Practical 3: Smoothing with thin plate splines

Smoothing and function estimation play important parts in applied statistics and data science. One approach combines basis expansions and penalized regression, both techniques with much wider application. In this practical you (working individually) will write R functions for smoothing $\mathbf{x}_i, y_i$ data where $\mathbf{x}_i$ is a vector containing 2 values. The idea is that you have a model:

$$y_i = f(\mathbf{x}_i) + \epsilon_i, \qquad i = 1, \ldots, n$$

where $\mathbf{x}_i$ and y_i are observed, f is an unknown smooth function, and ϵ_i a zero mean error term with variance σ^2. f can be represented using a *thin plate spline* function. To do this we will choose k points $\mathbf{x}_j^*$ spread 'nicely' throughout the the $\mathbf{x}_i$ points (if n is not too large we might just set $k = n$ and set $\mathbf{x}_i^* = \mathbf{x}_i$, otherwise we might randomly sample k of the $\mathbf{x}_i$ points to use as $\mathbf{x}_j^*$ points). Then define a function

$$\eta(r) = \begin{cases} r^2 \log(r) & r > 0 \\ 0 & \text{otherwise} \end{cases}$$

and define $\eta_j(\mathbf{x}) = \eta(\|\mathbf{x} - \mathbf{x}_j^*\|)$. We can represent f as

$$f(\mathbf{x}) = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \sum_{j=1}^{k} \eta_j(\mathbf{x})\delta_j \tag{1}$$

where $\boldsymbol{\delta}$ and $\boldsymbol{\alpha}$ are parameter vectors to be estimated. Typically we choose k to be fairly large, to avoid biasing estimates by using an overly restrictive model, but this may in turn allow the model to overfit. To avoid this the model is usually estimated by minimizing a weighted sum of lack of fit and wiggliness of f:

$$\sum_{i=1}^{n} \{y_i - f(\mathbf{x}_i)\}^2 + \lambda \int \frac{\partial^2 f}{\partial x_1^2}^2 + 2\frac{\partial^2 f}{\partial x_1 \partial x_2}^2 + \frac{\partial^2 f}{\partial x_2^2}^2 dx_1 dx_2$$

λ controls the smoothness of the estimated f. The $\boldsymbol{\delta}$ and $\boldsymbol{\alpha}$ minimizing this turn out to be the minimizers of

$$\|\mathbf{y} - \mathbf{T}\boldsymbol{\alpha} - \mathbf{E}\boldsymbol{\delta}\|^2 + \lambda \boldsymbol{\delta}^T \mathbf{E}^* \boldsymbol{\delta} \text{ subject to } \mathbf{T}^{*\mathsf{T}} \boldsymbol{\delta} = \mathbf{0},$$

where $E_{ij}^* = \eta_j(\mathbf{x}_i^*)$, $E_{ij} = \eta_j(\mathbf{x}_i)$, $\mathbf{T} = (\mathbf{1}, \mathbf{x})$ and $\mathbf{T}^* = (\mathbf{1}, \mathbf{x}^*)$ (here $\mathbf{x}$ is $n \times 2$ and $\mathbf{x}^*$ is $k \times 2$). The constraint is inconvenient, but can be eliminated by re-parameterization. Let

$$\mathbf{T}^* = \mathbf{Q} \begin{bmatrix} \mathbf{R} \\ \mathbf{0} \end{bmatrix}$$

and let $\mathbf{Z}$ be the last $k-3$ columns of $\mathbf{Q}$ (something like `Z <- qr.Q(qr(Ts),complete=TRUE)[,-(1:3)]` would find `Z` explicitly). Then $\boldsymbol{\delta} = \mathbf{Z}\boldsymbol{\delta}_z$ will always meet the constraint for any $\boldsymbol{\delta}_z$ vector. Hence we can re-write the objective to minimize as

$$\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2 + \lambda \boldsymbol{\beta}^T \mathbf{S} \boldsymbol{\beta} \text{ where } \mathbf{X} = [\mathbf{EZ}, \mathbf{T}], \mathbf{S} = \begin{bmatrix} \mathbf{Z}^T \mathbf{E}^* \mathbf{Z} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \text{ and } \boldsymbol{\beta} = \begin{bmatrix} \boldsymbol{\delta}_z \\ \boldsymbol{\alpha} \end{bmatrix}$$

Minimizing this w.r.t. $\boldsymbol{\beta}$ we get

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{S})^{-1}\mathbf{X}^T\mathbf{y}, \;\; \hat{\boldsymbol{\mu}} = \mathbf{X}\hat{\boldsymbol{\beta}} \text{ and EDF} = \text{trace}\{(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{S})^{-1}\mathbf{X}^T\mathbf{X}\}$$

where $\hat{\boldsymbol{\mu}}$ is the model prediction of $E(\mathbf{y})$ and EDF is the *effective degrees of freedom* of the model, which is between 3 and k reflecting the fact that we have constrained the model fit by imposing the penalty during fitting.

A popular method for estimating λ is generalized cross validation, which chooses λ to minimize

$$V(\lambda) = \|\mathbf{y} - \hat{\boldsymbol{\mu}}\|^2 / (n - \text{EDF})^2.$$

This gets rather expensive if we have to compute the EDF and $\hat{\boldsymbol{\mu}}$ using the above formulae, but things can be made much cheaper with some further transformations of the problem. First form a new QR decomposition $\mathbf{X} = \mathbf{QR}$ (here $\mathbf{Q}$ has the same dimension as $\mathbf{X}$, see `?qr.Q` to extract it from the result of `qr(X)`), and then the symmetric eigen decomposition $\mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T = \mathbf{R}^{-T}\mathbf{S}\mathbf{R}^{-1}$. Then it easy to show that $\hat{\boldsymbol{\beta}} = \mathbf{R}^{-1}\mathbf{U}(\mathbf{I} + \lambda\boldsymbol{\Lambda})^{-1}\mathbf{U}^T\mathbf{Q}^T\mathbf{y}$ and EDF $= \text{tr}\{(\mathbf{I} + \lambda\boldsymbol{\Lambda})^{-1}\}$. Now $\mathbf{I} + \lambda\boldsymbol{\Lambda}$ is a diagonal matrix, so if $\mathbf{a} = (\mathbf{I} + \lambda\boldsymbol{\Lambda})^{-1}\mathbf{b}$ then $a_i = b_i/(1 + \lambda\Lambda_{ii})$ - i.e.

there is no expensive matrix inversion to be done. The same goes for the EDF calculation. Hence the computation of the GCV score is now very cheap for each trial λ value.

Your task is to write a function, `fitTPS` for fitting thin plate splines to $\mathbf{x}, y$ data, choosing the smoothing parameter by GCV. Your function will return objects of class `tps`, and you will also write a `plot` method function for this class. You should do this by first writing a function `getTPS` to set up the $\mathbf{X}$ and $\mathbf{S}$ matrices for the spline given an $n \times 2$ matrix `x` and `k` the basis size, which `fitTPS` will then use.

In detail:

1. `getTPS(x,k=100)` should be a function with arguments:

   `x` an $n \times 2$ matrix of location values.

   `k` the number of basis functions to use.

   The function should reset k to n if $k \geq n$, in which case use `x` for the $\mathbf{x}^*$ values. Otherwise select k rows randomly from `x` to give the $\mathbf{x}^*$ values. The function should return a list of named items.

   `xk` a $k \times 2$ matrix of the $\mathbf{x}^*$ values used.

   `X` the $\mathbf{X}$ matrix.

   `S` the $\mathbf{S}$ matrix.

   Information specifying the $\mathbf{Z}$ matrix, so that you can later predict from the fitted thin plate spline. Either the $\mathbf{Z}$ matrix itself, or the `qr` decomposition used to obtain $\mathbf{Z}$ (enabling you to compute with $\mathbf{Z}$ without actually forming it explicitly).

2. `fitTPS(x,y,k=100,lsp=c(-5,5))` should be a function with arguments

   `x` an $n \times 2$ matrix of location values.

   `y` the n vector of values to smooth.

   `k` the number of basis functions to use.

   `lsp` the $\log \lambda$ limits between which to search for the optimal smoothing parameter value.

   The function should use `getTPS` to set up the thin plate spline, then apply the QR and eigen decompositions required to transform the problem to one enabling efficient GCV computation, and then search for the GCV minimizing smoothing parameter over a grid of 100 values evenly spaced on the log scale between `lsp[1]` and `lsp[2]`.

   The function should return an object of class `tps`, which should be a list containing at least the following named items:

   `beta` the best fit $\hat{\boldsymbol{\beta}}$ at the optimal λ value.

   `mu` the corresponding $\hat{\boldsymbol{\mu}}$.

   `medf` the effective degrees of freedom at the optimal λ value.

   `lambda` the vector of 100 λ values searched over.

   `gcv` the corresponding GCV score vector.

   `edf` the corresponding vector of EDFs.

   Other items will be needed to enable predictions from the model.

3. A function `plot.tps` whose first argument is an object of class `tps` as returned from `fitTPS`, which will plot your fitted thin plate spline as a contour plot or a perspective plot (your choice). Note that to predict from the fitted model you can transform from $\hat{\boldsymbol{\beta}}$ to $\hat{\boldsymbol{\alpha}}$ and $\hat{\boldsymbol{\delta}}$ and use (1). Alternatively produce the equivalent of $\mathbf{X}$ for the new $\mathbf{x}$ values at which you want to predict, but using the $\mathbf{Z}$ computed for fitting.

Here is some code to generate example test data and plot the true function (`ff`) underlying it.

```
ff <- function(x) exp(-(x[,1]-.3)^2/.2^2-(x[,2] - .3)^2/.3^2)*.5 +
                  exp(-(x[,1]-.7)^2/.25^2 - (x[,2] - .8 )^2/.3^2)
n <- 500
x <- matrix(runif(n*2),n,2)
y <- ff(x) + rnorm(n)*.1 ## generate example data to fit
## visualize test function ff...
m <- 50;x2 <- x1 <- seq(0,1,length=m)
xp <- cbind(rep(x1,m),rep(x2,each=m))
contour(x1,x2,matrix(ff(xp),m,m))
persp(x1,x2,matrix(ff(xp),m,m),theta=30,phi=30)
```

Working individually you should aim to produce well commented[1], clear and efficient code for the task. The code should be written in a plain text file called `xxxx.r`, where `xxxx` is your student number (i.e. something like `s123456789`). There should be only functions in what you submit, so that when the file is sourced, functions are defined, but nothing else is run. Your solutions should use only the functions available in base R (or packages supplied with base R). Format the code and comments tidily, so that they display nicely on an 80 character width display, without line wraps (or only occasional ones). The work must be completed individually, and you must not share code for this task (you can discuss concepts of course). Note that the university has some automatic tools for detecting when code has been copied between groups (and also from online sources, and from work of students at other universities which has been submitted to the same checking platform). Also, students tend to make very distinctive errors when coding that stand out even if obvious things are done to hide that code has been copied.

The first comment in your code should list your name and student number. Your code file should not refer to this sheet, and should be sufficiently well commented that someone reading it can tell what it is about and the strategies you are using, without reference to this sheet. However, your initial overview comments should be brief - reproducing all the maths in this sheet is not expected, just the bare essentials.

Your code will be subject to some auto-marking. For that reason it is **essential** that you stick exactly to the specification of the functions given, and that your submitted code does nothing but load functions when `source` into R. *You will lose marks if I have to edit your code in order to* `source` *it or run it.* Any tests or examples in your code file should be written into functions, but those functions should not be run when the file is `source`d.

One piece of work - the text file containing your commented R code - is to be submitted on Learn by 12:00 Friday 3rd November 2023. No extensions are available on this course, because of the frequency with which work has to be submitted. Technology failures will not be accepted as a reason for lateness (unless it is provably the case that Learn was unavailable for an extended period), so aim to submit ahead of time.

Marking Scheme: Full marks will be obtained for code that:

1. does what it is supposed to do, and has been coded in R approximately as indicated.
2. produces correct results under a variety of automatic tests that functions written according to the specification should pass.
3. is carefully commented, so that someone reviewing the code can easily tell exactly what it is for, what it is doing and how it is doing it, without having read this sheet, or knowing anything else about the code. Note that *easily tell* implies that the comments must also be as clear and *concise* as possible. You should assume that the reader knows basic R, but not that they know exactly what every function in R does.
4. is well structured and laid out, so that the code itself, and its underlying logic, are easy to follow.
5. is reasonably efficient, meaning that care has been taken to use the matrix decomposition approach given above to keep down the cost of fitting, with some care taken to avoid excessively costly matrix computations.
6. contains no evidence of having been copied, in whole or in part, from other students on this course, students at other universities (there are now tools to help detect this, including internationally), online sources etc.

[1]Good comments give an overview of what the code does, as well as line-by-line information to help the reader understand the code. Generally the code file should start with an overview of what the code in that file is about, and a high level outline of what it is doing. Similarly each function should start with a description of its inputs outputs and purpose plus a brief outline of how it works. Line-by-line comments aim to make the code easier to understand in detail.