Code 01a

This code loads the PM 2.5 constituent data set.

(1) **y**: the $n \times P$ matrix that has all the data. The different columns correspond to the different observed items, in the order of sulfur, ammonium ion, nitrate, sulfate.

(2) **u**, **v**, **t**: are $n \times 1$ vectors of longitude, latitude, and time of the observations. All the observations were copied from **'pm25_spec_working_data4'**.

(3) **Ny**: number of observed items, P.

(4) **N**: number of observations, n.

(5) **l**: the $P \times 1$ vector of factor loadings that excludes the functional part, $[\lambda_{1,0}, \ldots, \lambda_{P,0}]^T$.

(6) **s**: the $P \times 1$ vector of the variance of random errors, $[\sigma_1^2, \ldots, \sigma_P^2]^T$.

The values of l, s store here are to be used as initial values.

Code 01b

In this code we create the spline basis matrix for the grid points of (u,v), t.

(1) **uv_grid**, **t_grid**: are the grid points of (u, v), t used for plotting the estimated functions. In this documentation for Code 01b the use of (u,v), t refers to the grid points. The grid points of (u,v) were imported from **'uv_bound_a'**. The grid points of t were equally spaced from 0.05 to 1.95 by 0.01.

(2) **knot_u**, **knot_v**, **knot_t**: the knot vectors of u, v, t used in bspline() to generate the spline basis matrices. These vectors were created in 'Analysis 3 Code 01b'. They have equally spaced inner knots within the range of u, v, t, and also have repeated boundary points as the outer knots. The number of outer knots are equal to the degree of the spline. The number of inner knots for u, v, t are 1, 1, 6, respectively. The number of marginal spline coefficients for u, v, t are 5, 5, 10, respectively (represented as K_u, K_v, K_t in this documentation).

The reason why I use explicit knots is because the grid points and the actual data somehow do not have the same bounds. In order to let the two sets match in terms of using the same spline coefficients, I have to use the same knots. Otherwise SAS will decide where the knots are based on the range of the vector.

(3) **z_uv**, **z_t**: are $\mathbf{Z}_{uv}$, $\mathbf{Z}_t$ used to impose the linear constraints on the spline coefficients.

First we construct $\mathbf{X}_u$, $\mathbf{X}_v$, $\mathbf{X}_t$ as the spline basis matrices for u, v, t (denoted by 'c_u', 'c_v', 'c_t' in the code). The tensor product matrix for (u,v) is $(\mathbf{X}_u \otimes \mathbf{J}_{K_v}^T) \circ (\mathbf{J}_{K_u}^T \otimes \mathbf{X}_v)$, that is, each row of $\mathbf{X}_{uv}$ is the Kronecker product of the rows of $\mathbf{X}_u$ and $\mathbf{X}_v$.

We set the centering constraints for $\boldsymbol{\beta}_{uv}$, $\boldsymbol{\beta}_t$ as $\frac{1}{n}\mathbf{1}_n^T\mathbf{X}_{uv}\boldsymbol{\beta}_{uv} = 0$, $\frac{1}{n}\mathbf{1}_n^T\mathbf{X}_t\boldsymbol{\beta}_t = 0$. It guarantees the mean of the function values f_{uv}, f_t, of all the grid points are zero. We then create $\mathbf{Z}_{uv}$, $\mathbf{Z}_t$ using 'center()'.

(4) **c_uv_grid**, **c_t_grid**: are $\mathbf{X}_{uv}\mathbf{Z}_{uv}$ and $\mathbf{X}_t\mathbf{Z}_t$, respectively.

Code 01b2

This code creates the spline basis matrices for (u, v), t from the actual data.

(1) **c_uv_z**, **c_t_z**: are $\mathbf{X}_{uv}\mathbf{Z}_{uv}$ and $\mathbf{X}_t\mathbf{Z}_t$, respectively. The procedure is the same as in Code 01b for the grid points of (u,v), t. 'c_uv_z' and 'c_t_z' are $\mathbf{X}_{uv}\mathbf{Z}_{uv}$

and $\mathbf{X}_t\mathbf{Z}_t$, respectively. Note that $\mathbf{Z}_{uv}$, $\mathbf{Z}_t$ were created in Code 01b, based on the grid points. **Here $\mathbf{X}_{uv}$, $\mathbf{X}_t$ are the basis matrices of (u, v), t from the actual data set, different than in Code 01b2, even though the same notation is used.**

(2) **c_z**: is $\mathbf{X} = [\mathbf{J}_n, \mathbf{X}_{uv}\mathbf{Z}_{uv}, \mathbf{X}_t\mathbf{Z}_t]$. **Notice the notational difference here. In the manuscript, $\mathbf{X}_{uv}$, $\mathbf{X}_t$ are already transformed with the centering constraints.**

(3) **nb_uv**, **nb_t**, **nb**: the number of columns in 'c_uv_z', 'c_t_z', 'c_z'.

(4) **B_uv_z**, **B_vu_z**, **B_t_z**: the 'half' penalty matrices, $\mathbf{B}_{u|v}\mathbf{Z}_{uv}$, $\mathbf{B}_{v|u}\mathbf{Z}_{uv}$, $\mathbf{B}_t\mathbf{Z}_t$.

First we have the 'half' penalty matrices, $\mathbf{B}_u$, $\mathbf{B}_v$, $\mathbf{B}_t$, created from 'penalty()' (denoted by 'B_u', 'B_v', 'B_t' in the code). In this documentation, a 'half' penalty matrix, $\mathbf{B}$, means it satisfies $\mathbf{B}^T\mathbf{B} = \mathbf{S}$, the penalty matrix.

$\mathbf{X}_{\tilde{u}}$, $\mathbf{X}_{\tilde{v}}$ (denoted by 'c_u2', 'c_v2' in the code) are the basis matrices of $\tilde{u}$, $\tilde{v}$, the vectors of grid points used for integrating the univariate penalties of $\boldsymbol{\beta}_{p,u}$, $\boldsymbol{\beta}_{p,v}$ when penalizing $\boldsymbol{\beta}_{p,uv}$. We first look at one part of the composite penalty for $\boldsymbol{\beta}_{p,uv}$, $\boldsymbol{\beta}_{p,uv}^T\mathbf{S}_{u|v}\boldsymbol{\beta}_{p,uv}$. Since the penalty matrix $\mathbf{S}_{u|v} = \mathbf{S}_u \otimes (\mathbf{X}_{\tilde{v}}^T\mathbf{X}_{\tilde{v}})$ and $\mathbf{S}_u = \mathbf{B}_u^T\mathbf{B}_u$, $\mathbf{S}_{u|v} = (\mathbf{B}_u^T \otimes \mathbf{X}_{\tilde{v}}^T)(\mathbf{B}_u \otimes \mathbf{X}_{\tilde{v}})$. So the 'half' penalty matrix of $\mathbf{S}_{u|v}$ is $\mathbf{B}_u \otimes \mathbf{X}_{\tilde{v}}$. Similarly, the 'half' penalty matrix of $\mathbf{S}_{v|u}$ is $\mathbf{X}_{\tilde{u}} \otimes \mathbf{B}_v$.

Then, we add the $\mathbf{Z}$ matrix to the 'half' penalty matrix. The rationale is if $\boldsymbol{\beta} = \mathbf{Z}\boldsymbol{\beta}_z$, then $\boldsymbol{\beta}^T\mathbf{S}\boldsymbol{\beta} = \boldsymbol{\beta}_z^T(\mathbf{Z}^T\mathbf{S}\mathbf{Z})\boldsymbol{\beta}_z = \boldsymbol{\beta}_z^T(\mathbf{Z}^T\mathbf{B}^T)(\mathbf{B}\mathbf{Z})\boldsymbol{\beta}_z$. So the 'half' penalty matrix becomes $\mathbf{B}\mathbf{Z}$.

(5) **B_vec**, **B_vec_idx**: matrices that facilitate the coding of composite penalty matrix for the estimation part.

The 'B_vec' column vector is a rearrangement of the three $\mathbf{B}$ matrices, 'B_uv_z', 'B_vu_z', 'B_t_z'. Each row of these matrices are aligned side by side through 'btran()' (and transposed to be a column vector) and then stacked together. We can recover 'B_uv_z', 'B_vu_z', 'B_t_z' from 'B_vec' using information stored in the first three columns of 'B_vec_idx'. The first and second columns ('idx1' and 'idx2') tell us the start and end points in 'B_vec' that mark the section related to each $\mathbf{B}$ matrix. The third column ('idx3') gives us the number of columns each $\mathbf{B}$ matrix has.

The fourth and fifth column ('idx4' and 'idx5') tell us the start and end points that mark the section of $\boldsymbol{\beta}_p = [\lambda_{p,0}, \boldsymbol{\beta}_{p,uv}^T, \boldsymbol{\beta}_{p,t}^T]^T$ penalized by each $\mathbf{B}$ matrix. **Note that 'idx4' and 'idx5' are both shifted by 1 in the code because $\lambda_{p,0}$ is not penalized.**

We have all this information, from 'idx1' to 'idx5', for all the three penalty matrices involved, and they are listed as the three rows of 'B_vec_idx', in the order of $\mathbf{B}_{u|v}$, $\mathbf{B}_{v|u}$, $\mathbf{B}_t$.

Later we can see that it simplifies the code for calculating the derivatives of GCV because we only operate with 'B_vec' and 'B_vec_idx'. So, when we change the penalty structure, we only need to change 'B_vec' and the indices. But, we do not need to modify much, if at all, the code for the actual algorithm.