APTS Statistical Computing: Assessment 2014/15

The work provided here is intended to take up to half a week to complete. Students should talk to their supervisors to find out whether or not their department requires this work as part of any formal accreditation process. It is anticipated that departments will decide on the appropriate *level* of assessment locally, and may choose to drop some (or indeed all) of the parts, accordingly. So make sure that your supervisor or local organizer of APTS assessment has looked at the assignment before you start, and has told you which parts of it to do. In order to avoid undermining institutions' local assessment procedures the module lecturer will not respond to enquiries from students about this assignment.

Supplementary functions `linear.basis`, `derivation.matrix`, and `qinv` are here: `http://people.bath.ac.uk/~fl353/apts/functions.R`

This assignment is about non-parametric "smoothing". In order to obtain a coherent statistical framework that is essentially representation independent, we first define a penalty function for a function $f(x)$ on $[a, b]$ as

$$\int_a^b f^{(p)}(x)^2 \, \mathrm{d}x,$$

where $f^{(p)}(\cdot)$ is the pth derivative of $f(\cdot)$, $p \geq 0$. This penalty is closely related to a Bayesian prior based on the stochastic intrinsic random process model $f^{(p)}(x)\,\mathrm{d}x = \mathrm{d}W(x)$, and expresses our belief about the 'wiggliness' of the function $f(\cdot)$.

Since $f(\cdot)$ is an infinite dimensional object, in the computer we need to express it with a finite dimensional representation, and here we choose a piecewise linear basis expansion. Consider an increasing sequence of K x-values, $k_1, k_2 \ldots k_K$. A piecewise linear function on $[k_1, k_K]$ can be defined as

$$f^K(x) = \sum_{i=1}^K b_i^K(x) f_i^K$$

where f_i^K is a coefficient (interpretable as the value of $f_K(k_i)$), and $b_i^K(x)$ is a *basis function*,

$$b_i^K(x) = \begin{cases} (k_{i+1} - x)/(k_{i+1} - k_i) & k_i \leq x < k_{i+1}, \quad i < K \\ (x - k_{i-1})/(k_i - k_{i-1}) & k_{i-1} \leq x < k_i, \quad i > 1 \\ 0 & \text{otherwise.} \end{cases}$$

The higher K is, and the smaller $k_{i+1} - k_i$ are, the more flexible $f^K(x)$ will be. If we then take $f_i^K = f(k_i)$, the finite dimensional representation $f^K(\cdot)$ closely approximates the continuous function $f(\cdot)$.

The continuous penalty for $f(\cdot)$ can be approximated using the coefficients of $f^K(\cdot)$, as the quadratic form

$$\int_a^b f^{(p)}(x)^2 \, \mathrm{d}x \approx \sum_{i=1}^{K-p} \left(\sum_{j=1}^K D_{ij} f_j^K \right)^2 \Delta_i = \mathbf{f}^\top \mathbf{D}^\top \operatorname{diag}(\Delta_j) \mathbf{D} \mathbf{f}$$

where the matrix $\mathbf{D}$ and vector $\boldsymbol{\Delta}$ can be obtained with the function `derivation.matrix` in the supplement.

1. Examine the `mcycle` data from R package `MASS`. Use the functions `linear.basis` from the supplement and the `lm` command in R to fit non-penalised piecewise linear models of the form

$$\texttt{accel}_i = f^K(\texttt{times}_i) + \epsilon_i$$

to the data. Produce a plot comparing models with $K = 10, 20, 30$ and 40, and equally spaced k_i values. i.e. produce a scatterplot of `accel` against `times`, and overlay the curves showing $\widehat{f^K}(\texttt{times})$ against `times` for the different K values. Transform the `linear.basis` output into a non-sparse matrix with `as.matrix` before using it with `lm` (but *not* in the questions below). Note that some of the coefficients may be estimated as `NA`!

Use the function `derivation.matrix` from the supplement to evaluate what the penalty for derivatives of order $p = 2$ for each fit would be.

2. Controlling the complexity of the piecewise linear model by varying K is not very satisfactory, and easily leads to overfitting and ill-posed estimation problems. We therefore instead introduce a penalty on the second order derivatives, as outlined in the introduction on the previous page, which only requires that the knot equence is dense enough to allow a good approximation of the continuous function.

If y is the response variable, the model can be fitted by minimizing a penalized version of the normal least squares objective,

$$L(f(\cdot), \lambda) = \sum_{i=1}^{n} (y_i - f(t_i))^2 + \lambda \int f^{(2)}(x)^2 \, \mathrm{d}x$$

w.r.t. $f(\cdot)$. λ is a smoothing parameter, controlling the tradeoff between smoothness of f and close fit to the data y_i. It can readily be verified that in the finite dimensional representation of f with coefficients $\boldsymbol{\beta}$ this can be written as

$$L(\boldsymbol{\beta}, \lambda) = \|\mathbf{y} - \mathbf{A}\boldsymbol{\beta}\|^2 + \lambda \boldsymbol{\beta}^\mathsf{T} \mathbf{Q}_\mathrm{f} \boldsymbol{\beta},$$

where the elements of the matrix $\mathbf{A}$ are $A_{ij} = b_j^K(t_i)$, and $\mathbf{Q}_\mathrm{f} = \mathbf{D}^\top \operatorname{diag}(\Delta_j)\mathbf{D}$.

Show that, for a given λ, the formal expression for the 'hat matrix' $\mathbf{H}$ such that $\hat{\boldsymbol{\mu}} = \mathbf{H}\mathbf{y}$ ($\boldsymbol{\mu} = \mathbb{E}(\mathbf{y})$), is given by $\mathbf{H} = \mathbf{A}\mathbf{Q}_\lambda^{-1}\mathbf{A}^\mathsf{T}$, where $\mathbf{Q}_\lambda = \mathbf{A}^\mathsf{T}\mathbf{A} + \lambda \mathbf{Q}_\mathrm{f}$.

3. Produce a plot like the one from part 1, but now illustrating the smooth fits given by several alternative λ values (choose these to give results varying from very smooth to very wiggly, and include something in the middle that looks reasonably sensible for the data). Don't worry about stable or efficient computation for the moment, and use $K = 40$ with equally spaced k_i values (but do use `solve(matrix, vector)` where appropriate instead of evaluating matrix inverses, and verify that the Cholesky factor of $\mathbf{Q}_\lambda$ is sparse, with $\mathcal{O}(K)$ non-zero elements).

4. One objective method for choosing the smoothing parameter λ is Generalized Cross Validation (GCV). This selects the λ value that minimizes

$$\mathcal{V}(\lambda) = \frac{n\|\mathbf{y} - \mathbf{H}\mathbf{y}\|^2}{(n - \operatorname{tr}(\mathbf{H}))^2}$$

(the dependence of the r.h.s. on λ is via $\mathbf{H}$). In terms of K and n (the length of $\mathbf{y}$), what order would the computational cost of evaluating $\mathcal{V}$ be, for any trial value of λ, if the expression for $\mathbf{H}$ given in part 3 was used naively, without using sparse matrix operations?

A recursion formula based on the Cholesky decomposition $\mathbf{Q}_\lambda = \mathbf{R}^\mathsf{T}\mathbf{R}$, implemented in `qinv` in the supplement[1], can be used to obtain the elements of $\mathbf{Q}_\lambda^{-1}$ corresponding to the non-zero elements of $\mathbf{Q}_\lambda$. Call this partial inverse matrix $\mathbf{S}_\lambda$. The cost of computing $\mathbf{R}$ and $\mathbf{S}_\lambda$ is $\mathcal{O}(K)$ for the 1D penalty model, and results in $\mathcal{O}(K)$ nonzero elements. Using the identity $\mathrm{tr}(\mathbf{AB}) = \mathrm{tr}(\mathbf{BA})$, show that $\mathcal{V}$ can be calculated with cost $\mathcal{O}(K + n)$.

Note that $\mathrm{tr}(\mathbf{H})$ can be interpreted as the *effective degrees of freedom* of a penalized fit.

5. Write an R function with arguments `y` and `x` containing the response variable to smooth, and the covariate to smooth with respect to (`accel` and `times` in the `mcycle` data example). The function should also take an argument `K` giving the number of basis functions to use.

   The function should smooth the x, y data using a penalized piecewise linear smoother of the sort introduced above. It should choose the smoothing parameter by GCV, by evaluating $\mathcal{V}$ for each $\log\lambda$ value generated by `seq(-10,10,length=200)`, and selecting the smoothing parameter yielding the smallest $\mathcal{V}$. The function should have $O(K + n)$ computational cost overall.

   The function should return a list containing 4 vectors: the fitted values $\hat{\boldsymbol{\mu}}$ corresponding to the GCV optimal smooth model, a 200-vector of $\log\lambda$ values, and two vectors of corresponding GCV scores $\mathcal{V}$ and effective degrees of freedom $\mathrm{tr}(\mathbf{H})$.

   Some operations require transfroming small `Matrix` objects into ordinary vectors. Use `as.matrix()` and `as.vector()` for this, but be careful not to accidentally produce any large dense matrices!

6. After testing your routine, produce a scatterplot of `accel` vs. `times` with the GCV optimal smooth estimates overlaid, alongside a plot of the GCV score against effective degrees of freedom.

[1]A *much* faster implementation is available as `inla.qinv()` in the INLA packages, see `http://r-inla.org/`