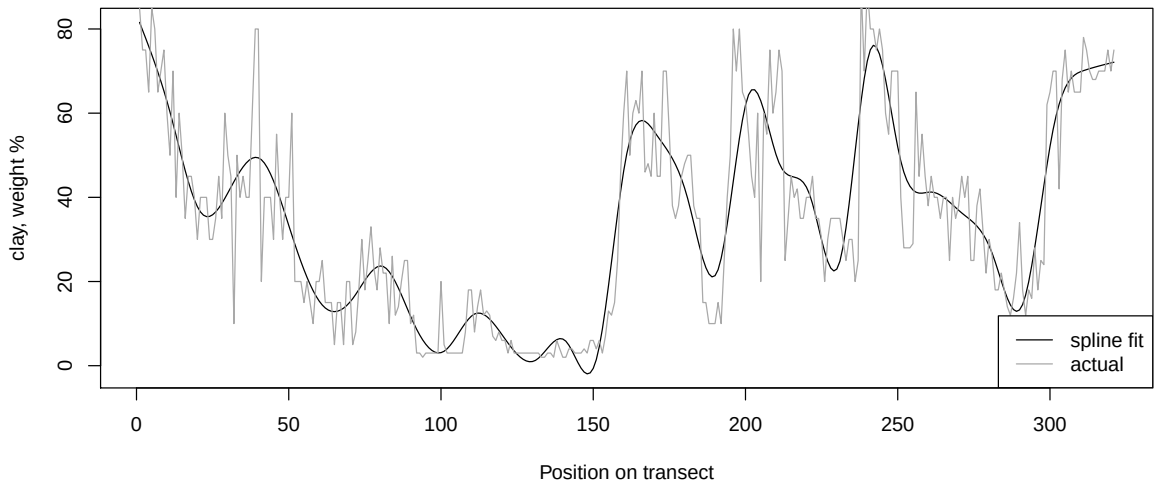
```
> m.10 <- lm(ds$clay2 ~ basis.10)
> summary(m.10)$adj.r.squared

[1] 0.800346
```

The computed model also includes the fitted values at each known point, which we extract with the `fitted` generic method:

```
> summary(pred.10 <- fitted(m.10))

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 -1.945  15.030  35.860  34.610  49.390  81.570

> summary(ds$clay2)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   2.00   15.00   34.00   34.61   50.00   90.00
```

TASK 9 : Plot the original station data, with the values from the fitted splines. •

```
> plot(pred.10, type="l",ylab="clay, weight %", xlab="Position on transect")
> lines(ds$clay2, col="darkgray")
> legend("bottomright", lty=1,
+        col=c("black","darkgray"),
+        c("spline fit","actual"))
```



We note several features of this fit:

- The fitted values have the same mean, similar medians and quartiles. However, the extrema are quite different: the actual maximum is not reached, and an artefact of the polynomial fit produces a non-physical (negative) value for the minimum.
- The degree of smoothness seems to track the local trends fairly well.

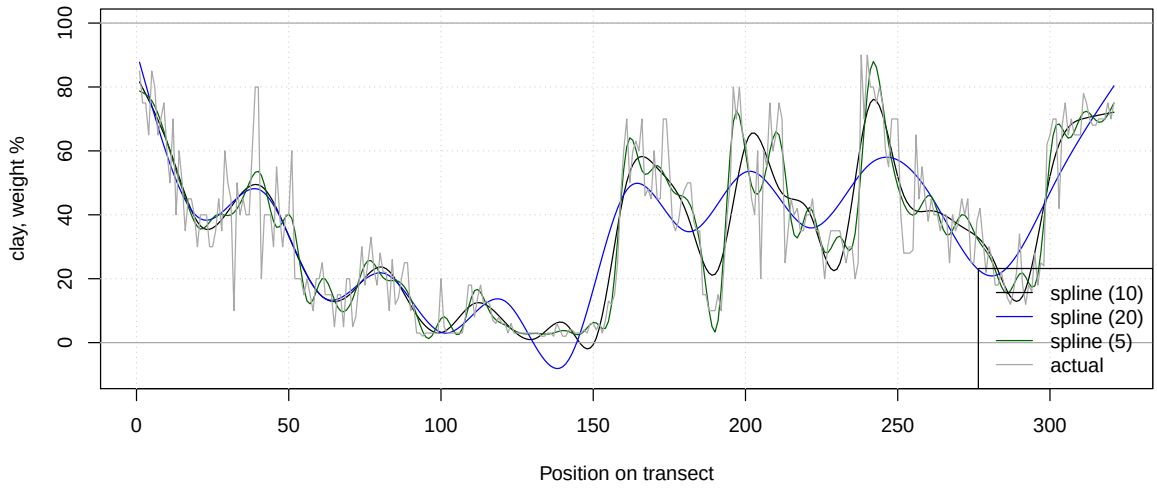
TASK 10 : Compare this with fits with half and double the number of knots. •

```
> basis.20 <- ns(ds$seq, df=floor(n/20))
> basis.5 <- ns(ds$seq, df=floor(n/5))
> m.20 <- lm(ds$clay2 ~ basis.20)
> summary(m.20)$adj.r.squared

[1] 0.6955464

> m.5 <- lm(ds$clay2 ~ basis.5)
> summary(m.5)$adj.r.squared

[1] 0.8528901
```

We can see that the more knots, the closer to the known values; but this does not necessarily mean a better fit to the process that produced the data – if the process is noisy, a spline that does not fit so well may better represent it.

TASK 11 : Show the three spline fits (dividing the transect into 20, 10 and 5-station increments). •

```
> plot(pred.10, type="l", ylab="clay, weight %", xlab="Position on transect",
+      ylim=c(-10,100))
> grid()
> abline(h=0, col="darkgray"); abline(h=100, col="darkgray")
> lines(predict(m.20), col="blue")
> lines(predict(m.5), col="darkgreen")
> lines(ds$clay2, col="darkgray")
> legend("bottomright", lty=1, col=c("black","blue","darkgreen","darkgray"),
+        c("spline (10)","spline (20)", "spline (5)", "actual"))
```



The graph shows that using too few knots can lead to over-smoothing and even an un-physical result (outside the possible range of the variable); here the negative values near station 140 when the transect is split into 20-station increments.

Note that the splines are all linear at the boundaries: this is a property of natural splines.

If a prediction at other than the observations is wanted, we use function `interpSpline` to determine the coefficients of the spline from the data, and then the generic `predict` function, which specialized to the invisible `predict.bSpline` prediction function.

> **Note:** The function `predict.lm` which predicts from the fitted linear model can not be used, because the predictors are the basis function values computed at the original stations; there is no data value anywhere else.

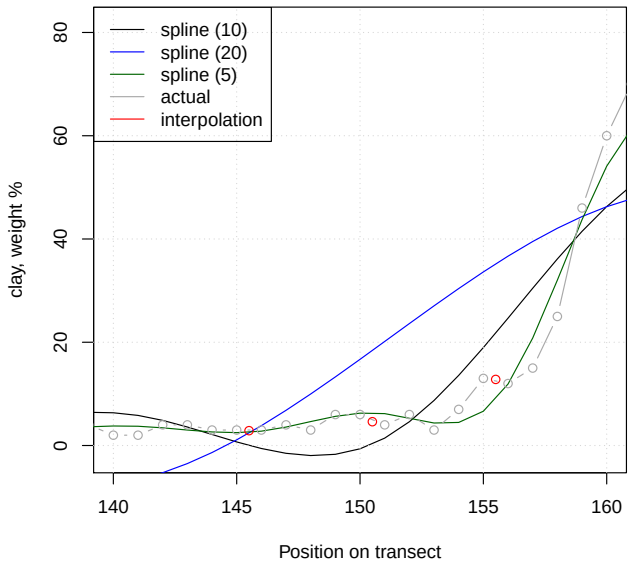
TASK 12 : Predict the value half-way between stations 145 and 146, 150 and 151, and 150 and 160. •

```
> pts <- c(145.5,150.5,155.5)
> str(ispl <- interpSpline(ds$seq, ds$clay2, bSpline=TRUE))

List of 3
```

```
 $ knots       : num [1:327] -2 -1 0 1 2 3 4 5 6 7 ...
 $ coefficients: num [1:323] 99.1 85 70.9 81.3 53.8 ...
 $ order       : num 4
 - attr(*, "formula")=Class 'formula' length 3 ds$clay2 ~ ds$seq
  .. ..- attr(*, ".Environment")=<environment: 0x7fdd5a426ed0>
 - attr(*, "class")= chr [1:3] "nbSpline" "bSpline" "spline"

> (pred3 <- predict(ispl, x=pts)$y)

[1]  2.875619  4.611803 12.803852

> ds$clay2[145:156]

 [1]  3  3  4  3  6  6  4  6  3  7 13 12
```

Display this part of the spline curve, original data, and predicted value.

```
> plot(pred.10, type="l", ylab="clay, weight %",
+      xlab="Position on transect", xlim=c(140,160))
> grid()
> lines(predict(m.20), col="blue", type="l")
> lines(predict(m.5), col="darkgreen", type="l")
> lines(ds$clay2, col="darkgray", type="b")
> legend("topleft", lty=1, col=c("black","blue","darkgreen","darkgray","red"),
+        c("spline (10)","spline (20)", "spline (5)", "actual", "interpolation"))
> points(x=pts, y=pred3, col="red")
> rm(pred3)
```



Evidently the interpolation spline closely matches the immediately surrounding values.

2.2 Smoothing splines

One problem with the natural spline approach of the previous § is that the analyst must decide on the number of knots and their placement, using prior knowledge on the presumed smoothness of the relation. An alterative approach is **smoothing** splines. These make a compromise

between fit and degree of smoothness; this matches the typical situation when it is not necessary to match the exact values at the knots, but rather to make a smooth curve that (we hope) represents the local trends without "too much" noise. The optional degree of smoothness can be estimated by **generalized cross-validation**[19].

The solution minimizes the general formula:

$$\min_f \sum_{i=1}^{N} \{y_i - f(x_i)\}^2 + \lambda J[f] \tag{2}$$

where J is the penalty function and λ controls how important it is; $\lambda = 0$ means there is no roughness penalty and the data will be fit exactly; as $\lambda \to \omega$ the solution approximates the least-squares line, i.e., the trend line averaged over all the points.

In 1D an appropriate penalty is:

$$J[f] = \int_{\mathbb{R}} \{f''(t)\}^2 \, dt \tag{3}$$

I.e., the integral of the squared second derivative over the interval; so the more curves, the higher the penalty. In practice the integral is discretized over the knots.

TASK 13 : Fit a smoothing spline to the clay content. •

Smooth splines can be fit by the `smooth.spline` function of the `stats` package that is loaded with base R. We start with the default smooth spline. It is possible to specify the degree of smoothing with the `spar` smoothing parameter or the `df` degrees of freedom argument. If these are missing, leave-one-out cross-validation is used to determine the optimal λ; we choose this option.

```
> (spl.fit <- smooth.spline(ds$clay2))

Call:
smooth.spline(x = ds$clay2)

Smoothing Parameter  spar= 0.2544574  lambda= 1.204002e-07 (12 iterations)
Equivalent Degrees of Freedom (Df): 73.46845
Penalized Criterion: 18226.75
GCV: 95.48895

> round(length(ds$seq)/spl.fit$df,1)

[1] 4.4
```

Among the returned parameters are:

spar : the smoothing parameter, usually in $(0 \ldots 1]$;

lambda : the value of λ of Equation 2 corresponding to `spar`;

degrees of freedom : equivalent degrees of freedom used in the fit;

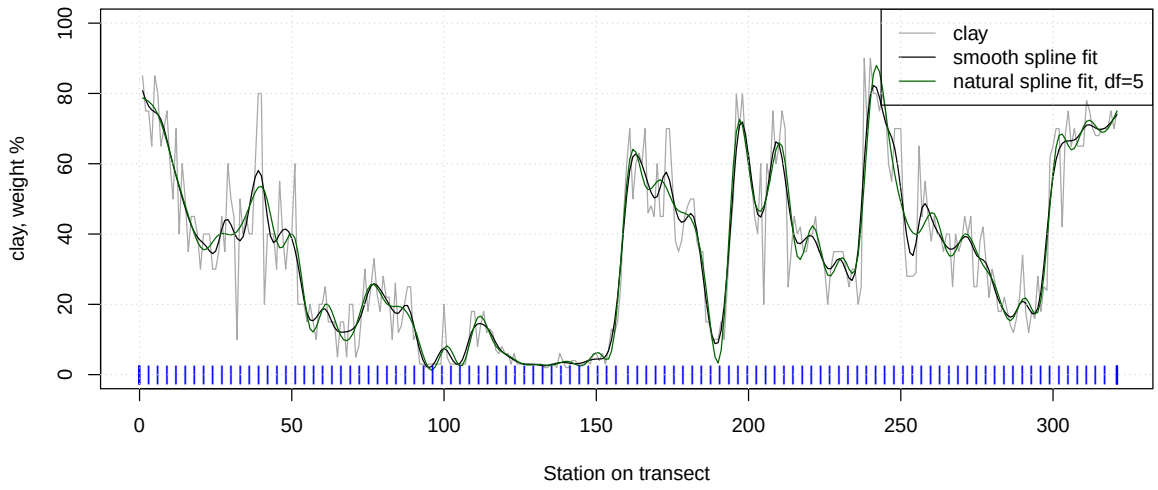
y : vector of fitted values

Here we see that about 73.5 degrees of freedom were used. This is equivalent to a non-smooth fit with $320/73.5 = 4.4$ stations between knots, similar to the non-smooth fit with 5 stations between knots[3], i.e., the most knots among the three we compared in §2.1.

TASK 14 : Plot the original data and the spline fit. •

We also plot the knot positions as small blue bars.

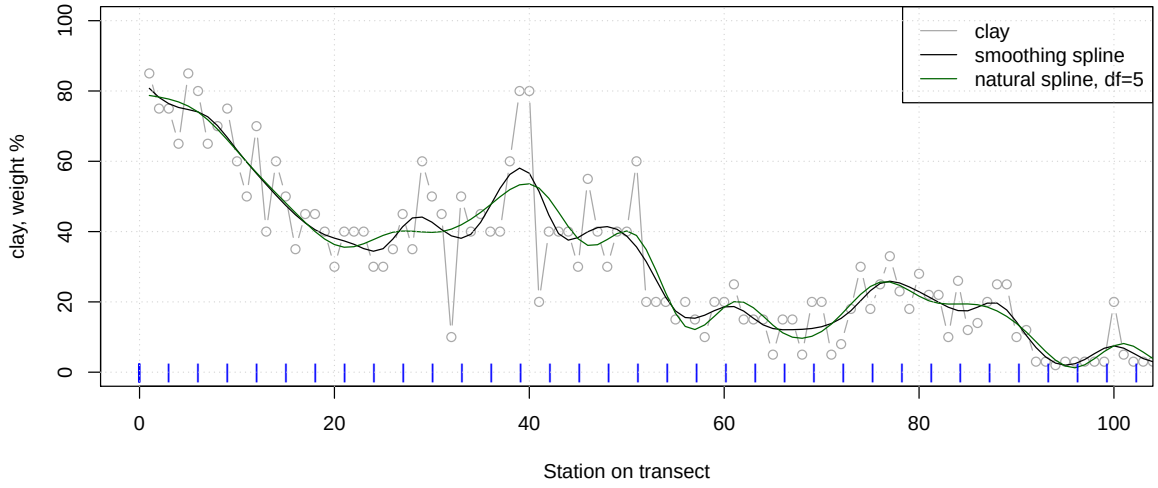
```
> plot(ds$clay2, type="l", col="darkgray", ylim=c(0,100), xlim=c(0,320),
+      xlab="Station on transect", ylab="clay, weight %")
> lines(spl.fit$y)
> lines(predict(m.5), col="darkgreen", type="l")
> points(spl.fit$fit$knot*length(ds$clay2), rep(0, length(spl.fit$fit$knot)),
+        spl.fit$nk, pch="|", col="blue")
> legend("topright", c("clay","smooth spline fit", "natural spline fit, df=5"),
+        lty=1, col=c("darkgray","black", "darkgreen"))
> grid()
```



TASK 15 : Repeat the plot, zooming in on stations 0 to 100. •

[3] Confusingly, this number is referred to as the degrees of freedom in the non-smooth fit.

```
> plot(ds$clay2, type="b", col="darkgray", ylim=c(0,100), xlim=c(0,100), xlab="Station on transect", ylab="clay,
> lines(spl.fit$y)
> lines(predict(m.5), col="darkgreen")
> points(spl.fit$fit$knot*length(ds$clay2), rep(0, length(spl.fit$fit$knot)),
+        spl.fit$nk, pch="|", col="blue")
> legend("topright", c("clay","smoothing spline", "natural spline, df=5"),
+        lty=1, col=c("darkgray","black", "darkgreen"))
> grid()
```



In this case, the penalty function was optimized with a somewhat closer fit to the observations than the natural spline with 5 degrees of freedom. The figure shows a slightly closer match to the observations, according to the penalty function without excessive curvature.

3 Thin-plate splines

3.1 Theory

Hastie et al. [9, §5.7] explains the mathematics of multi-dimensional smoothing splines. A more thorough mathematical treatment is given by Wood [23] and Mitasova and Mitas [16]; these are developments from the "minimum curvature" methods of Briggs [2]. Applications include Hutchinson [10] and Mitasova and Hofierka [15].

The method we use to fit the surface is known as a **thin plate spline** (TPS); the analogy is with a thin (so, flexible) plate that is warped to fit the data. But this can range from very "rigid", i.e., just a single surface (the usual least-squares plane of a first-order trend surface) to very "flexible", i.e., perfectly fitting every observation. In general we want something in between: if we think there is an overall surface we just fit it as one polynomial (first, second ... order polynomials on the coördinates), but if we want to fit more locally, we must expect local noise which should be somehow locally averaged-out.

Fitting a TPS depends on the k data points with known coördinates and attribute values. They can be described by $2(k+3)$ parameters, six of which are overall affine transformation parameters (to center the function in 2D) and $2k$ of which link to the control points.

The general method is to minimize the residual sum of squares (RSS) of the fitted function, subject to a constraint that the function be "smooth" in some sense; this is expressed by a **roughness penalty** which balances the fit to the observations with smoothness. This is a minimization problem. If $\mathbf{x_i}$ is one point in 2D space (i.e., it has two coördinates) and y_i is the attribute value at the same points, the aim is to minimize:

$$\min_f \sum_{i=1}^{N} \{y_i - f(\mathbf{x_i})\}^2 + \lambda J[f] \tag{4}$$

where J is the penalty function and λ controls how important it is; $\lambda = 0$ means there is no roughness penalty and the data will be fit exactly; as $\lambda \to \omega$ the solution approximates the least-squares plane, i.e., the trend surface averaged over all the points.

In 2D an appropriate penalty is:

$$J[f] = \int_{\mathbb{R}} \int_{\mathbb{R}} \left[\left(\frac{\partial^2 f(\mathbf{x})}{\partial x_1^2} \right)^2 + 2 \left(\frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_2} \right)^2 + \left(\frac{\partial^2 f(\mathbf{x})}{\partial x_2^2} \right)^2 \right] \mathrm{d}x_1 \mathrm{d}x_2 \tag{5}$$

where (x_1, x_2) are the two coördinates of the vector $\mathbf{x}$. This is the same approach as taken for the 1D smoothing splines, above (§2.2), but now considering second derivatives in two dimensions as well as the second cross-derivative.

In practice the double integral is discretized over some grid known as **knots**; these may be defined by the observations or may be a different set, maybe an evenly-spaced grid.

This penalty can be interpreted as the "bending energy" of a thin plate represented by the function $f(x)$; by minimizing this energy the spline function in over the 2D plane is a thin (flexible) plate which, according to the first term of Equation 4 would be forced to pass through data points, with minimum bending. However the second term of Equation 4 allows some smoothing: the plate does not have to bend so much, since it is allowed to pass "close to" but not necessarily through the data points. The higher the λ, the less exact is the fit. This has two purposes: (1) it allows for measurement error; the data points are not taken as exact; (2) it results in a smoother surface.

Of course, the question is, how to balance the smoothness and the penalty. In the case of kriging the degree of smoothing away from known points is determined by the variogram, especially the proportion of the variance due to the nugget. For TPS there is no model, so there is no internal estimate of the desired degree of smoothing. So cross-validation is used to determine the degree of smoothness.

The solution to Equation 5 is a linear function:

$$f(\mathbf{x}) = \beta_0 + \beta^T \mathbf{x} + \sum_{j=1}^{N} \alpha_j h_j(\mathbf{x}) \tag{6}$$

where the β account for the overall trend and the α are the coefficients of the warping.

The set of functions $h_j(\mathbf{x})$ is the **basis kernel**, also called a **radial basis function** (RBF), for thin-plate splines:

$$h_j(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}_j\|^2 \log \|\mathbf{x} - \mathbf{x}_j\| \tag{7}$$

where the norm distance $r = \|\mathbf{x} - \mathbf{x}_j\|$ is also called the **radius** of the basis function. The norm is usually the Euclidean (straight-line) distance.

3.2 The `fields` R package

The `fields` package is a "collection of programs written in the R language for curve and function fitting with an emphasis on spatial data" [17]. It was developed at the National Center for Atmospheric Research (USA). It includes methods for kriging, sampling design, and visualization; we will use the `Tps` "thin plate splines" method for surface interpolation[4].

TASK 16 : Load the `fields` package. •

```
> require(fields)

Loading required package:  fields

Loading required package:  spam

Loading required package:  grid

Spam version 1.3-0 (2015-10-24) is loaded.
Type 'help( Spam)' or 'demo( spam)' for a short introduction
and overview of this package.
Help for individual functions is also obtained by adding the
suffix '.spam' to the function name, e.g.  'help( chol.spam)'.


Attaching package:  'spam'

The following objects are masked from 'package:base':

    backsolve, forwardsolve

Loading required package:  maps

Warning:  package 'maps' was built under R version 3.2.3


# maps v3.1:  updated 'world':  all lakes moved to separate new #
# 'lakes' database.  Type '?world' or 'news(package="maps")'.  #
```

[4] Recall that R names are case-sensitive; this is `Tps` not `tps`.

3.3 Fitting thin-plate splines

We use the "Jura soils" dataset: a set of soil samples from the Swiss Jura, supplied by J.-P. Dubois of the Ecole Polytechnique Fèdérale in Lausanne (CH). This dataset is used as a running example in the well-known text of Goovaerts [7]; there are also many research papers [1, 6, 8, 12, 13, 21, 25] that use it as a test case for geostatistical methods. It is included as an example in the `gstat` package.

TASK 17 : Load the Jura dataset •

```
> require(gstat)

Loading required package:  gstat

Warning:  package 'gstat' was built under R version 3.2.3

> data(jura)
> ls(pattern="jura")

[1] "jura.grid"    "jura.pred"    "jura.val"
[4] "juragrid.dat"
```

The Jura dataset contains several objects; we will work with the "calibration" observations `jura.pred` and an interpolation surface `jura.grid`. The objective is to map one of the heavy metal concentrations over the grid, based on the observations.

```
> str(jura.pred)

'data.frame': 259 obs. of  13 variables:
 $ Xloc   : num  2.39 2.54 2.81 4.31 4.38 ...
 $ Yloc   : num  3.08 1.97 3.35 1.93 1.08 ...
 $ long   : num  6.85 6.85 6.86 6.88 6.88 ...
 $ lat    : num  47.1 47.1 47.1 47.1 47.1 ...
 $ Landuse: Factor w/ 4 levels "Forest","Pasture",..: 3 2 2 3 3 3 3 3 3 3 ...
 $ Rock   : Factor w/ 5 levels "Argovian","Kimmeridgian",..: 3 2 3 2 5 5 5 1 1 3 ...
 $ Cd     : num  1.74 1.33 1.61 2.15 1.56 ...
 $ Co     : num  9.32 10 10.6 11.92 16.32 ...
 $ Cr     : num  38.3 40.2 47 43.5 38.5 ...
 $ Cu     : num  25.72 24.76 8.88 22.7 34.32 ...
 $ Ni     : num  21.3 29.7 21.4 29.7 26.2 ...
 $ Pb     : num  77.4 77.9 30.8 56.4 66.4 ...
 $ Zn     : num  92.6 73.6 64.8 90 88.4 ...

> str(jura.grid)

'data.frame': 5957 obs. of  6 variables:
 $ Xloc   : num  0.3 0.35 0.35 0.4 0.4 0.4 0.4 0.4 0.4 0.4 ...
 $ Yloc   : num  1.7 1.7 1.75 1.7 1.75 1.8 1.85 1.9 2.1 2.15 ...
 $ long   : num  6.82 6.82 6.82 6.82 6.82 ...
 $ lat    : num  47.1 47.1 47.1 47.1 47.1 ...
 $ Landuse: Factor w/ 4 levels "Forest","Pasture",..: 2 2 2 2 1 1 1 1 3 3 ...
 $ Rock   : Factor w/ 5 levels "Argovian","Kimmeridgian",..: 3 3 3 2 2 2 3 3 1 1 ...
```

The `fields` package does not use `sp` classes, nor does it work directly on dataframes with fields representing the coördinates. It requires two arguments: a matrix of coördinates (rows are observation locations, columns are the two coördinates) and a vector of dependent variables (the attribute to map). The second we already have as data fields of the `jura.pred` dataframe. So we must make a matrix from the two coörd-

inate fields.

TASK 18 : Reformat the coördinate of `jura.pred` into the form expected by the `fields` package. •

The coördinates are formatted as a matrix with the `matrix` function; the single vector of coördinate built from the separate vectors for X and Y are organized by column (i.e., first the column for X, then for Y), thus we specify the `byrow=FALSE` argument[5] and specify that there are two columns:

```
> jura.pred$coords <- matrix(c(jura.pred$Xloc, jura.pred$Yloc), byrow=F, ncol=2)
> str(jura.pred$coords)

 num [1:259, 1:2] 2.39 2.54 2.81 4.31 4.38 ...
```

TASK 19 : Fit the default thin-plate spline. •

The `Tps` function[6] fits a thin-plate spline to observations. There are many parameters but we start with the defaults:

```
> surf.1 <-Tps(jura.pred$coords, jura.pred$Co)
> class(surf.1)

[1] "Krig" "Tps"
```

The class of the returned object, Krig, Tps, is `Krig`, defined by `fields`.

```
> summary(surf.1)

CALL:
Tps(x = jura.pred$coords, Y = jura.pred$Co)

 Number of Observations:                259
 Number of unique points:               259
 Number of parameters in the null space 3
 Parameters for fixed spatial drift     3
 Effective degrees of freedom:          156.1
 Residual degrees of freedom:           102.9
 MLE sigma                              0.9095
 GCV sigma                              1.095
 MLE rho                                68350
 Scale passed for covariance (rho)      <NA>
 Scale passed for nugget (sigma^2)      <NA>
 Smoothing parameter lambda             1.21e-05

Residual Summary:
      min     1st Q    median     3rd Q       max
-3.151000 -0.266200 -0.006246  0.201500  2.901000

Covariance Model: Rad.cov
  Names of non-default covariance arguments:
       p

DETAILS ON SMOOTHING PARAMETER:
 Method used:   GCV    Cost:  1
   lambda       trA       GCV   GCV.one GCV.model      shat
1.210e-05 1.561e+02 3.016e+00 3.016e+00        NA 1.095e+00

 Summary of all estimates found for lambda
```

[5] this is the default, but we show it here explicitly
[6] note the "T" is not "t"!

```
              lambda    trA   GCV  shat -lnLike Prof
GCV         0.0000121 156.13 3.016 1.095        599.9
GCV.model          NA     NA    NA    NA           NA
GCV.one     0.0000121 156.13 3.016 1.095           NA
RMSE               NA     NA    NA    NA           NA
pure error         NA     NA    NA    NA           NA
REML        0.0001309  95.17 3.479 1.483        576.4
            converge
GCV                2
GCV.model         NA
GCV.one            2
RMSE              NA
pure error        NA
REML               4
```

There is a lot to see here. A very important parameter is the smoothing parameter λ:

```
> print(surf.1$lambda)

[1] 1.210237e-05
```

This can be specified by the analyst, but if omitted it is determined by **generalized cross-validation** (GCV) [19].

The `fields` package solves the thin-plate spline as a special case of kriging, using the `Krig` function and specifying that the spatial covariance is a **radial basis function** (RBF), i.e., a real-valued function whose value depends only on the distance from some center, e.g., a given point. A RBF can have any form as long as the only argument is $||r||$, where r is the radius from the origin. For thin-plate splines, a single RBF is used, $\phi(r) = r^2 \log(r)$. This can be proven to be the solution to Equation 6, as expressed in Equation 7.

Task 20 : Predict over the grid with the fitted splines. •

First the prediction locations must be formatted as a matrix, then the `predict.Krig` function can be called to do the prediction. This function is called automatically by the generic `predict` method for objects of class `Krig`, including the output from `Tps`.

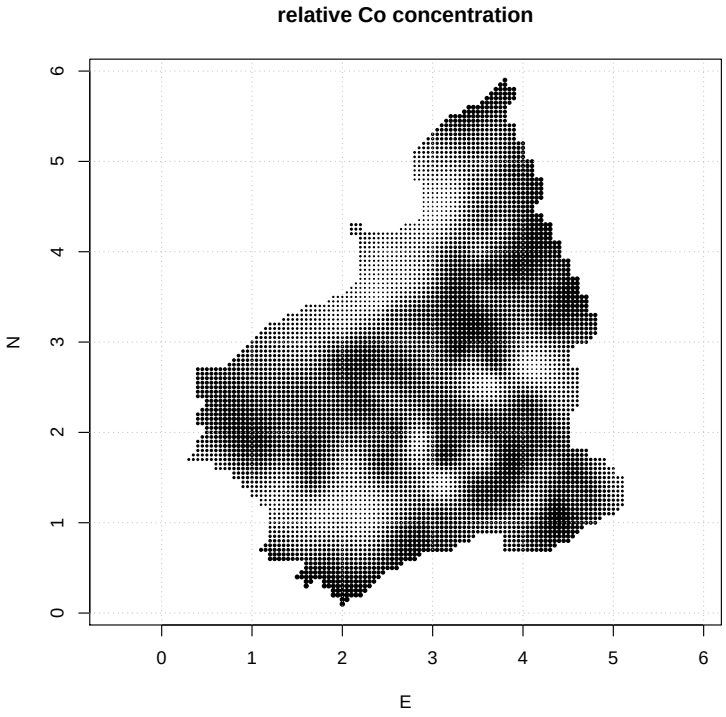
```
> jura.grid$coords <- matrix(c(jura.grid$Xloc, jura.grid$Yloc), byrow=F, ncol=2)
> surf.1.pred <- predict.Krig(surf.1, jura.grid$coords)
> summary(as.vector(surf.1.pred))

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.094   7.585  10.090   9.553  11.700  16.890
```

Task 21 : Show the fitted surface as 2.5D plot. •

We use the `plot` function, with the coördinates as axes, and show the value by the size of the symbol:

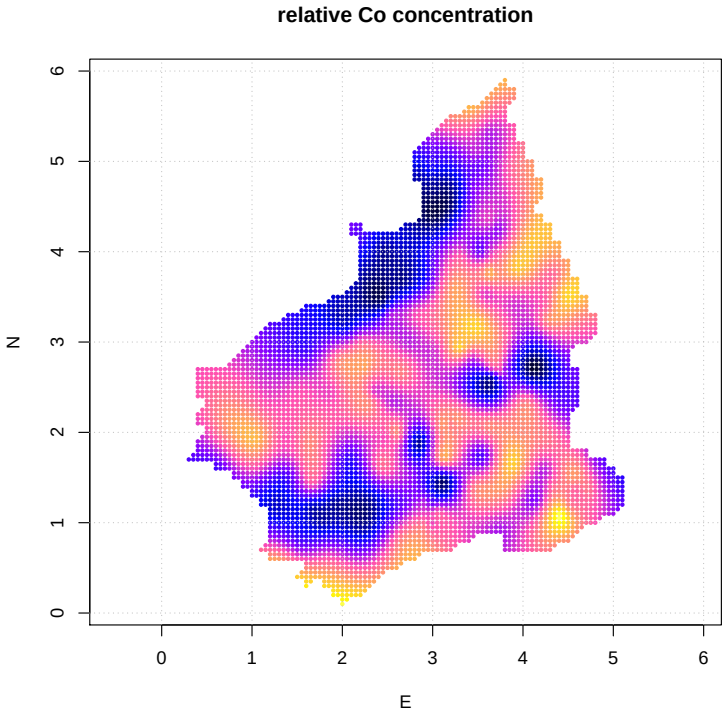
```
> surf.1.pred.m <- matrix(surf.1.pred,
+                         length(jura.grid$coords[,1]),
+                         length(jura.grid$coords[,2]),
+                         byrow=F)
> str(surf.1.pred.m)
```

```
 num [1:5957, 1:5957] 6.95 6.98 7.33 7.04 7.43 ...

> plot(jura.grid$coords, pch=20, asp=1,
+      cex=0.8*surf.1.pred/max(surf.1.pred),
+      xlab="E", ylab="N",
+      main="relative Co concentration")
> grid(col="gray")
```



We can also use colours for the third dimension; the `cut` function divides the predictions into equal-size ranges and codes them by the range number into which they fall: We use the `bpy.colors` function of the `sp` package to specify a blue-purple-yellow colour ramp.

```
> require(sp)

Loading required package:  sp

> plot(jura.grid$coords, pch=20, asp=1, cex=.6,
+      col=bpy.colors(256)[cut(surf.1.pred, 256)],
+      xlab="E", ylab="N",
+      main="relative Co concentration")
> grid(col="gray")
```



3.4 Evaluation

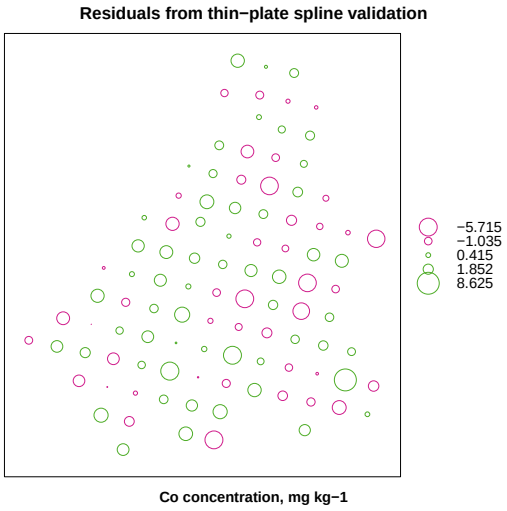
We can evaluate ("validate") the spline approach with an independent dataset, here `jura.val`. Note that the fit was optimized by cross-validation with the calibration dataset, and we have an estimate of generalized cross-validation error. But this is an *internal* validation; if an independent dataset is used the validation is *external*.

TASK 22 : Predict at the validation points and compute the validation statistics. •

We summarize the results in a `SpatialPointsDataFrame` provided by the `sp` package.

```
> jura.val$coords <-
+     matrix(c(jura.val$Xloc, jura.val$Yloc), byrow=F, ncol=2)
> surf.1.val <-
+     predict.Krig(surf.1, jura.val$coords)
> surf.1.val.res <-
+     (jura.val$Co - surf.1.val)
> summary(as.vector(surf.1.val.res))

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-5.7150 -1.0350  0.4150  0.3621  1.8520  8.6250

> require(sp)
> surf.1.val.res.sp <-
+     SpatialPointsDataFrame(coords = jura.val$coords,
+                            data=data.frame(res=surf.1.val.res))
> (rmse.tps <- with(surf.1.val.res.sp@data, sum(res^2)/length(res)))

[1] 6.331696
```

```
> rmse.tps/median(jura.val$Co)

[1] 0.6293932

> (me.tps <- with(surf.1.val.res.sp@data, sum(res)/length(res)))

[1] 0.3620904

> bubble(surf.1.val.res.sp, zcol="res", pch=1,
+        main="Residuals from thin-plate spline validation",
+        sub="Co concentration, mg kg-1")
```



There is a slight negative bias (mean of the actual observations at the evaluation locations is somewhat higher than that of the predictions) and a high RMSE, compared to the median value of the target variable. There does not seem to be any spatial pattern to the residuals – it's just that the thin-plate spline does not approach some very high or low values.

4 Comparison with a geostatistical approach

A natural question is how this interpolation compares to a geostatistical approach, e.g., Ordinary Kriging (OK).

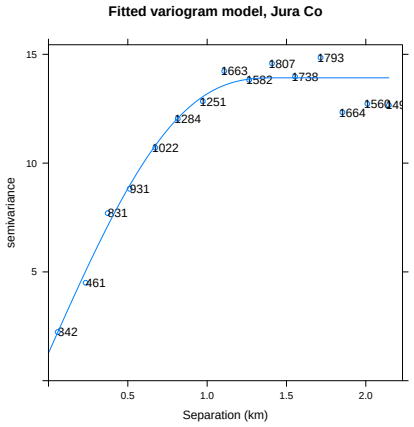
TASK 23 : Compare to OK interpolation. •

OK depends on a model of spatial autocovariance, so we must first select and parameterize an authorized variogram model of the spatial structure of the calibration points. This can then be used to krige at the validation points. We use `gstat` functions for geostatistical analysis; this package works with objects of class `SpatialPointsDataFrame`, so we copy the calibration points and convert with the `coordinates` function of the `sp` package:

```
> jura.pred.sp <- jura.pred
> coordinates(jura.pred.sp) <- ~Xloc + Yloc
> v <- variogram(Co ~ 1, loc=jura.pred.sp)
> (vmf <- fit.variogram(v, vgm(12.5, "Pen", 1.2, 1.5)))
```

```
  model     psill    range
1   Nug  1.269763 0.000000
2   Pen 12.649771 1.449021
```

```
> plot(v, plot.numbers=T, model=vmf,
+      main="Fitted variogram model, Jura Co",
+      xlab="Separation (km)")
```



We use this model to krige at the validation points:

```
> jura.val.sp <- jura.val
> coordinates(jura.val.sp) <- ~Xloc + Yloc
> k <- krige(Co ~ 1, loc=jura.pred.sp, newdata=jura.val.sp, model=vmf)

[using ordinary kriging]
```

We compare the summary statistics with the spline prediction at these points. First the predictions:

```
> summary(k$var1.pred)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.554   7.698   9.978   9.468  11.280  14.090

> summary(as.vector(surf.1.val))

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.851   7.396   9.896   9.431  11.640  14.570

> summary(k$var1.pred - as.vector(surf.1.val))

    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-1.69800 -0.34730  0.06987  0.03719  0.51050  1.52100
```

The OK prediction has a narrower overall and inter-quartile range; that is, in this case OK smooths more than splines.

Now the residuals:

```
> summary(k$res <- (jura.val.sp$Co - k$var1.pred))

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-5.5910 -1.2230  0.4242  0.3249  1.7330  8.9510

> summary(as.vector(surf.1.val.res))

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-5.7150 -1.0350  0.4150  0.3621  1.8520  8.6250

> summary(k$res - as.vector(surf.1.val.res))
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-1.52100 -0.51050 -0.06987 -0.03719  0.34730  1.69800
```

We see that the spline residuals have a less extreme maximum but more extreme minimum.

And finally the validation RMSE and ME:

```
> (rmse.k <- with(k@data, sum(res^2)/length(res)))

[1] 6.130507

> print(rmse.tps)

[1] 6.331696

> (me.k <- with(k@data, sum(res)/length(res)))

[1] 0.3249007

> print(me.tps)

[1] 0.3620904
```

In this case OK performs slightly better, probably due to the strong local spatial structure which can be captured by a reliable variogram model. The wider IQR and range produced by the TPS were not justified by the validation data.

To visually compare the predicted fields, we also krige over the grid and compare the two predictions side-by-side. However first we need to convert them to the same data structure.

TASK 24 : Predict over the interpolation grid by OK. •

```
> jura.grid.sp <- jura.grid
> coordinates(jura.grid.sp) <- ~Xloc + Yloc
> gridded(jura.grid.sp) <- TRUE
> k <- krige(Co ~ 1, loc=jura.pred.sp, newdata=jura.grid.sp, model=vmf)

[using ordinary kriging]
```

TASK 25 : Convert the TPS interpolation to a `SpatialPixelsDataFrame`. •

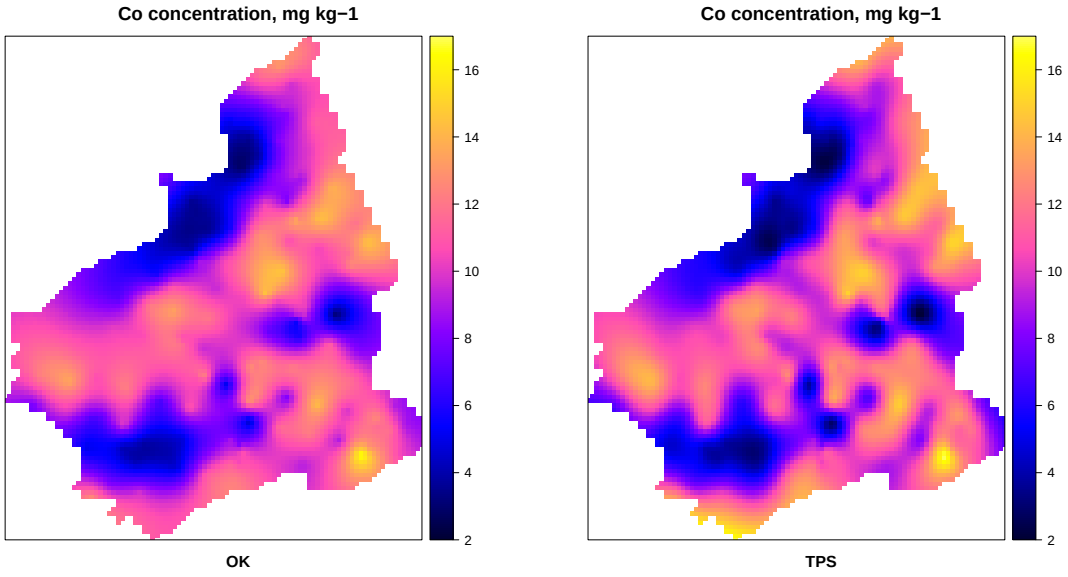
We use the existing `SpatialPixelsDataFrame` just produced by kriging, and add the TPS prediction to it as a data field. Note the structure of the data vector is identical, since both predictions were over the same grid.

```
> class(k)

[1] "SpatialPixelsDataFrame"
attr(,"package")
[1] "sp"

> k$tps.pred <- surf.1.pred
> str(k)

Formal class 'SpatialPixelsDataFrame' [package "sp"] with 7 slots
  ..@ data       :'data.frame': 5957 obs. of  3 variables:
  .. ..$ var1.pred: num [1:5957] 8.88 8.8 9.04 8.71 8.99 ...
```

```
  .. ..$ var1.var : num [1:5957] 9.7 8.84 8.87 7.89 7.96 ...
  .. ..$ tps.pred : num [1:5957, 1] 6.95 6.98 7.33 7.04 7.43 ...
  ..@ coords.nrs : int [1:2] 1 2
  ..@ grid       :Formal class 'GridTopology' [package "sp"] with 3 slots
  .. .. ..@ cellcentre.offset: Named num [1:2] 0.3 0.1
  .. .. .. ..- attr(*, "names")= chr [1:2] "Xloc" "Yloc"
  .. .. ..@ cellsize         : Named num [1:2] 0.05 0.05
  .. .. .. ..- attr(*, "names")= chr [1:2] "Xloc" "Yloc"
  .. .. ..@ cells.dim        : Named int [1:2] 97 117
  .. .. .. ..- attr(*, "names")= chr [1:2] "Xloc" "Yloc"
  ..@ grid.index : int [1:5957] 8149 8150 8053 8151 8054 7957 7860 7763 7375 7278 ...
  ..@ coords     : num [1:5957, 1:2] 0.3 0.35 0.35 0.4 0.4 0.4 0.4 0.4 0.4 0.4 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:5957] "1" "2" "3" "4" ...
  .. .. ..$ : chr [1:2] "Xloc" "Yloc"
  ..@ bbox       : num [1:2, 1:2] 0.3 0.1 5.1 5.9
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "Xloc" "Yloc"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
  .. .. ..@ projargs: chr NA
```

TASK 26 : Display the two interpolations side-by-side. •

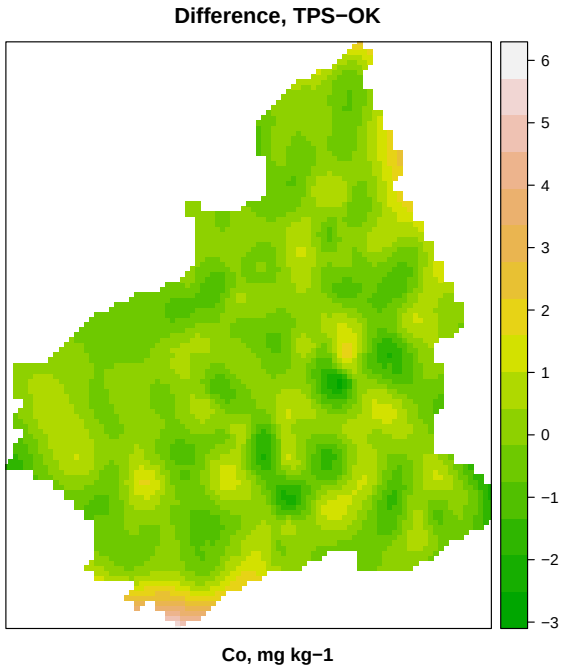
```
> zlim <- c(floor(min(k$var1.pred, k$tps.pred)),
+     ceiling(max(k$var1.pred, k$tps.pred)))
> ats <- seq(zlim[1], zlim[2], length=256)
> spplot(k, zcol="var1.pred", col.regions=bpy.colors(256), at=ats,
+        main="Co concentration, mg kg-1", sub="OK")
> spplot(k, zcol="tps.pred", col.regions=bpy.colors(256), at=ats,
+        main="Co concentration, mg kg-1", sub="TPS")
```



TASK 27 : Display the differences between the two predictions. •

```
> summary(k$diff <- k$tps.pred - k$var1.pred)

       V1
 Min.   :-2.525298
 1st Qu.:-0.417421
```

```
 Median :-0.031525
 Mean   : 0.009275
 3rd Qu.: 0.376290
 Max.   : 5.719727

> spplot(k, zcol="diff", col.regions=terrain.colors(64),
+        main="Difference, TPS-OK", sub="Co, mg kg-1")
```



The spline predicts considerably higher values in the extreme south and to a lesser extent along the NE edge. It predicts somewhat lower at some "basins" in the interior.

5 Considerations for using thin-plate spline interpolation

A few studies [e.g. 4, 24] have compared TPS or similar radial basis function methods to kriging.

An early analysis of splines vs. kriging is from Dubrule:

> "Spline interpolation is equivalent to kriging with fixed covariance and degree of polynomial trend. Usually kriging is performed in two steps, a structural analysis, which fits a covariance and a degree of trend to the variable under study, then the interpolation itself, which uses the results of the structural analysis. With spline interpolation, no preliminary structural analysis is performed. The covariance and degree of trend do not depend on the variable under study. This should result in a loss of accuracy of splines compared to kriging."
>
> ...
>
> "[S]plines and kriging are two methods which should be used

> alternately, depending on what one wants to obtain. If the map is going to be used for future calculations, one needs accuracy; kriging is good in this situation. If one wants to quickly obtain a clear map showing the main features of the variable, splines are a good tool." [4]
>
> "Kriging, by minimizing the estimation variance, is designed to provide estimates which are as close as possible to the actual values. Spline interpolation, by minimizing the total curvature, is designed to provide maps which have nice cosmetic properties." [5]

Smoothing splines do not take into account the spatial structure of the target variable; they just use its values to fit the radial basis functions. If the roughness penalty λ is optimized by cross-validation, the spatial structure of the target variable is implicitly considered but not explicitly modelled.

Since no model is needed, one could consider an advantage of smoothing splines that there is no need to model regional trends or local spatial structure; and further than this structure can vary across the field. Thus there is no assumption of either first- or second-order stationarity, and no need to model. This may be especially attractive if there are not enough observations to reliably estimate a variogram. However, estimating the optimal roughness penalty is also problematic with few observations.

> **Note:** Another approach to relaxing stationarity assumption is moving-window kriging [20] as implemented by the VESPER computer program [14]; however this requires a dense network of observations and is best-suited to precision agriculture.

A clear disadvantage of splines is that we can not interpret a model of spatial structure (whether a trend surface or local autocorrelation), we can only predict. Such a model is often interesting to reveal the spatial process which is presumed to have produced the observations; for example in ecology or soil science this can suggest hypotheses of processes over the landscape.

Further, there is no internal measure of prediction error, equivalent to the kriging prediction variance; this is because there is no knowledge of the target variable's spatial structure. So spline predictions can only be assessed from an independent dataset or (somewhat) by cross-validation.

In summary, splines have a place in the analyst's toolkit for a fast method to visualize the main features of a spatial dataset (1D or 2D) and to make spatial predictions, but they are not suitable for spatial analysis.

References

[1] O Atteia, J-P Dubois, and R Webster. Geostatistical analysis of soil contamination in the Swiss Jura. *Environmental Pollution*, 86(3): 315–327, 1994. 15

[2] I. C. Briggs. Machine contouring using minimum curvature. *Geophysics*, 39(1):39–48, January 1974. ISSN 0016-8033, 1942-2156. doi: 10.1190/1.1440410. 12

[3] J. C. Davis. *Statistics and data analysis in geology*. John Wiley & Sons, New York, 3rd edition, 2002. 2

[4] O. Dubrule. Comparing splines and kriging. *Computers and Geosciences*, 10(2-3):328–338, 1984. 1, 24, 25

[5] O. Dubrule. Reply: comparing splines and kriging. *Computers & Geosciences*, 12(5):729–730, 1986. doi: 10.1016/0098-3004(86)90051-8. 1, 25

[6] X Emery and J M Ortiz. Weighted sample variograms as a tool to better assess the spatial variability of soil properties. *Geoderma*, 140:81–89, 2007. 15

[7] P Goovaerts. *Geostatistics for natural resources evaluation*. Applied Geostatistics. Oxford University Press, New York; Oxford, 1997. 15

[8] P Goovaerts, R Webster, and J-P Dubois. Assessing the risk of soil contamination in the Swiss Jura using indicator geostatistics. *Environmental and Ecological Statistics*, 4(1):31–48, 1997. 15

[9] T Hastie, R Tibshirani, and J H Friedman. *The elements of statistical learning data mining, inference, and prediction*. Springer series in statistics. Springer, New York, 2nd ed edition, 2009. ISBN 9780387848587. 1, 2, 12

[10] M. F. Hutchinson. Interpolating mean rainfall using thin plate smoothing splines. *International Journal of Geographical Information Science*, 9(4):385 – 403, 1995. 12

[11] K. Krivoruchko. *Spatial statistical data analysis for GIS users: DVD containing book + data*. ESRI, Redlands, 2011. 1

[12] R M Lark. Modelling complex soil properties as contaminated regionalized variables. *Geoderma*, 106(3-4):173–190, 2002. 15

[13] B P Marchant and R M Lark. Robust estimation of the variogram by residual maximum likelihood. *Geoderma*, 140:62–72, 2007. 15

[14] B. Minasny, A. B. McBratney, and B. M. Whelan. VESPER: variogram estimation and spatial prediction plus ERror - australian centre for precision agriculture - the university of sydney, 2005. URL `http://sydney.edu.au/agriculture/pal/software/vesper.shtml`. 25

[15] H. Mitasova and J. Hofierka. Interpolation by regularized spline with tension: II. Application to terrain modeling and surface geometry

analysis. *Mathematical Geology*, 25(6):657–669, 1993. doi: 10.1007/BF00893172. 12

[16] H. Mitasova and L. Mitas. Interpolation by regularized spline with tension: I. Theory and implementation. *Mathematical Geology*, 25 (6):641–655, 1993. doi: 10.1007/BF00893171. 12

[17] Fields Development Team. `fields`: Tools for spatial data, 2006. URL `http://www.cgd.ucar.edu/Software/Fields`. 14

[18] William N. Venables and Brian D. Ripley. *Modern Applied Statistics with {S}. Fourth Edition*. Springer, 2002. 1

[19] G. Wahba. *Spline models for observational data*. Society for Industrial and Applied Mathematics, 1990. 10, 17

[20] C. Walter, A. B. McBratney, A. Douaoui, and B. Minasny. Spatial prediction of topsoil salinity in the Chelif Valley, Algeria, using local ordinary kriging with local variograms versus whole-area variogram. *Australian Journal of Soil Research*, 39(2):259–272, 2001. 25

[21] R Webster, O Atteia, and J-P Dubois. Coregionalization of trace metals in the soil in the Swiss Jura. *European Journal of Soil Science*, 45 (2):205–218, 1994. 15

[22] R.C Webster and H. E. de la C. Cuanalo. Soil transect correlograms of north oxfordshire and their interpolation. *Journal of Soil Science*, 26:176–194, 1975. 2

[23] S. N. Wood. Thin plate regression splines. *Journal of the Royal Statistical Society Series B-Statistical Methodology*, 65:95–114, 2003. doi: 10.1111/1467-9868.00374. 12

[24] Yunfeng Xie, Tong-bin Chen, Mei Lei, Jun Yang, Qing-jun Guo, Bo Song, and Xiao-yong Zhou. Spatial distribution of soil heavy metal pollution estimated by different interpolation methods: Accuracy and uncertainty analysis. *Chemosphere*, 82(3):468–476, January 2011. doi: 10.1016/j.chemosphere.2010.09.053. 24

[25] T Yao. Nonparametric cross-covariance modeling as exemplified by soil heavy metal concentrations from the Swiss Jura. *Geoderma*, 88 (1-2):13–38, 1999. 15

Index of R Concepts

`bpy.colors` (`sp` package), 18

`coordinates` (`sp` package), 20
`cut`, 18

`df` argument (`ns` function), 4
`df` argument (`smooth.spline` function), 10

`fields` package, 14–17
`fitted`, 6

`gstat` package, 15, 20

`interpSpline` (`splines` package), 8

`Krig` (`fields` package), 17
`Krig` class, 16, 17

`lm`, 5

`matrix`, 16

`ns` (`splines` package), 4

`plot`, 17
`predict`, 8, 17
`predict.bSpline` (`splines` package), 8
`predict.Krig` (`fields` package), 17
`predict.lm`, 8

`read.table`, 3

`skip` function argument, 3
`smooth.spline`, 10
`sp` package, 15, 18–20
`spar` argument (`smooth.spline` function), 10
`SpatialPixelsDataFrame` class, 22
`SpatialPointsDataFrame` class, 19, 20
`splines` package, 4
`stats` package, 10

`Tps` (`fields` package), 14, 16, 17