Contents

Thin-Plate Splines

Background

A thin-plate spline is a particular type of spatial-smoothing and interpolation model for data in two-dimensions[1]. The general idea is to fit the data using a "smooth" surface that does not have a well-defined form *a priori* (i.e. it is not necessarily, for example, a plane or a paraboloid) but instead the surface roughly follows a localized averaging of the data. More formally, the data is modeled by the function $y = f(x) + \epsilon$ where x is a point in a two-dimensional plane, y is the outcome variable of interest and f is the prediction function represented by this surface (let $\hat{f}$ denote the fitted value of f given a set of training data).

The first thing to note here is that this is a non-parametric model. In our vague description above, since the surface interpolates, or 'smooths' or 'averages' the training data points, to calculate $\hat{f}(x)$ for any particular value of x is to take some kind of weighted average of all the values of y in the training data. In other words, for a training data set of size n, evaluating $\hat{f}(x)$ is an $\mathcal{O}(n)$ operation. Furthermore, as we will see, there is an additional set of weights associated with $\hat{f}$ (one for each data point in the training data) that must be computed using an $n \times n$ matrix, which must be inverted, et cetera. These computational nuances are important to bear in mind.

There is also a hyperparameter involved, typically denoted as λ. In theory, if x is continuously valued and no two points are identical, then the function could be made to interpolate exactly at every point, although in practice this would lead to an overfit and unhelpful prediction function[2]. The opposite situation is with full-smoothing, which is to say a surface with no curvature, which is equivalent to a linear model (i.e. a plane). These

[1] A solid background on smoothing splines is useful to understand this section if the reader does not already have that. A good start to this can be found in Hastie, Tibshirani and Friedman, *The Elements of Statistical Learning*, sections 5.2-4.
[2] Again, see Hastie et al. for some graphics on this. A google search of "smoothing splines" is likely to also yield a graphic illustrating the effect of λ.

two situations correspond to $\lambda = 0$ and $\lambda = \infty$, respectively; the ideal is a happy medium, and there is rich theory on the topic of choosing the ideal value of λ.

The sections below cut to the chase on what this model looks like and how it is computed. The first is a generic thin-plate spline, and the second is a semi-parametric model that joins several individual splines with shared linear parameters to solve them all at once.

Usage in Bias Calculations

The final model presented in this document lays out a model that involves multiple, separate, semi-parametric thin-plate splines with a set of shared model parameters that are solved simultaneously. In that model, the shared data field and parameters across the different splines are denoted respectively by Z and $\boldsymbol{\beta}$. For our purpose, those shared fields and parameters are the station and satellite bias values. Following the model set by Tyler, the version of this model that has been implemented populates the data field as a dummy variable representing the PRN and another dummy variable representing the station. Rather than set those to 1, they are scaled to the value of $D/K \approx 9.5177539$. The nuances of the model also require multiplying it by the slant-to-vertical factor at that particular observation, though that will be demonstrated below.

Previous Model: Linear Model for "Coincidence" Events

The previous model represented a sensible approach to estimating bias: it searched through the data for points in time where multiple satellite-station connections had a thin-shell pierce point in approximately the same location at the same time. Hypothetically, in those instances they *should* be observing the same vTEC estimate, and the differences can be used to solve for the biases. Namely, the model was specified like this. For a particular observation i, the observed sTEC value $sTEC_i$ is equal to a hypothetical "correct" vTEC value β_{c_i} (where c_i indexes the "coincidence" event within which multiple observations have the same hypothetical vTEC), adjusted for the angle of the connection, plus the respective bias values of the satellite and receiver, times the conventional coefficient:

$$\frac{D}{K}(\delta_r - \delta_s) + \frac{1}{S2V_i}\beta_{c_i} = sTEC_i$$

Tyler's model additionally follows the convention that the bias for the satellite is a negative value, so we keep that.

In that model, a vector of coefficients for δ_r, δ_s, and β_c is solved for by linear regression. Also, note the factor $\frac{1}{S2V}$ in front of the hypothetical correct vTEC value, which adjusts it to be comparable to sTEC. In our model that will be moved slightly.

New Model: Simultaneous Thin-Plate Splines with Shared Parameters

In the spline model, the model is similar. One difference is that the outcome variable is specified as being vTEC rather than sTEC. So namely, for a particular observation i, which takes place at location x_i (a 2-vector of coordinates) at a particular time t, the observed value $vTEC_i$ is equal to the actual value plus the biases:

$$vTEC_i = f_t(x) + \frac{D}{K}(\delta_r - \delta_s) * S2V_i$$

Here, the function f_t is the prediction function from the thin-plate spline. It amounts to a smooth function that maps the ionosphere over the domain of the x-values in the training data, and it outputs vTEC. Notably, since the ionosphere varies over time, t here represents a point or set of points in time that are close enough together to have constant vTEC values (in the model jargon, t represents a "layer" of the model). As discussed more later, as a practical matter the time point t actually corresponds to two (or more) points in time, one from which to pull the "knot" data points, and one for some extra training data points that are "un-knotted", usually separated by a short period of time on the order of 5-15 minutes. The gap should ideally be large enough that

corresponding locations (i.e. for the same satellite-station pair) have some distance between them, but close enough that we can safely assume the ionosphere is constant. Although they are pulled from different ticks, they are considered to be from the same time point as it indexes individual ionospheric projections.

Since not all satellites are visible throughout the day, a representative set of times must be chosen (e.g. every 1-2 hours for a full day), such that every station and satellite combination is represented (that is not a hard requirement, but is both ideal and easy to achieve). Each of those times represents a layer of the model because it will have its own spatial prediction function $\widehat{f}_t$ that is fitted on the data from that time. In that sense, the training data is partitioned into subsets, one for each layer of the model. Those subsets are further partitioned into knotted and un-knotted points, which have technical implications for the model solver.

The final model described here shows how this set of layered splines with the shared coefficients is set up and solved. In that case, the vector $\boldsymbol{\beta}$ represents the joint set of (δ_r, δ_s) for all receivers and stations in the data. The data matrix Z is set to be $\pm\frac{D}{K} * S2V_i$ at each observation for the columns corresponding to (r, s). The rest is simple math :-)

Generic Thin-Plate Spline Model

Notation

Let:

- y_i be the outcome variable at the location x_i for data points $i = 1, \dots, n$.
  - Here $x_i = \left(1, x_i^{(1)}, x_i^{(2)}\right)$ is a point in $\mathbb{R}^2$ represented in homogenous coordinates, to simplify adding a constant later.
  - Let Y, X denote the full set of these, respectively, in column-vector form.
- n be the sample size
- $f(x)$ be the prediction function, so $E[y] = f(x)$.
- $k(r) = r^2 \log r$ is a radial-basis **kernel function**.
  - Alternately denote $k(x, x_0) = k(\|x - x_0\|)$ for shorthand where $\|x - x_0\|$ is the Euclidean distance between the two points.
  - $\underset{(1\times n)}{K(x)} = [k(x_1, x) \quad \dots \quad k(x_n, x)]$ is the row vector of the kernel function applied to each data point and a new point x.
  - $\underset{n\times n}{K} = \left[k\left(x_i, x_j\right)\right]_{ij}$ be the **kernel matrix** on the data points
- λ is the smoothing parameter.
- $\mathbf{0}$ represents an appropriately sized vector or matrix of zeros.

Model

The model that follows is a combination of a standard linear model and a model representing the deviation from the linear model, which is the non-parametric part. Here the coefficients are represented by $\boldsymbol{a}$ and $\boldsymbol{w}$, which are size 3 and n, respectively.

Prediction Function:

$$\begin{aligned} f(x) &= a_0 + a_1 x^{(1)} + a_2 x^{(2)} + \sum_{i=1}^{n} w_i k(\|x_i - x\|) \\ &= x\boldsymbol{a} + K(x)\boldsymbol{w} \end{aligned}$$

Loss Function: Penalized Squared Error

$$L(Y, X, w, a) = (Y - X\boldsymbol{a} - K\boldsymbol{w})^T (Y - X\boldsymbol{a} - K\boldsymbol{w}) + n\lambda \boldsymbol{w}^T K \boldsymbol{w}$$

$$\Longrightarrow \boldsymbol{w}, \boldsymbol{a} = \operatorname{argmin} L(Y, X)$$

Solution:

- For f to be differentiable we must have that:

$$\sum_{i=1}^{p} w_i = \sum_{i=1}^{p} w_i x_i^{(1)} = \sum_{i=1}^{p} w_i x_i^{(2)} = 0$$

$$\Longrightarrow X^T \boldsymbol{w} = \boldsymbol{0}$$

- This gives us a linear form of the system of equations with shape $\big((n+3) \times (n+3)\big)$

$$\underbrace{\begin{bmatrix} K + n\lambda I_n & X \\ X^T & \boldsymbol{0} \end{bmatrix}}_{:=L} \begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{a} \end{bmatrix} = \begin{bmatrix} Y \\ \boldsymbol{0} \end{bmatrix}, \qquad \Rightarrow \begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{a} \end{bmatrix} = L^{-1} \begin{bmatrix} Y \\ \boldsymbol{0} \end{bmatrix}$$

Slightly More Complicated Version: Semi-Parametric

Before getting into this version of the model, it is worth discussing the previous. First, note that in the form above, the solution is exactly determined: $n + 3$ equations in $n + 3$ variables. Now let's imagine that there is an additional covariate vector z (of length p) and we want to include a linear parameter for it in the model. Then the prediction function becomes:

$$f(x) = a_0 + a_1 x^{(1)} + a_2 x^{(2)} + \sum_{i=1}^{n} w_i k(\|x_i - x\|) + \beta_1 z_1 + \cdots + \beta_p z_p = x\boldsymbol{a} + K(x)\boldsymbol{w} + z\boldsymbol{\beta}$$

This form of the model is suddenly underdetermined. In order to correct this, we have to add a number of extra points to the training data which are *not* included in the points used to fit $\hat{f}$. The term for this is that the original n points are called "knot" points, while the additional points are un-knotted.

Notation

- Denote the number of knotted points as n_K and the number of un-knotted points as n_U, with a total of $n = n_K + n_U$. Furthermore, the knotted data points are indexed from $i = 1, \dots, n_K$ while the un-knotted data points continue from there and are index from $i = n_K + 1, \dots, n$.
- Let K_K denote the kernel matrix as in the previous model, on the knotted points. Here we will also have $K_U = \big[k(x_i, x_j)\big]_{ij}$ where $i = 1, \dots, n_K$ but $j = n_K + 1, \dots, n$. In other words, K_U is the stacked row matrix of $K(x_i)$ for all the un-knotted data points x_i.
- Let Z denote the full set of data points z_i as a set of stacked row vectors. So Z is an $(n \times p)$ matrix.
- Let X_K, Y_K, Z_K denote the respective sub-matrices for just the knotted points, and likewise with the subscript U for the unknotted points.

Model

Now the system of equations looks slightly different. In this case, unless we have exactly as many extra data points as we do new parameters (i.e. $n_U = p$), this matrix will be over-determined and we will have to solve it by least squares:

$$\underbrace{\begin{bmatrix} K_K + n\lambda I_n & X_K & Z_K \\ K_U & X_U & Z_U \\ X^T & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix}}_{:=L} \begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{a} \\ \boldsymbol{\beta} \end{bmatrix} = \begin{bmatrix} Y_K \\ Y_U \\ \boldsymbol{0} \end{bmatrix}, \qquad \Rightarrow \begin{bmatrix} \widehat{\boldsymbol{w}} \\ \boldsymbol{a} \\ \boldsymbol{\beta} \end{bmatrix} = (L^T L)^{-1} L^T \begin{bmatrix} Y_K \\ Y_U \\ \boldsymbol{0} \end{bmatrix}$$

In this way, the addition of the un-knotted data points has allowed us to solve for the parameters in $\boldsymbol{\beta}$ along with the thin-plate spline.

Even More Complicated Version: Interlacing of Multiple Splines

For this model, we are going to extend the previous model even more. Now we are going to assume there are multiple different sets of data points, with each set belonging to its own "surface" but sharing a set of parameters across all of them. This is functionally equivalent to using a single surface, but adding the specification that the kernel function evaluated at two points in different sets is defined to be zero (additionally, the coefficients $\boldsymbol{\alpha}$ are not shared across sets). This imparts a block-diagonal structure on L, which keeps the matrix inversion computation from exploding.

Notation

Assume there are m groups of data points, each with n_j data points in it for $j = 1, \dots, m$. Let the sample size be $n = n_1 + \cdots + n_m$. Additionally, assume that for each group there are $n_j' < n_j$ points that will be used as knot points, with the remainder not considered knot points. For convenience, let $n_j'' = n_j - n_j'$ and assume that points $1, \dots, n_j'$ are the knot points, with points $n_j' + 1, \dots, n_j$ excluded as knots. Also for convenience denote $n' = n_K = n_1' + \cdots + n_j'$ and likewise $n'' = n_U$ for un-knotted points.

- y_{ji} be the outcome variable in group j, data point i, at the location x_{ji} for data points $i = 1, \dots, n_j$.
  - Here $x_{ji} = \left(1, x_{ji}^{(1)}, x_{ji}^{(2)}\right)$ is a point in $\mathbb{R}^2$ represented in homogenous coordinates, to simplify adding a constant later.
  - Let Y_j, X_j denote the column vector for the full set of these for group j. Let Y be the stacked column-matrix of Y_j.
- Assume there are p covariates $z^{(1)}, \dots, z^{(p)}$ and that the subscripting above applies to each $z^{(\cdot)}$. Let $z = [z^{(1)} \quad \dots \quad z^{(p)}]$ denote the row vector of these covariates for a given data point. Let Z_j denote the stacked row vectors for all data points in group j, and let Z denote the matrix of stacked Z_j for $j = 1, \dots, m$.
- $f(x, z)$ be the prediction function, so $E[y] = f(x, z)$.
- $k(r) = r^2 \log r$ is the **kernel function**
  - Alternately denote $k(x, x_0) = k(\|x - x_0\|)$ for shorthand where $\|x - x_0\|$ is the Euclidean distance between the two points.
  - $K_j(x) = \underset{(1 \times n_j')}{\left[k\left(x_{j1}, x\right) \quad \dots \quad k\left(x_{jn_j'}, x\right)\right]}$ is the row vector of the kernel function applied to each data point in group j and a new point x. Let K_j'' be the stacked column matrix of $K_j(x)$ over the un-knotted data points in group j, i.e.:
  - $\underset{n_j' \times n_j'}{K_j'} = \left[k\left(x_{ji}, x_{jk}\right)\right]_{ik}$ be the **kernel matrix** on the data points in group j, $i, k = 1, \dots, n_j'$. $\underset{n_j'' \times n_j'}{K_j''} = \left[K_j\left(x_{j\left(n_j'+k\right)}\right)\right]_k$, for $k = 1, \dots, n_j''$.
  - Essentially K' is the symmetric kernel matrix of the knotted points evaluated over the knotted points and K'' is the matrix of the knotted points evaluated over the un-knotted points.
- λ is the smoothing parameter.
- $\mathbf{0}$ represents an appropriately sized vector or matrix of zeros.

Prediction Function:

Let $x_0 = \left(x_{j0}, z_0\right)$ denote a new data point assumed to belong to group j:

$$f(x_0) = a_{0_j} + a_{x_j} x_{j0}^{(1)} + a_{y_j} x_{j0}^{(2)} + \sum_{k=1}^{p} \beta_k z_{ji}^{(k)} + \sum_{i=1}^{n_j} w_{ji} k\left(\left\|x_{ji} - x_{j0}\right\|\right)$$
$$= x\boldsymbol{a}_j + z\boldsymbol{\beta} + K_j(x_0)\boldsymbol{w_j}$$

Let $\boldsymbol{a} = \left(\boldsymbol{a}_1, \dots, \boldsymbol{a}_j\right)^T$ and likewise with $\boldsymbol{w}$. Additionally, let:

$$\underbrace{K}_{(n\times n_K)} = \begin{bmatrix} K_1' + n_1\lambda I_{n_1} & & \mathbf{0} \\ K_1'' & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & K_j' + n_j\lambda I_{n_j} \\ \mathbf{0} & & K_j'' \end{bmatrix}, \quad \underset{(n\times 3m)}{X} = \begin{bmatrix} X_1 & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & X_j \end{bmatrix}, \underset{(n\times p)}{Z} = \begin{bmatrix} Z_1 \\ \vdots \\ Z_j \end{bmatrix}$$

In the above, each X_i includes both the knotted and un-knotted points. Below we will let X_i' be the same but excluding the un-knotted points.

The differentiability constraint from the simpler model holds only on each group of x-values. That is, for all $j = 1, \dots, m$ it must hold that $X_j^T \boldsymbol{w}_j = \mathbf{0}$. Thus we can rewrite the system of linear equations as:

$$\underbrace{\begin{bmatrix} K & X & Z \\ X'^T & \mathbf{0} & \mathbf{0} \end{bmatrix}}_{\substack{(n_{(.)}+3m)\times(n'+3m+p) \\ =L}} \begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{a} \\ \boldsymbol{\beta} \end{bmatrix} = \begin{bmatrix} Y \\ \mathbf{0} \end{bmatrix}$$

The same differentiability constrain on f does not hold for the Z values. Since this system is overdetermined we can solve it by least squares:

$$\begin{bmatrix} \widehat{\boldsymbol{w}} \\ \widehat{\boldsymbol{a}} \\ \widehat{\boldsymbol{\beta}} \end{bmatrix} = (L^T L)^{-1} L^T \begin{bmatrix} Y \\ \mathbf{0} \end{bmatrix}$$

So now we have to focus on the problem of how to invert $L^T L$, which for hundreds of stations and multiple layers can have rows *and* column sizes in the tens to hundreds of thousands:

$$L^T L = \begin{bmatrix} K^T & X' \\ X^T & 0 \\ Z^T & 0 \end{bmatrix} \begin{bmatrix} K & X & Z \\ X'^T & \mathbf{0} & \mathbf{0} \end{bmatrix} = \begin{bmatrix} \underset{(n'\times n')}{K^T K + X'X'^T} & \underset{(n'\times 3m)}{K^T X} & \underset{(n'\times p)}{K^T Z} \\ \underset{(3m\times n')}{X^T K} & \underset{(3m\times 3m)}{X^T X} & \underset{(3m\times p)}{X^T Z} \\ \underset{(p\times n')}{Z^T K} & \underset{(p\times 3m)}{Z^T X} & \underset{(p\times p)}{Z^T Z} \end{bmatrix}$$

To solve, we can treat this as a block matrix and invert it that way. Specifically, the top-left block is block-diagonal, which makes matrix-inversion tractable for a large data set. More prosaically, the matrix inversion can be done almost separately for each spline layer of the model. Specifically, we will break $L^T L$ into a block matrix in the following matter. Let $L^T L = \begin{bmatrix} A & B \\ B^T & D \end{bmatrix}$ where:

- $A = K^T K + X'X'^T$
  - Both terms are block diagonal. So we can compute $K_j^T K + X_j' X_j'^T$ and its inverse one at a time.
- $B = K^T [X \quad Z] = C^T$

- $D = \begin{bmatrix} X^T \\ Z^T \end{bmatrix} [X \quad Z]$

In that way we can use the following generic formula to invert a block-matrix:

$$(L^T L)^{-1} = \begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1} & -A^{-1}B(D - CA^{-1}B)^{-1} \\ -(D - CA^{-1}B)^{-1}CA^{-1} & (D - CA^{-1}B)^{-1} \end{bmatrix}$$
$$= \begin{bmatrix} A & B \\ B^T & D \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} + A^{-1}B(D - B^T A^{-1}B)^{-1}B^T A^{-1} & -A^{-1}B(D - B^T A^{-1}B)^{-1} \\ -(D - B^T A^{-1}B)^{-1}B^T A^{-1} & (D - B^T A^{-1}B)^{-1} \end{bmatrix}$$

This solution is implemented in the module `pytid.utils.thin_plate_spline`.